THE UNIVERSITY
of EDINBURGH

CAMP-BDI
An approach for Multiagent Systems Robustness through Capability-aware Agents Maintaining Plans

Alan Gordon White

Doctor of Philosophy

Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh

March 2017

Abstract

Rational agent behaviour is frequently achieved through the use of plans, particularly within the widely used BDI (Belief-Desire-Intention) model for intelligent agents. As a consequence, preventing or handling failure of planned activity is a vital component in building robust multiagent systems; this is especially true in realistic environments, where unpredictable exogenous change during plan execution may threaten intended activities.

Although reactive approaches can be employed to respond to activity failure through replanning or plan-repair, failure may have debilitative effects that act to stymie recovery and, potentially, hinder subsequent activity. A further factor is that BDI agents typically employ deterministic world and plan models, as probabilistic planning methods are typical intractable in realistically complex environments. However, deterministic operator preconditions may fail to represent world states which increase the risk of activity failure.

The primary contribution of this thesis is the algorithmic design of the CAMP-BDI (Capability Aware, Maintaining Plans) approach; a modification of the BDI reasoning cycle which provides agents with beliefs and introspective reasoning to anticipate increased risk of failure and pro-actively modify intended plans in response.

We define a capability meta-knowledge model, providing information to identify and address threats to activity success using precondition modelling and quantitative quality estimation. This also facilitates semantic-independent communication of capability information for general advertisement and of dependency information - we define use of the latter, within a structured messaging approach, to extend local agent algorithms towards decentralized, distributed robustness. Finally, we define a policy based approach for dynamic modification of maintenance behaviour, allowing response to observations made during runtime and with potential to improve re-usability of agents in alternate environments.

An implementation of CAMP-BDI is compared against an equivalent reactive system through experimentation in multiple perturbation configurations, using a logistics domain. Our empirical evaluation indicates CAMP-BDI has significant benefit if activity failure carries a strong risk of debilitative consequence.

Lay Summary

Intelligent agents typically execute plans (partially or fully ordered activity sequences) to achieve their goals. Realistic environments may see changes during activity execution that threaten the success of planned future activities, requiring agents implement robustness strategies to prevent consequent goal failure. Existing implementations typically employ reactive robustness approaches by responding to activity failures through behaviour such as replanning or plan repair. However, in realistic environments the efficacy of reactive methods may be hindered if activity failure itself risks debilitative consequences.

This thesis contributes the CAMP-BDI (Capability Aware, Maintaining Plans) approach; a method for proactive robustness, built upon the widely used Belief-Desire-Intention (BDI) model for rational agent reasoning. We first describe the provision of agents with knowledge that models their social responsibilities (i.e. their obligations and dependencies towards others) and capabilities (i.e. which activities they can perform, and how well). We then describe use of this knowledge within algorithms for *proactive* robustness, which detect threats to planned future activities from environmental change – responding to avoid failure with appropriate plan modification. The application of this robustness behaviour is described within the context of both individual agents and plan executing teams.

Our experimental evaluation showed CAMP-BDI as offering improved robustness over typical reactive methods where activity failure risked debilitative consequences. We suggest CAMP-BDI offers a valuable method for aiding agent robustness, and can complement existing robustness approaches, within realistic environments.

Acknowledgements

This thesis represents the outcome of a long, hard journey which would not have been possible without the help and advice of my primary supervisor, Professor Austin Tate. In equal measure, I owe a debt of thanks to Dr Stephen Potter and Dr Michael Rovatsos, my secondary supervisors in 2010-2012, and 2012 onwards respectively. I would also like to thank my examiners, Dr Ron Petrick and Prof. Michal Pěchouček, for their invaluable input and suggestions towards improving the quality of this thesis.

On a technical level, I'd like thank Prof. Jomi F. Hubner and Dr. Rafael H. Bordini, for their assistance with understanding the Jason framework during my implementation work. Although I ultimately opted for an alternate method, Professor Felipe Meneguzzi provided valuable advice and help regarding the JavaGP planner implementation used in the Peleus system.

I also acknowledge my former cohort under EADS sponsorship, Dr Erich Zywssig, and my erstwhile colleagues at EADS Innovation Works during my funding period. It would also be remiss not to extend my best wishes to everyone in IF 2.35 - past and present - for putting up with me and my sometimes incoherent whiteboard writings.

My wife Dr. Poay Ngin Lim served, and continues to serve, as a source of help, advice and support. Last, but by no means least, I'd like to thank my daughter Juno – an inexhaustible source of both determination and inspiration.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Alan Gordon White*)

Table of Contents

Chapter 1

Introduction

Intelligent agents are increasingly employed in challenging realistic environments, such as military, emergency response, aerospace, or power management systems. This thesis focuses upon Multiagent Systems (MAS) robustness, specifically with regard to the Belief-Desire-Intention (BDI) model (Rao and Georgeff [1995]). As plans are important in defining the rational behaviour of intelligent agents (Pollock [1999]), mitigation against the failure of plans and planned activity represents an important component of agent robustness.

We target realistic environments where successful plan execution may be threatened by exogenous change during execution – contradicting the beliefs under which plans were formed and leading to activity failure. Existing BDI architectures typically employ reactive approaches to handle failure, such as replanning. However, activity failure may risk lasting debilitative consequences – which can hinder reactive mechanisms, hamper subsequent goal achievement, and potentially extend to loss of material resources – or human life.

This thesis presents the CAMP-BDI approach for plan execution robustness through proactive plan modification (referred to as *maintenance*). We contribute algorithms for performing plan maintenance, combined with a supporting architecture to provide knowledge for introspective reasoning and a policy mechanism which supports flexibility through runtime modification of key variables. We extend locally defined behaviour to the distributed case, using structured communication and provision of contractual knowledge to allow decentralized maintenance.

Our experimental evaluation shows CAMP-BDI can offer improved robustness in environments where failure risks debilitative consequences, by preventing negative post-failure states that can hinder reactive recovery. These results also show improved planning and activity costs over reactive methods, with excess absolute costs mitigated by improved robustness (i.e. avoiding expenditure on ultimately failed goals). The maintenance policy mechanism we define also allows further cost mitigation in practical implementations – expected to employ CAMP-BDI to complement reactive methods – through tailoring the sensitivity of maintenance for specific agent and/or activity types.

1.1 Background and Context

This work addresses *robustness* in MASs formed of BDI agents acting within realistic environments, in the context of plan execution. Multiagent systems are employed in realistic domains including aerospace (Šišlák *et al.* [2010]), military (Hoogendoorn *et al.* [2006]) or emergency response (Zhan and Chen [2008]). The inherently componentized nature of a MAS can be ideal in such environments due to supporting techniques such as a decomposition, reducing the knowledge and capability requirements of individuals whilst allowing co-ordinated behaviour through virtual organizations such as teams or *holons* (Fischer *et al.* [2003]). However, realistic environments can present agents with difficult characteristics (Russell and Norvig [2009]) by being dynamic, inaccessible, or stochastic. These environments may be *hostile*, where agents risk debilitation from exogenous change or following activity failure.

Planning holds critical importance within BDI agent rationality, making mitigation against planned activity failure a key aspect of agent robustness. As MASs frequently require goals be achieved through the coordinated efforts of agent *teams*, mitigation must consider distributed plans, especially as individual failure can have reciprocal impact across a multiagent team - such as a scout helicopter's failure to warn cohorts of a threat leading to consequent ambush. Planning often models the environment in deterministic terms (Meneguzzi *et al.* [2010]), as resultant problems and plans are typically easier to understand and visualize (and more tractable) than with probabilistic approaches (such as Markov decision processes). However, this can only *approximate* the complexity of realistic environments; plans will possess hidden probabilistic or qualitative components, as states deemed not *significant* enough to represent in opera-

tor preconditions can still influence the outcome of execution (McCarthy [1958]).

Toyama and Hager [1997] categorize robustness approaches as *ante* or *post*-failure; the former seeking to resist failure, the latter responding to it. Architectural frameworks for BDI agents (Bordini and Hübner [2006], dInverno *et al.* [2000]) frequently employ the latter for plans, using repair or replanning to recover from failure. Reactive response can be justified as offering greater certainty and efficiency through occurring only when failure is *definite*; proactive/pre-emptive systems may risk false positives and negatives – performing ultimately unnecessary mitigation activity, or failing to identify (and prevent) failures.

However, reactive recovery entails failure must occur before any mitigation is performed; in a realistic environment, activity failure may be accompanied by debilitative consequences that increase the difficulty of recovery. In certain domains these consequences may be severe - extending to potential loss of life (e.g. in military or emergency response domains). Finally, as realistic domains are continuous, failure consequences may also hinder subsequent activity and impact the longer term efficacy of the MAS. Existing proactive approaches often involve providing flexibility to modify or refine plans during execution. Continual planning approach defers planning decisions until (close to) their execution, but can risk inadvertently stymieing longer term goal achievement. Conformant or conditional planning attempts to form plans which prevent failure arising from uncertainty, but require abstraction for tractability in complex domains – reducing their ability to prevent *all* failures.

1.2 Motivating Example

We wish to improve robustness in realistic environments which are subject to unpredictable exogenous change and where activity failure risks debilitative consequences. An example of these properties can be found in *Transport* domains (Figure 1.1), where goals are to transport cargo objects between locations. MAS robustness can be viewed in terms of the number of deliveries performed (system goals achieved) against increasing environmental perturbation (rate of exogenous change). Our approach assumes exogenous changes are detected by agents and subsequently represented in their Beliefs, reasoning that event types directly impacting agent activity are likely to be identified and modelled during system design and domain analysis.

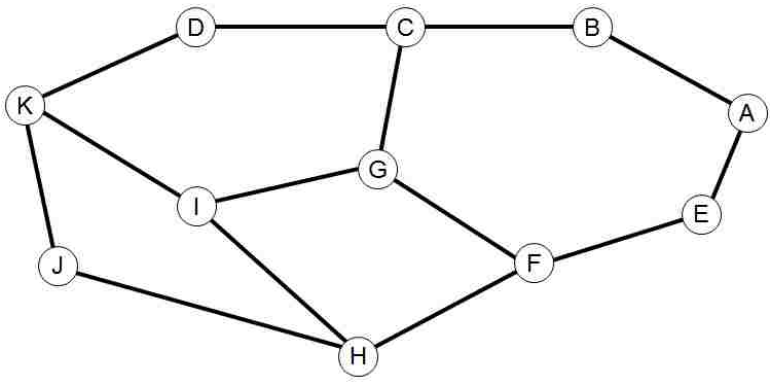


Figure 1.1: Example geography; nodes represent locations and connections the bi-directional roads between them.

For example, *Truck1*, situated at H, adopts an (intended) goal to deliver specified cargo to K. This leads to an intention to perform the following plan:

1 Travel to A (through a sub-plan of movements; $H \rightarrow F \rightarrow E \rightarrow A$)
2 Load cargo object *cargo1* at A
3 Travel to K (following route $A \rightarrow B \rightarrow C \rightarrow D \rightarrow K$)
4 Unload *cargo1* at K.

We are concerned with preventable, anticipatable failure stemming from exogenous change contradicting the beliefs held at the time of (and used for) intention formation. For example, preconditions of planned activities may be violated by change before execution, such as a landslip blocking $E \rightarrow A$. Alternatively, exogenous change could increase failure risk without explicitly violating preconditions – such as $E \rightarrow A$ becoming slippery, increasing the risk of *Truck1* skidding and crashing on $E \rightarrow A$ without preventing it's use *outright*.

Failure may have debilitative consequences; for example, if *Truck1* crashes on the slippery $E \rightarrow A$ it may damage both itself and any cargo being carried. The resulting post-failure may render recovery more difficult and/or costly, if not impossible; e.g. if *Truck1* requires additional agents to return to the road, or cargo destruction forces use of some alternative, more distant, cargo resource. Debilitative consequences may persist and threaten *future* goals – damage to *Truck1* can risk that agent failing future activities or being unable to act atall.

We suggest agents can be embodied with *capability* meta-knowledge, allowing introspective reasoning to identify where exogenous changes threatens activities within intended plans. This can allow agents to determine both when to attempt plan modification to address that threat, and which modifications are required.

Our prior example described *Truck1* suffering movement failure due to $E \rightarrow A$ being slippery. Our desired behaviour (depicted in Figure 1.2) is that, when *Truck1* becomes aware of the change in road conditions, it employs capability knowledge to identify an increased risk of failure for travel along $E \rightarrow A$. *Truck1* then uses that capability knowledge to guide appropriate plan modifications, forming an alternate route to avoid $E \rightarrow A$.

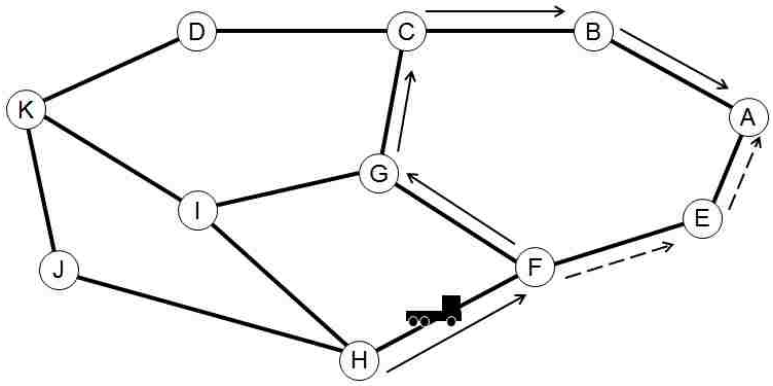


Figure 1.2: Example maintenance behaviour. *Truck1*, currently on road $H \rightarrow F$, detects slippery conditions on $E \rightarrow A$. It modifies it's plan to use an alternate route (solid lines), avoiding the now-slippery (riskier) road used in the original plan (dashed lines).

MASs frequently employ multiagent teams (performing distributed plans), meaning individual agent failures reciprocally impact others in the same team. For example, a *Commander* agent can adopt a goal to satisfy a cargo request at K; resulting in a plan that selects *Cargo1* for delivery, and *Truck1* as the delivery agent – leading to subsequent dependency relationships. Failure on *Truck1*'s part entails failure of *Commander*'s dependant activity. Robustness approaches must consider failure on both local *and* team levels; i.e. if *Truck1* cannot prevent or recover from failure, *Commander* must adapt accordingly.

In a realistic environment, the world is likely to be be highly complex and only

partially observable – rendering centralized robustness approaches impractical. Indeed, environmental complexity is a frequent motivation for employing a MAS (Sycara [1998]), as it allows a 'divide and conquer' approach reflecting the distribution of knowledge in the environment. A decentralized, distributed approach becomes desirable, allowing system-wide behaviour to be accomplished through structured local behaviour. We suggest this be accomplished by making agents aware of obligations to, and dependencies upon, other agents to perform tasks – enabling obligants to communicate the status of their obligations to dependants, and dependants to use that information to identify whether they should maintain their corresponding dependant plan in response.

An example of this behaviour occurs if *Truck1* is unable to find a route to pick up *Cargo1* at *A* (e.g. if every possible route risks non-deterministic failure). Due to the obligation accepted from *Commander*, *Truck1* would inform that dependent agent that it had a reduced chance of success, and had been unable to improve its local plan (i.e. local maintenance had failed). *Commander* could then modify it's dependant *local* plan; such as to use a *Helicopter* agent not constrained by road conditions, or another *Truck* in a position to form a less risky route.

This behaviour is intentionally analogous to repair of Hierarchical Task Network (HTN) plans (Tate [1977]). HTN plan repair can be summarised as escalating (from the most primitive level) re-refinement of abstract tasks, until a suitable refinement is found. We view a distributed multiagent team in similar terms, where the plan associated with an intended goal is analogous to a selected refinement for an abstract task [1]. Our desired *system level* behaviour is for repair to proceed in a similar manner, with maintenance responsibility escalating up the hierarchy (from the lowest – i.e. most specialized – agent) until an agent successfully modifies it's plan, counteracting the threat to the distributed intention.

Typical BDI implementations adopt a reactive approach, responding to activity failure with replanning or plan repair. Our motivation stems from where failure both can arise from exogenous change after plan formation, and risks debilitative consequences – hindering the effectiveness of reactive recovery. Agents could, intuitively, use con-

[1] In this context, selecting an obligant to perform a delegated activity can be viewed as selecting an agent to *refine* that task.

tinuous replanning – forming a new plan (for the intended goal) after every activity execution using the most current set of beliefs. However, this risks significant computational cost due to frequent planning operations. Continuous replanning also risks necessary material or agent resources not being reserved and consequently lost to contention, *or* increased communications costs if they are; the agent can identify required resources for the *current* intended plan, but these risk cancellation with subsequent revision.

1.3 Research Objectives

The aim of this thesis is *to identify and design an approach towards plan execution robustness for BDI agents, based upon proactive modification of plans to avoid anticipated (risk of) activity failure*. This follows our hypothesis (Section 1.4) that activity failure avoidance can offer robustness benefits, where *reactive* recovery is hindered by debilitated states arising as a consequence of failure. Our research objectives were as follows:

1. To determine knowledge requirements for agents to anticipate where an intended activity risks failure, following exogenous change.
2. To provide BDI agents with behaviour to anticipate activity failure and avoid resultant intention failure through proactive plan modification.
3. To provide agent team level behaviour that enables proactive robustness within the context of distributed plan execution.
4. To show proactive plan modification can improve robustness over reactive approaches, within environments whose properties befit our motivation.

1.4 Hypothesis

We hypothesized that, *in realistic environments where failure risks debilitative consequences, a proactive approach of pre-emptive plan modification can improve robustness over a purely reactive approach.*

Drawing from our literature review, we defined robustness as the ability to achieve

system goals (i.e. of a hierarchically decomposing, multiagent team) despite environmental perturbation.

Our consideration of pre-emptive behaviour, combined with support of longer term planning, led to us to form several *design hypotheses* during our specification and design:

- Agents can be embodied with capability knowledge to represent both activities they can perform themselves, and those they can delegate to others.
- The resultant capability model can be used to determine when plan failure is threatened, and to direct mitigation behaviour.
- Localized behaviour can be extended to perform decentralized, distributed maintenance through knowledge-sharing within, and communication of, dependency contracts.
- Policies – sets of behavioural constraints, applied to sets of agent-capability pairs – can be used to tailor agent maintenance behaviour during runtime, allowing adaptation to changing knowledge of the agent and environment.

1.5 Contributions

This thesis contributes the design of the CAMP-BDI (**C**apability **A**ware, **M**aintaining **P**lans) approach; an algorithm and supporting architecture for pre-emptive plan modification to avoid failure, described as an extension to the generic BDI reasoning cycle.

We can divide this contribution into the following parts, each representing a significant individual elements of our design;

- An algorithm for performing pre-emptive maintenance, based upon modifying intended plans in response to exogenous change during execution.
- The capability meta-knowledge model, used to represent external capability and dependency contract information in a multiagent team and facilitate our maintenance algorithm.
- Description of structured messaging behaviour to extend individual agent maintenance behaviour into the distributed context of a plan-executing team.
- A policy based mechanism allowing runtime modification of key variables and constraints used by the algorithm, allowing tailoring of maintenance behaviour

and providing a framework for potential extension to support generalization and agent reuse across different environments.

1.6 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 discusses existing domains, explaining our motivation and introducing the *Cargoworld* domain used for description and evaluation of CAMP-BDI.

Chapter 3 overviews agent system concepts, including the BDI approach; this is followed by discussion of approaches for agent robustness in **Chapter 4**.

The importance of *plans* in guiding BDI behaviour leads to consideration of automated planning in **Chapter 5**, to determine knowledge requirements for our intended behaviour. **Chapter 6** considers approaches for plan robustness under uncertainty, which suggest conceptual mechanisms applicable to our work.

Chapter 7 specifies desired behaviour using the *Cargoworld* domain. The following three chapters detail our primary contribution. **Chapter 8** defines the supporting architecture, employed by the core *maintenance* algorithms detailed in **Chapter 9** – with **Chapter 10** describing extension to perform distributed maintenance.

Chapter 11 and **Chapter 12** discuss our evaluation. **Chapter** 13 considers how CAMP-BDI may be applied towards domains outside *Cargoworld*, before **Chapter 14** concludes this thesis.

Chapter 2

Motivating Domain

Uhrmacher and Swartout [2003] noted that empirical investigation of a MAS requires a suitable domain; this chapter discusses a number of domains and environments where agents – and teams – could benefit from proactive failure mitigation. This leads to the choice of domain for the current research.

2.1 Domain and Environment Properties

Russell and Norvig [2009] classify the operating environment for a multiagent system along several axes;

- **Accessible** (*Fully Observable*) or **Inaccessible** (*Partially Observable*) – in an accessible environment, the sensory apparatus can perceive the entire world state when required; conversely, inaccessible environments require agents to *preserve* knowledge of changes in world state for use in reasoning.
- **Deterministic** or **Non-deterministic** – in a deterministic environment the next state of the environment is always determined by the current state and the next action(s) of the agent(s) therein. In non-deterministic environments actions may have multiple potential outcomes, with exogenous events also potentially altering world state. Deterministic but inaccessible environments may appear non-deterministic to agents due to limited visibility of all action outcomes.
- **Episodic** or **Non-episodic** – in an episodic environment, agents experience discrete 'episodes' of perception followed by action. Action quality depends solely upon the current episode; any consequences will not persist into future episodes.
- **Static** or **Dynamic** – a dynamic environment is one where world state may change over time, including while an agent reasons. In a static environment,

conversely, time constraints are not a concern upon agent reasoning. An environment is *semi*-dynamic if it does not change with time, but the performance score of the agent does (e.g. if slow decision making is penalized).

- **Discrete** or **Continuous** – discrete domains contain a limited number of clearly defined percepts and actions; continuous domain percepts cover continuous ranges of values.

A realistic domain is inaccessible, non-deterministic (stochastic), non-episodic, dynamic and continuous. Our approach targets the non-episodic, non-deterministic and dynamic characteristics – specifically where exogenous change occurs unpredictably during plan execution. We assume the domain can be reduced to deterministic terms in order to employ classical planning; but also that other world states – not significant enough to represent in deterministic preconditions – can *influence* activity outcome (i.e. increasing risk of failure). We also assume non-determinism through exogenous change and potential debilitative failure consequences (which may not be sufficiently known to model in deterministic terms), where non-episodic characteristics entail activity effects – including any post-failure debilitation – may impact *future* activities.

2.2 Example IPC Domains

The International Planning Competition (IPC), organized within the International Conference on Automated Planning and Scheduling (ICAPS)[1], evaluates the performance of planners in various domains using metrics including plan generation time, activity or temporal cost, or achievement of optional ('soft') goals. Although IPC domains are orientated towards testing *planners* rather than the robustness of plan executing agents within that domain, they can still be related (Crosby *et al.* [2014]) to operating environments of plan-using MASs and may provide useful guidance regarding plan formation and execution scenarios. This section describes a number of domains from IPC-4 (Edelkamp *et al.* [2011]) and IPC-5 (Dimopoulos *et al.* [2006]), which we informally classify into *Space* and *Transport* types, to consider how extension in a realistic manner may introduce properties relevant to our motivation.

[1] `http://www.icaps-conference.org`

2.2.1 Space Domains

IPC-3 introduced the *Satellite* (later extended in IPC-4) and *Rovers* domains (Fox and Long [2003]), both derived from NASA scenarios. The *Satellites* domain involves a 'constellation' of co-operating satellites with heterogeneous sensors, modelling their fuel, data capacity and temperature properties. The planner must find an optimal route for satellites to travel to observation targets, transmit data to earth-bound ground stations – either directly or via a cohort – within defined time bounds and avoiding overheating from direct sunlight. The *Rovers* domain depicts multiple autonomous rovers exploring the surface of Mars. Again, the planner must form routes between waypoints to perform appropriate information gathering and to allow (line-of-sight) transmission of resultant data to a lander. An extended metric version introduced power constraints, where Rovers halt to recharge batteries and must co-ordinate to minimize overall recharging time.

In both domains, we can envisage plausible extensions to include exogenous change and debilitation cases where proactive behaviour would hold intuitive benefits – particularly as (given their location) it would be inherently difficult to send resources and equipment to repair Satellites or Rovers following post-failure damage. For example, a Satellite could suffer fuel loss from micro-meteor impact (or have less fuel than expected due to modelling errors), causing failure of an orbital manoeuvre and leaving that Satellite at risk of further orbital degradation and destructive re-entry. Reactive handling would occur *after* fuel had been expended to the point of failure, potentially leaving that Satellite without sufficient remaining fuel to recover (and no suitable agents near enough to assist). Proactive behaviour, conversely, could anticipate failure risk and ensure the satellite refuelled in advance to mitigate that risk.

In another example, a Rover could discover an intended (planned) route is more difficult than expected by sensing an area of softer sand than believed at route formation. Reactive failure would occur after reaching that location and failing – potentially leaving the Rover stranded in soft ground and either expending excessive energy to escape or depending upon (potentially unavailable) other Rovers to assist. Proactive robustness mechanisms could modify plans to perform preventative behaviour, such as by re-routing over harder ground, or recharging to ensure full batteries to (assist) escape if stuck.

2.2.2 Transport Domains

Transport or *mobile problem* type domains are commonly featured within IPC competitions (Edelkamp *et al.* [2011]), and identified by Long and Fox [2000] as *"a common feature of planning problems, whether as a central or incidental component"*. These domains can be characterised as concerned with achieving some goal requiring correct formation and traversal of a route plan (i.e. to arrive at that destination, deliver cargo, or perform some other activity), and hold interesting properties in terms of both the factors influencing activity success and the potential consequences of failure.

In the *Trucks* domain from IPC-5, planners must find a minimal cost plan to deliver packages to set locations using actions to move a truck, load a package onto a truck, unload a package, or deliver a package (i.e. 'consume' that package to satisfy a request). Trucks are constrained by their cargo storage space and delivery time deadlines. The *DriverLog* domain is similar to *Trucks*, but introduces an additional route-finding problem of guiding *drivers* to appropriate Trucks. This means plan efficiency must also consider the use of Drivers to enable Truck movement (Gregory and Lindsay [2007]), and suggests obvious similarities to multiagent domains requiring co-operation between heterogeneous agents.

We can envisage common scenarios for both these domains where proactive behaviour – i.e. failure prevention – is beneficial. For example, trucks could accumulate wear and tear through travel, with accompanying degradation of performance. A reactive approach would only respond once degradation led to failure – but such a scenario might see debilitation such as mechanical damage from over-fatigue, or skidding then crashing. These consequences would increase the difficulty of recovery (if not rendering it impossible) and potentially threaten future activity, by rendering that truck useless until recovered and repaired. In contrast, a proactive approach could identify the increasing risk from wear and tear (provided this can be sensed or inferred) and – when a certain threshold is met – modify the plan appropriately to perform repairs or delegate to an alternate truck.

Other scenarios may be envisaged; such as route modification to avoid roads rendered dangerous by partial flooding or ice. A proactive approach could reduce *backtracking* costs by making earlier changes, compared to reactively responding upon

reaching that road and being unable to execute the planned activity (or beginning execution and failing midway). In *DriverLog*, this can extend to scenarios where Drivers cannot reach their assigned Truck in time. A reactive system would only respond once that Driver had definitively failed to reach the truck; but a proactive system could invoke compensatory behaviour once it was *sufficiently unlikely* the Driver would succeed – allowing earlier response to identify and assign an alternative. This could also improve the ability to reassign the delayed Driver, if movement was aborted in a more convenient (central) location for reaching *other* Trucks.

The *Triangle-Tireworld* domain (Little and Thibaux [2007]), featured in the probabilistic track of IPC-4, presents an environment where cars move from a start to an end location along unidirectional roads (Figure 2.1). Each location is associated with a certain probability of a flat tyre occurring when travelled through; certain locations hold a spare tyre that may be fitted to the car (extension in IPC-5 added a probability of failure for fitting a spare) *or* stored for future use.

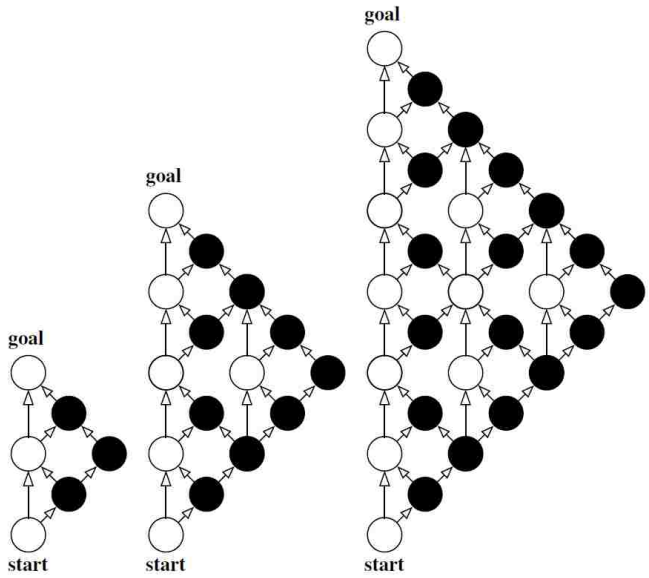


Figure 2.1: Triangle-Tileworld, from Little and Thibaux [2007]. Black circles represent locations *with* a spare tyre, white locations without, and arrows unidirectional roads.

The planner must account for the possibility of a burst tyre by both travelling through locations with spares, and loading spares for future use. Although not preemptive behaviour – spare tyres only have utility *after* a burst tyre results in failure – we draw interest from the generation of plans which *prepare* for possible failure by

ensuring burst tyres can be replaced. The domain does not consider exogenous change – for example, if another car can remove spare tyres from locations, modifying the probability of success for plans formed based upon initial state tyre locations.

2.3 Example Multiagent Experimentation Domains

This section overviews example domains and environments from prior multiagent experimentation. Such domains may employ varying degrees of abstraction, from the simplified *Tileworld* to realistic environments like *Pacifica* or *Robocup Rescue*.

2.3.1 Tileworld

The *Tileworld* domain (Pollack and Ringuette [1990]) presents a grid based environment, used as an abstract agent testbed (Figure 2.2). Agents hold goals to pick up and move a set, variable number of tiles into holes in the environment; further constraints (e.g. shape) may impact the utility of placing a particular tile in a given hole. Tileworld has been revised over time with more realistic properties – such as resource levels or multiagent activity (Ephrati *et al.* [1995])– and to vary in dynamism, uniformity of tasks and movement speed. Although Hanks *et al.* [1993b] argue Tileworld is sufficiently extensible to cover a variety of evaluation requirements, Lees [2002] observes such modification may be overly driven by the specific agent or behaviour being tested – reducing the applicability and generality of results.

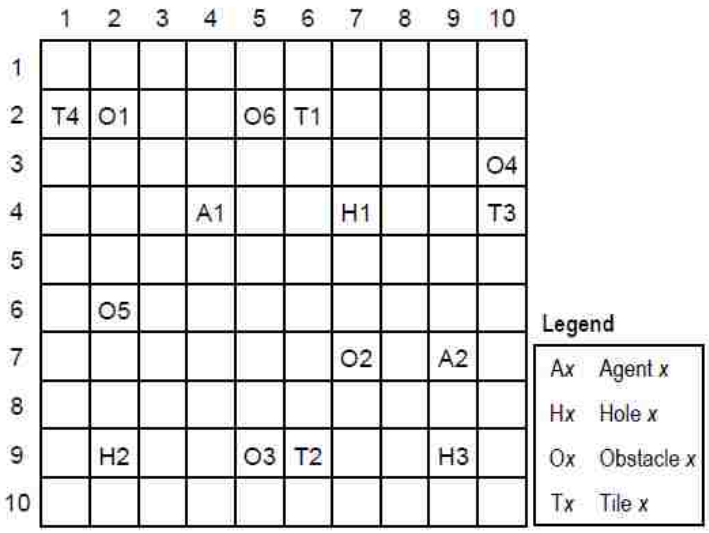


Figure 2.2: Annotated diagram of a Tileworld environment, from Choy *et al.* [2004].

There are numerous potential scenarios where proactive failure avoidance behaviour could be advantageous in a Tileworld. For example, if holes appear and disappear randomly, proactive behaviour would allow agents to modify intended plans if their currently intended route risks falling into a newly emerged hole. If agents incurred damage through fatigue – risking further damage through fatigue driven failure – proactivity can be used to avoid activity by a damaged agent, instead driving that agent to (if possible) self-repair or delegate the task to another.

2.3.2 Truckworld

Hanks *et al.* [1993a] define *Truckworld* as a simulator testbed for reactive planning, modelling a world composed of locations connected by roads. Agents represent *Trucks*, performing transport tasks (similar to the *Truck* domain in Section 2.2.2) and modelled at a level of detail that includes their constituent components (i.e. fuel tanks, tyres, loading arms and cargo bays). Locations are populated by objects of various types and properties, including tyre chains that can be fitted to aid driving down wet roads, or bombs which may explode and damage agents in the vicinity. Exogenous events include weather changes, such as rainstorms altering road conditions; the probability of such events may be modified based on various factors, such as time of day. Constraints may be placed upon agent sensory ability – such as limiting perception of sound by distance (Hanks *et al.* [1993b]).

The Truckworld presents an arguably more realistic environment than Tileworld, albeit with a homogenous agent set limited to *Truck* types. There are obvious cases where proactive failure mitigation may be of use. For example, rain may render a road muddy – threatening failure of future travel, and risking agents becoming consequently stuck or damaged. Proactive behaviour can allow earlier adaptation of plans, potentially avoiding backtracking if the agent only reacted upon reaching that road, or allowing agents to (plan to) fit chains in advance and reserve suitable resources earlier (protecting against contention).

2.3.3 Pacifica / PRECiS

Pacifica / PRECiS (Planning, Reactive Execution and Constraint Satisfaction) was created as an openly available testbed, offering scenarios within a fictional island geog-

raphy (Figure 2.3). Pacifica provides uncertainty and dynamism within a realistic domain, with the geographic scale supporting heterogeneous multiagent activity, where exogenous threats range from insurgent attacks to natural disasters. Pacifica has been employed in experiments including distributed collaborative planning and scheduling in NEOs (Non-combatant Evacuation Operations) (Reece *et al.* [1993]) and multiagent emergency response (Komenda *et al.* [2009a]). These involved logistical scheduling and disaster response tasks, such as evacuating civilians or transporting medical supplies. Failure in plan activities when performing such tasks, if not recovered from, may entail severe consequences (such as stranding of refugees or failure to resupply field hospitals) – potentially including threatening human lives.

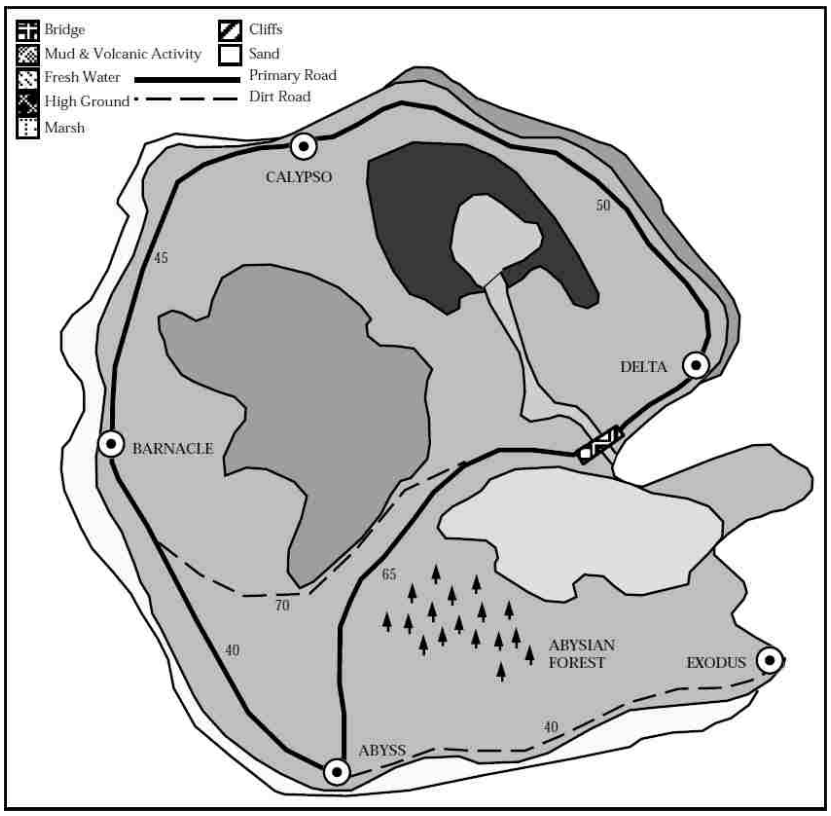


Figure 2.3: Sample Pacifica / PRECiS environment geography, from Reece *et al.* [1993]

These scenarios include cases where the situation may be unfolding, with changing or unknown world state. Plans may be initially formed or selected with incomplete, incorrect or subsequently contradicted state information[2], as it may not be feasible for

[2]It does not strictly matter whether a state has changed or simply been discovered as different; our plan robustness focus is concerned with recognizing whether the *assumptions* under which the plan was formed were contradicted, and identifying if intended activities are at risk of failure.

agents to delay activity until they have absolute certainty regarding the world state. For example, a road believed passable and safe may subsequently become known as blocked or dangerous. In the latter case, agents may still use that – still traversable – road, but risk damage to the agent or their cargo or passengers. Reactive recovery would incur any (potentially lasting) damage from failure, but proactive strategies would avoid such failure – potentially offering greater flexibility in doing so, if threats can be identified far enough in advance.

2.3.4 Blogohar

The Blogohar scenario (Figure 2.4) is designed around two human 'players'; one representing a military force combating a violent insurgency, the other a humanitarian organization seeking to evacuate civilians (Sensoy *et al.* [2010]). Both must form collaborative plans to best achieve their individual goals, and are constrained by policies derived from real world guidelines (for example, restrictions upon communication between military and humanitarian organizations).

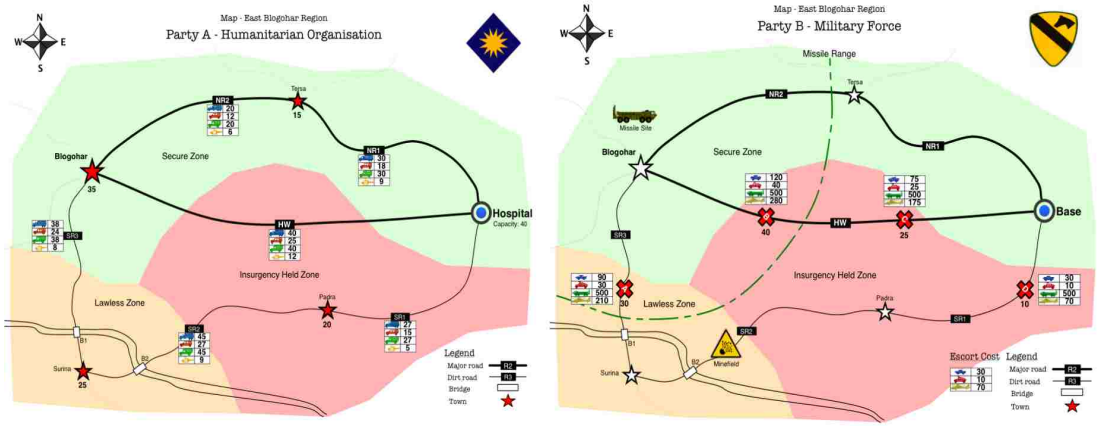


Figure 2.4: Map of the *Blogohar* domain for both 'players', from Sensoy *et al.* [2010]

.

As with Pacifica, non-agent antagonists introduce debilitative and unpredictable exogenous change — providing motivation for pre-emptive activity, particularly as *hostile* antagonists introduce the possibility of debilitation. These types of scenario also suggest non-deterministic representation of risk should be considered, for states not significant enough to represent in precondition terms yet still influencing the outcome of activities. For example, humanitarian agents could revise plans upon awareness of nearby insurgent activity, to combat a consequently increased risk of failure – such as changing route to use safer areas, or requesting military escorts.

2.3.5 Robocup Rescue

The Robocup Rescue simulator (Figure 2.5) defines a scenario based upon the Kobe earthquake of 1995 (deriving from the domain description in Kitano *et al.* [1999]). This domain has been employed for investigation into multiagent planning and collaboration, including use of Partial Global Planning (Pereira *et al.* [2011]) and by Siebra and Tate [2006] to extend the I-X mixed-initiative planning approach (Tate [2001]) through the I-Rescue application. Three types of physical agent are modelled (with equivalent logical commander agents); Each physical type performs specific types of task – *Fire Brigades* extinguish fires, *Ambulances* evacuate casualties and *Police* clear blocked roads.

Figure 2.5: Screenshot of the Robocup Rescue environment during a simulated earthquake disaster; from `http://www.robocuprescue.org/simleagues.html`

The simulator models fire spread, building collapse and both agent and non-agent (civilians requiring rescue) entity health – providing conditions for generating goal tasks, and for representing threats to agent capability. Fire and building collapse may damage agents (although graded loss of capability is *not* modelled); the latter can also block roads. Whilst a reactive approach would respond to exogenous events only *after* they cause activity failure (potentially damaging the acting agent), a proactive approach would seek to avoid such failure and damage. This is arguably particularly important given the relative homogeneity of agent types; if an agent is damaged or destroyed from failing an activity, the typical response would be to find another instance of the same

type to perform an equivalent activity. This makes it important to preserve individual agents, as goals cannot be achieved through structured use of other agent types with semantically differing abilities[3].

Activity failure risk would (be anticipated as) increase with worsening building conditions or fire spread; agents could compensate by bringing in supporting agents to tackle severe fires, or re-routing away from buildings at risk of collapse. For example, a *Fire Brigade* agent could determine an intended route is blocked, and pre-emptively dispatch a *Police* agent to clear it; pre-emption can ensure support requirements are identified and agents reserved earlier than if only responding to failure upon the *Fire Brigade* agent reaching (and failing to use) the blocked road.

2.4 The Cargoworld

We require definition of a suitable environment to provide examples of our desired behaviour, and as a basis for experimental evaluation. Whilst we have highlighted existing experimental domains, there are several issues restricting their viability for our experimentation. Firstly, as a somewhat abstracted environment, we judged extension of the Tileworld as risking accidental bias (Lees [2002]). More real-world orientated domains suffer from lack of available modern simulators (such as with Truckworld and Pacifica), or lacked the configurability required for evaluation – such as to control rates and probabilities of exogenous change, or for debilitative failure consequences.

Consequently, we define the *Cargoworld* – embodying a geographical model similar to numerous domains (including those surveyed previously), with potential for exogenous change, and potential for debilitative failure consequences. Cargoworld is a *Transport*-style domain deriving from the principle expressed by Tate *et al.* [1998] that agents "*go places, do things*"; we argue this general concept can be abstracted to cover a multitude of other domains[4]. System goals are concerned with movement of cargo from an initial junction to a requesting destination, and require agent co-operation. The domain features a variety of heterogeneous threats, agent types and avoidance or recovery responses – to avoid failure mitigation simply consisting of repeating the same

[3]The principal planning problem arguably becomes not which activities are required to achieve a goal, but rather which instances of an agent type are most efficiently located.

[4]i.e. that in most domains, system activity is formed of processes of preparing for or enabling some goal-required activity ("go places"), and of actually performing that activity ("do things").

activities using different instances of the same agent types.

Entities in Cargoworld are situated within a bi-directional graph structure, representing a road network; each node represents a *location* or *junction* (Figure 2.6). The road connecting some junction A to B is given as $A \rightarrow B$ [5]. The system goal – or *top level goal*, referencing its location in a decompositional team-goal hierarchy – is to transport *cargo* to a specific destination (*request junction*), where it can be consumed.

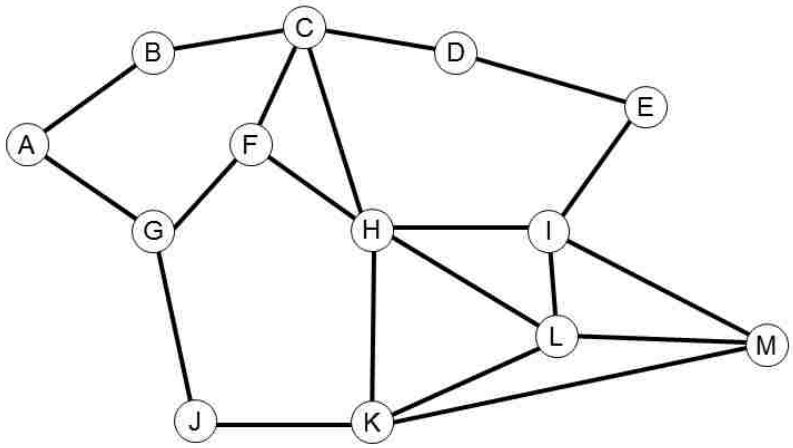


Figure 2.6: An example geography for the *Cargoworld*

Roads in Cargoworld have several properties, potentially contributing to risk for activities involving their use;

- Roads are either *tarmac* or *mud*, indicating surface composition.
- The road condition may be *dry*, *slippery* or *flooded*; this, in combination with road surface and type of travelling vehicle, influences the risk of failure when travelling along that road.
- Roads may be *blocked* by landslides or *toxic* due to chemical contamination following cargo spillage, preventing travel.

It is possible for a road to be (for example) flooded, blocked and toxic - these states are not mutually exclusive. However, a road can only have one surface composition, and cannot be (for example) dry *and* slippery simultaneously.

[5] $A \rightarrow B$, $B \leftarrow A$ or $B \rightarrow A$ all refer to the same bidirectional connection between A and B. However, we will use the arrow to indicate direction of travel when referring to a road within the context of agent movement.

2.4.1 Perturbation

Environmental perturbation is a key factor in considering robustness; we are concerned with unpredictable, stochastic domains where exogenous change can impact the likelihood of activity success. This leads to a degree of non-determinism; the world state after a successful activity is determined both by that activities effects and any exogenous changes during execution.

Exogenous change can be characterised as any state change which occurs for a reason outside of an agent's planned activity. We define several types of unpredictable exogenous change;

- Rain can fall on roads, causing transition from *dry* to *slippery* and, finally, *flooded* states.
- Roads can dry out – transitioning from flooded to slippery and finally dry states.
- *windy* conditions can arise, increasing risk for flight activities.
- Landslips (or equivalent events) may render roads **blocked** (impassable).
- The Cargoworld is subject to insurgent activity, which can render junctions dangerous; creating **dangerZones** at those locations. Agents (except APCs) cannot successfully act in, or move through, any junction with a dangerZone present.

2.4.2 Entity types

Several types of entity are situated in, and can act within, the Cargoworld; these are controlled by agents, which represent these physical entities within the MAS. In the remainder of this thesis, we refer to agents and entities interchangeably due to this proxy nature. However, it is important to distinguish that damage to a physical entity does not impair the associated agent; instead, that agent is responsible for representing and, where possible, handling the consequences of such damage within the context of planning activity and team relationships. Each entity has a given health state, which influences activity execution – there is a greater chance of activity failure if the acting agent is damaged, and a mortally damaged agent cannot perform *any* activity. Damage is a potential consequence of activity failure; health state gradually recovers over a period of inactivity.

Vehicles are entities capable of performing a *move* activity – travelling from a start

to a destination junction. With the exception of helicopters, movement is constrained to using roads; blocked, flooded, or – with the exception of an APC – slippery mud or toxic (contaminated) roads cannot be used for movement. Failure of a move activity may result in debilitative consequences that leave the acting entity 'stuck' off-road. Finally, failure of movement when loaded with cargo, or cargo loading or unloading, risks destruction of that cargo and spillage (if the cargo is of *hazardous* type – e.g. nuclear or chemical waste) rendering roads *toxic*.

- **Truck**s are road vehicles which can load or unload cargo, and move along roads; these represent the basic activities required to transport cargo between junctions.
- **Helicopters** are able to load and unload cargo, and travel directly between any two junctions, but can only land or take off from those containing an *airport*. Flight activities (*takeOff*, *land*, or *fly*) are threatened by *windy* conditions.
- Armoured Personnel Carriers (**APC**s) are able to secure dangerous areas – removing a *dangerZone*. They are uniquely capable of moving along slippery mud (with the same risk as for slippery tarmac) or contaminated roads.
- **Bulldozer**s can unblock roads (remove *blocked* states).
- **Hazmat**s (**Haz**ardous **Mat**erial handlers) can decontaminate roads (remove *toxic* states).

This offers a fairly heterogeneous agent set compared to a domain such as *Truckworld*. Entities either achieve the top-level goal of cargo delivery, or facilitate achievement by others. This gives the MAS more options and flexibility (in both proactive and reactive failure mitigation) beyond simply selecting an equivalent type agent to perform a task (i.e. 'fail-then-retry'). Some agent abilities – particularly *movement* – achieve the same goals with different semantics – impacting their preconditions, (side) effects, and the impact of various world states upon likelihood of success.

2.5 Summary

This chapter described extensions of example domains to show both how exogenous change can impact plan execution *and* indicate how failure may have lasting negative consequences which hinder reactive recovery. We also justified the choice of *Cargoworld* as our evaluation domain; an example MAS in this domain is detailed in Chapter 7, serving as a specification for experimental evaluation.

BDI systems typically employ a reactive approach towards failure mitigation (in the context of *plan* failure) – what Toyama and Hager [1997] define as *post-failure* robustness, rather than *ante*-failure strategies. In our knowledge, the relative benefits of proactivity and reactivity within realistic domains have not been directly compared. Consequently, the existing domains discussed here are limited in their definition and modelling of failure consequences, due to an assumption of, or desire to study, reactive strategies.

We do not argue reactive approaches are disadvantageous; indeed, the obvious uncertainty constraints upon any proactive approach will likely require complementary reactive handling for where a failure is not anticipated (false negatives) or preventable. Rather, we suggest the assumption of de-facto reactivity will bias domain definitions against considering the *possibility* of irrecoverable failure, as such scenarios may not be useful for evaluating post-failure recovery. However, we argue debilitative failure can, and will, exist in *real-world* scenarios – and represents a valid motivation for our approach.

Chapter 3

Agent Systems

We contribute an approach for robustness in the context of agent plan execution. This chapter defines the concept of an agent, detailing the BDI reasoning approach and it's extension to a multiagent context – providing a background for our contribution and discussion of agent robustness approaches in Chapter 4.

3.1 Agents and Multiagent Systems

Intelligent agents, as defined by Wooldridge [1999], are capable of *'flexible autonomous action'* through possessing three key characteristics;

- **Reactivity**: The ability to respond and adjust to changes in the environment.
- **Pro-activity**: The ability to autonomously adopt goals and perform consequent goal-directed behaviour - i.e. to 'take initiative' in line with the agent's design objectives.
- **Social ability**: The ability to interact with other agents to achieve goals, including structured interactions like contract formation and negotiation.

Rationality is a key component of intelligence. van der Hoek and Wooldridge [2003] define a rational agent as one which acts *"in it's own best interests"*; the agent, given a set of possible outcomes, will direct its behaviour to favour the most desirable outcome (where the calculation of desirability reflects the agent's designed purpose). Rational agents typically receive continuous input from their situated environment, responding with the selection of goals and the corresponding performance of actions to affect that environment; they consequently will hold beliefs about the world, goals, plans, and committed partial plans or intentions for response to external events or internal goals (Kinny *et al.* [1992]).

3.1.1 Multiagent Systems Approach

A Multiagent System (MAS) is composed of multiple interacting components (Wooldridge [2002], where that system's purpose is achieved through achievement of individual goals by constituent agents (McArthur *et al.* [2007]). MASs have been employed for domains including aerospace (Šišlák *et al.* [2010]), military (Sokolowski [2003]), space exploration (Micalizio and Torasso [2008a]), power management (McArthur *et al.* [2007], Santofimia *et al.* [2010]), coalition systems (Allsopp *et al.* [2002]) and emergency response (Zhan and Chen [2008]). The latter includes simulation of fire propagation (Han *et al.* [2010]), evacuation (Narzisi *et al.* [2007], Filippoupolitis *et al.* [2009]), and disaster response (Schurr and Tambe [2008], Marecki *et al.* [2009], Wu *et al.* [2008]).

Sycara [1998] states several motivations for adopting a MAS approach, including whether the domain naturally lends itself to a distributed solution, and where a MAS offers extensibility, flexibility or robustness benefits. Jennings [2000] argues MASs offer significant advantages over 'traditional' methods for complex, distributed systems – particularly in flexibility, as the autonomous nature of intelligent agents makes them ideal for dynamic environments (Zwitserloot and Pantic [2005]). Hahn *et al.* [2003] argue the componentized and modular nature of a MAS improves robustness through providing abstraction and offering potential dynamic service composition or redundancy – although Kumar and Cohen [2000a] note a requirement for fault tolerance and recovery techniques *specific* to the Multiagent paradigm.

3.2 The Belief-Desire-Intention Approach

Our contribution focuses upon the Belief-Desire-Intention (BDI) approach to rational agency. Derived from theories of human mental reasoning (Bratman [1999]), BDI has become a de-facto standard for implementing intelligent agents (Wickler *et al.* [2007]). Rationality is driven by processes of goal selection, plan identification and plan execution – the latter being the focus of our contributed robustness behaviour.

3.2.1 BDI Mental States

BDI agent reasoning can be viewed as two parts; *deliberation* – choosing a goal to pursue – and *means-end reasoning* – identifying a plan to achieve that goal – and is

driven by three mental state components;

- ***B**eliefs*; *believed* knowledge – i.e. about the environment and the agent itself.
- ***D**esires*; a set of potentially inconsistent goals, each considered desirable given current Beliefs.
- ***I**ntentions*; a *consistent* set of Desires the agent has committed to pursue – i.e. what the agent intends to *do*.

The definition of an intention can be seen to vary within the literature, depending upon the research focus and potentially the temporal context (i.e. within the reasoning cycle) under which intentions are considered. These differences can perhaps be attributed to the concept expressed by Wooldridge [2002] that, for an agent to behave rationally, adopting a goal will inherently lead to *acting* towards that goal – that forming an intention entails commitment to both a goal *and* to execute some planned set of activities. The specific definition of intention used within a particular work may therefore be influenced by the specific aspects of agent reasoning under consideration (i.e. can be context specific), and may also be further restricted by any assumptions regarding *implementation* of practical systems.

Georgeff and Ingrand [1989], for example, define an intention within the *Procedural Reasoning System* (PRS) – an early BDI framework – as a selected task to be executed, with the *I* set as a corresponding hierarchy. The intentions held by an agent can range from a committed goal (i.e. abstract task) to a specific executable activity, with the former being refined into the latter through successive reasoning cycles. In contrast, the BOID architecture (discussed in Section 3.3) is concerned with motivational sources for goal selection within agents employing BDI reasoning (including influences from *Obligations* to others – hence the 'O'), and consequently views Intentions as committed goals. Thangarajah *et al.* [2002] define intentions as selected *plans*, and argue there exists a need for standard formal representations for mental state concepts of Desires and selected Goals.

This thesis adopts the model of Simari and Parsons [2006], where an Intention defines both a committed goal *and* associated plan. As we are concerned with preventative modification, we require our reasoning to be able to consider both the current plan and – to facilitate reconsideration upon threat to contained activity(s) – the goal that plan is attempting to achieve. This also serves to state our assumption that *if an*

agent intends to perform some planned activity, it has some goal achievement reason for doing so, and that if an abstract task (goal) is refined to specific activities, the former can still be determined for (associated with) the latter. Finally, a goal:plan definition can be seen as a simplification of a further assumption that if intentions *do* represent decomposing task hierarchies both the original root goal and the current sequence of planned activities (i.e. a plan) can be inferred (provided knowledge is retained of the decomposition of intended tasks into subtasks).

3.2.2 Maintenance Goals

Braubach *et al.* [2006] define two types of goal driving agent proactivity; to *achieve* some state, and to *maintain* it over a set time or under defined conditions. Duff *et al.* [2006] distinguish reactive and proactive types of maintenance goals; the former requires re-establishment of the goal state if violated, the latter constrain goal and plan adoption to prevent violation. Reactive maintenance goals can be considered part of the 'background' reasoning under which agents select desires and identify plans, whilst proactive maintenance goals may motivate adoption of *achievement* goals to re-establish violated states.

3.2.3 The BDI Agent Reasoning Cycle

An example of a generic BDI reasoning cycle (Rao and Georgeff [1995]) is given in Algorithm 1. The reasoning cycle begins with the initialization mental state components (i.e. B, D and I), followed by continuous iteration during the agent lifecycle. The start of each reasoning cycle sees the agent update current Beliefs, including with percepts of external events perceived at the end of the preceding cycle (represented in the *eventQueue*). The *optionGenerator* uses these updates to identify *potential* Desires (*options*), of which a consistent subset will be used to form *Intentions* for execution. Here, B, D and I mental state components are globally accessible and implicitly updated with execution of the constituent functions of the reasoning cycle.

Each Desire represents a potential intention, with a non-conflicting subset selected by the *deliberate* function; these are then used to update the agent's I set. The *execute* step can be argued as necessarily vague to allow different implementation specifics. In general, *execute* can be summarized as both performing intention refinement processes and performing executable activities. For a model such as that used by PRS (described

in Section 3.2.1), where the I set is an effective hierarchy of committed tasks, this can be seen to result in addition of *new* intentions corresponding to the refinement of intended abstract tasks, and the execution of *atomic* intentions where a task corresponds to a primitive activity. In the case of our adopted representation of an intention as combining a goal and plan, the initial plan can represent an abstract goal-achievement task – with the plan being refined *and* having any atomic activities executed (i.e. interleaving planing and execution) during the reasoning cycle.

Algorithm 1: Generic reasoning cycle for a BDI agent (Rao and Georgeff [1995])

```
initializeState();
while agent is alive do
    // Generate potential desires
    options ← optionGenerator(eventQueue);
    // Select desire(s) to pursue
    selectedOptions ← deliberate(options);
    // Form intentions from selected desires
    selectedIntention ← updateIntentions(selectedOptions);
    // Form plans for intentions and execute atomic intentions
    execute();
    // Update event queue
    getNewExternalEvents();
    // Identify succeeded intentions
    dropSuccessfulAttitudes();
    // Identify impossible intentions
    dropImpossibleAttitudes();
    // Determine intentions/goal elements carried to the next
       cycle
    postIntentionStatus();
```

Intentions are progressed over multiple reasoning cycles; agents may interleave intentions depending upon their deliberation strategy. At the end of each reasoning cycle, sensing is performed to detect changes in the environment and identify the outcome of activity execution; the current intention set is updated to progress partially executed plans and remove those completed or now considered impossible.

3.2.4 Runtime Planning In BDI Agents

Due to reactive time constraints, BDI agent implementations – such as the Procedural Reasoning System (PRS) (Ingrand *et al.* [1992]) or *Jason* framework (Bordini and Hübner [2006]) – typically employ libraries of plan recipes, mapped to triggering events and selection conditions. Use of plan libraries has led to criticism that BDI agents cannot learn and adapt, and are restricted to scenarios envisaged during offline plan formation. Singh *et al.* [2010] have suggested an approach for re-use of library plans through learning new selection conditions for selection based upon historical success rates under various execution contexts – although this did not account for where multiple plans are attempted for the same goal, or automated repair or recovery.

Approaches have investigated integration of runtime planning in agent behaviour, albeit with constraints upon invocation. *CANPLAN* (Sardina *et al.* [2006] – and the later *CANPLAN2* (Sardina and Padgham [2007] – extended *Conceptual Agent Notation* (Winikoff *et al.* [2002]) to define declarative goal constructs, linking event triggered goals to defined goal states, a program, and failure/unachievability conditions (for decommitment) – where that program could be a plan recipe or invocation of a runtime HTN planner. Silva and Padgham [2005] also defined a framework where plan recipes can explicitly invoke runtime planning, using the agent plan library to form an HTN domain representation by mapping plans and goals to refinements and task definitions. The *Peleus* system (Meneguzzi and Luck [2008]) similarly allowed the invocation of a (classical) planner as an explicit activity within plan recipes. All of these approaches give the programmer, rather than the agent, control over when runtime planning is used on the basis of controlling computational cost; for example, to avoid the excess of an agent attempting planning for an intractable goal. This does require the designer to anticipate scenarios where runtime planning is required – potentially risking the same disadvantages argued for plan recipes.

Runtime planning offers optimal flexibility for BDI agents, but with an associated computational cost. The surveyed approaches support it's viability, at least as a constrained special case behaviour – ongoing advances in automated planner optimization should also improve the practicality of runtime planning. However, exogenous change may still threaten activities during execution *regardless* of whether the intended plan was formed at runtime or implementation time – meaning our motivation holds regard-

less of the method used to form intended plans.

For our robustness approach, use of runtime planning will offer greatest flexibility – like invocation of runtime planning in these approaches, our robustness behaviour must address scenarios not anticipated or predicted during the original plan formation. The ability to map BDI plan libraries (if existing) to HTN domain concepts, given by Silva and Padgham [2005], offers a means to support reasoning over whether agents posses the plans required to meet as-yet unrefined goals (or subgoals within plans). However, we cannot *require* runtime planning as an inherent requirement of our approach, as this could restrict it's applicability.

3.3 Mental States for Multiagent activity

The BDI approach models behaviour required for the first two properties of intelligence defined by Wooldridge [1999]; reactivity (reconsideration of intentions and desires in response to belief changes) and pro-activity (adoption of desires as intentions). However, with regard to *social ability*, BDI does not explicitly model *multiagent* behaviour. As co-ordinated activity is a key motivator for a MAS approach, this section discusses the mental state components employed (and, potentially, added) for various multiagent activity models, which may similarly be employed by our own robustness approach. These approaches are not necessarily *specific* to BDI agents, although they may use similar terminology and concepts to BDI mental states.

Joint Intentions theory (Levesque *et al.* [1990]) models agent behaviour performing co-operatively performing joint activity. *Joint Intentions* (*JIs*) are shared commitments to perform an action, modelled using the same primitives as individual commitment – (mutual and local) beliefs, goals, agents involved and plans. Agents pursue a *JI* so long they mutually believe the associated joint goal still holds and is achievable; if this belief no longer holds, agents adopt goals to inform other team members, re-establishing mutual belief as part of performing group de-commitment. Mutual beliefs and joint goals also constrain agent reasoning, to avoid local agent behaviour that would threaten joint activity success. *JI* does not address recovery from loss of mutual belief; agents *could* respond to decommitment by forming a new joint intention towards (retrying) the decommitted goal, but this is not explicitly required.

Joint Responsibilities (*JR*) theory, by Jennings [1992], extends *JI* by modelling separate commitments to goals and plans – allowing the latter to be decommitted whilst retaining commitment to the former. A *responsibility* is a commitment to the shared plan which persists until either the goal is achieved, the plan completes without achieving the goal, or an activity fails. If a plan is believed no longer suitable by an agent (i.e. due to failure or exogenous change), JR allows agents to suggest remedial actions as part of mutual belief maintenance – allowing group commitment to a new plan that restores mutual belief in achievement of (and avoiding decommitment from) the joint goal. This permits (and indeed inspires) the proactive robustness behaviour we desire – where agents modify plans, *preserving* commitment to an intended goal, rather than aborting (or inevitably failing) if exogenous change renders that plan non-viable. *JR* theory informs our treatment of intentions as combining a goal and associated plan – allowing the latter to be mutable, and where the former denotes that plan's *purpose*.

The communication requirements for both *JI* and *JR* may involve introspection to determine whether plans are viable – whilst decommitment conditions can be specified, meta-knowledge is likely to be required in order to define conditions where a committed plan is (believed) unable to achieve the goal. We can form a requirement that in our approach agents must possess *capability meta-knowledge* to determine whether activities within individual or joint plans are threatened. We assume this information is also shared between agents as part of distributed plan formation and execution.

Planned Team Activities (*PTA*) by Kinny *et al.* [1992] again uses a joint activity model based upon extended local mental state concepts – i.e. joint beliefs, joint plans, joint goals, and joint intentions (plans committed for execution). Advance reasoning over achievable goals and agent-activity assignments is supported by modelling known *skills* (executable primitive activities) and pre-formed plan libraries. *PTA* plans are acyclic graphs of activities, each corresponding to a required skill – successful execution requires finding a path through the graph and generating a set of *role-plans* that ensure all activities can (and will) be executed by appropriately skilled agents. Reactive failure recovery is supported through back-tracking to find alternate paths or alternate role-plans. *PTA*'s use of *skills* emphasises the utility of holding and sharing capability meta-knowledge when supporting distributed activity.

A similar decoupling of goal from plan as in *JR* is found in the *SharedPlans* formal

model of collaborative planning (Grosz *et al.* [1999]). *SharedPlans* characterizes two types of intention; to achieve some proposition (*Int.Th*) – i.e. goal – or to perform some activity (*Int.To*). The latter type can arise through means-end reasoning for the former. *SharedPlans* extends the mental state model of plans (Pollack [1990]) to the distributed context – holding a plan requires both knowledge of how to perform the requisite actions and an intention (i.e. *Int.To*) to do so. The *SharedPlan* – multiagent plan – is formed through individual means-end reasoning by agents and may contain both subsidiary *SharedPlans* (i.e. multi-level decomposition) or individual agent plans. Reasoning about plans requires agents hold knowledge regarding primitive activities (i.e. equivalent to *skills* in *PTA*) and plans for decomposed goals. Formation of a full (complete) *SharedPlan* from an incomplete *SharedPlan* requires group belief that a full plan *can* be formed; this knowledge requirement can be extended to cover awareness of both an agent's own capabilities and those of other team members.

The previous approaches and models are concerned with joint *activity*, but are not explicitly defined for BDI agents. *BOID* (Broersen *et al.* [2001a]) and *B-DOING* (Dignum *et al.* [2002]) extend BDI mental states to model goal selection under various motivation sources and constraints. *BOID* views Intentions as selected goals, with Desire and Obligation sets respectively internally and externally motivated candidate goals. BOID agents arbitrate between (*external* conflicts) and within (*internal* conflicts) their four mental states; e.g. conflict between *B* and *I* indicates the latter cannot be achieved following environmental change (Broersen *et al.* [2001b]). Agents are classified by the precedence ordering used to arbitrate internal conflicts – e.g. *selfish* agents prioritize Desires over Obligations, while *social* agents apply the converse (Broersen *et al.* [2002]).

The B-DOING architecture models motivational sources when forming and maintaining intentions; here, Intentions are committed *plans* to meet the selected goals represented in a Goals set. Figure 3.1 depicts the Goal and Intention Maintenance stages. Goal Maintenance arbitrates between the motivational components of *Obligations*, *Desires* and *Norms* to form a consistent set of Goals; Intention Maintenance uses means-end reasoning to form plans according to selected Goals and current Beliefs. Intentions may be modified or cancelled to maintain the consistency of committed Goals – or agents may modify their Goals to avoid dropping an intention they are strongly committed to. B-DOING models several motivational components in addition to De-

sires. *Norms*, applied to either an entire system or group of agents, represent societal desires and constraints upon behaviours and are an *inherent* requirement of operating within that agent society. *Obligations*, conversely, are formed with other agents through teamwork (such as during contract formation) and entered into by choice.

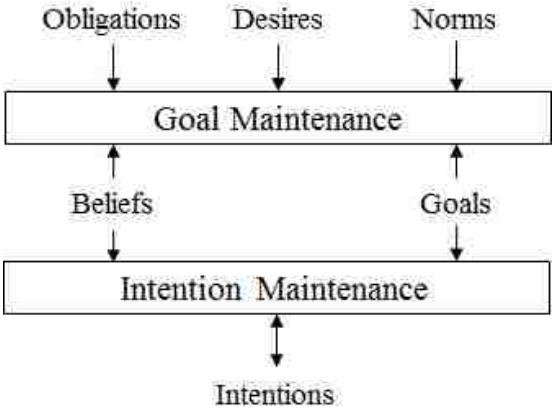


Figure 3.1: Goal and Intention Maintenance stages in B-DOING (Dignum *et al.* [2002])

BOID and B-DOING both model internal and external motivations – i.e. whether or not an intended goal and associated plan are driven by the agent itself or request by another. We assume that, at minimum, *Obligation* information will be available to our robustness approach. We also assume agents are aware of their dependencies upon others (neither B-DOING or BOID model mental components to represent dependencies, likely because these are not relevant as motivators for goal adoption), and that they form *contracts* to establish dependant-obligant relationships. Our eventual design assumes contracts are necessary to guard against contention over agent resource in the types of domain we target, and also that they can facilitate communication of mutual belief information (by defining delegation relationships).

3.4 Conclusion

This chapter discussed agents and multiagent systems (MASs), focusing upon the Belief-Desire-Intention (BDI) approach and extension of BDI mental states for distributed activity. We formed a number of requirements and assumptions:

- We target BDI-based agents due to BDI's status as a de-facto standard; our approach should employ BDI mental state concepts and be defined with reference to the reasoning cycle given by Rao and Georgeff [1995].

- We address plan execution robustness due to the importance of plans in BDI rationality.
- Previous work has shown the viability of runtime planning for BDI agents; as we are concerned with robustness in unforeseen circumstances, it is desirable to utilize runtime planning for improved flexibility.
- As agents may face computational constraints, we require our approach to not *rely* upon specific runtime planning methods.
- Contract formation is assumed necessary to protect against agent resource contention when delegating activity.
- To understand roles and communication responsibilities, agents are assumed to model their obligations and dependencies for delegated activities.
- We require agents hold beliefs (meta-knowledge) regarding their capabilities, to allow introspective reasoning and detection of threats to intended plans; this knowledge is assumed to have a potential additional utility for mutual belief maintenance – e.g. in *Joint Intention* (Levesque *et al.* [1990]) and *Joint Responsibilities* (Jennings [1992]) theory.
- Our approach should provide behaviour defined in JR theory – i.e. detect and counteract threats to planned activities, including communication to restore mutual belief in achievement of the relevant joint (intended) goal.
- Agent capability knowledge must cover primitive and composite activities – both the former (e.g. skills in *PTA*) and latter (e.g. plan knowledge in *SharedPlans*) are relevant in defining how well an agent can achieve goals. This covers both reasoning about specific, selected and intended plans, and which goals an agent *can* achieve (and under what circumstances).
- Distributed plan formation requires agents reason over their ability to delegate activities; our capability model must be communicable between agents, to support robustness reasoning where plans involve dependencies.

Chapter 4

Agent Robustness Strategies

MASs have been employed in domains where agents face unpredictable, partially observable and potentially dangerous environments. As domain difficulty increases, the risk of failure and value of robustness methods also increase. This section describes a number of such methods, which address different aspects of MAS robustness.

4.1 Defining Robustness

Burns and Wellings [1990] identify four causes of fault in a real time system – inadequate specification, design errors, processor failure, and communication error. Within the MAS context, Hägg [1997] suggests the latter two represent run-time considerations for designing fault-tolerance approaches. We similarly suggest three classes of failure which may affect agents; total agent failure ('death'), failure whilst remaining able to inform others, and activity failure. For the latter, an activity may complete execution successfully with different post-effects to those expected; whether this constitutes *failure* depends on whether the plan goal was to *have performed* that activity, or to have achieved a *specific state*.

Covrigaru and Lindsay [1991] state *'robustness is required for self-sufficiency'*, arguing agents must be able to adapt and respond to situations beyond those easily anticipated by a designer. Schillo *et al.* [2001] define robustness as the ability of a system to meet *'safety responsibilities'* – defined by Wooldridge *et al.* [1999] as being to prevent some undesirable condition – despite debilitation or failure. Hahn *et al.* [2003] further define robustness as *'graceful degradation of performance under perturbation'* – recognizing that if full recovery is impossible, sub-optimal performance is

preferable to total loss of ability. Degradation may be expressed through reduced goal achievement, or relaxation of constraints (e.g. extending deadlines or removing resource restrictions) to still achieve *suboptimal* versions of original goals. Hägg [1997] suggests three levels of fault tolerance; *full* fault tolerance (performance and functionality are never significantly degraded), graceful *degradation* (operations continue with *some* loss of functionality or performance) and *fail-safe* (specific vital functions are preserved).

Our contribution focuses on plan execution robustness – with our perturbative concern being the rate of exogenous change in the environment, combined with the probability of *debilitative* effects from activity failure. Our approach aims to prevent activity failure on the basis that the latter – post-failure debilitation – can hinder post-hoc recovery. Although we can measure the efficacy of our *specific* approach in terms of activity success rate, this is not suitable for comparison against reactive approaches (which do not aim to prevent *activity* failure, but recover from it). Given the previous, we will define and measure robustness in terms of (intended) goal achievement rate.

4.2 Failure Diagnosis

Agents may need to *diagnose* failures to enable recovery. Joint activity, for example, may fail if team members hold inconsistent beliefs – individuals may view their own behaviour as correct, yet the outcome may be negative from the perspective of the *team*. Determining if and where such inconsistencies lie is therefore necessary for their resolution.

In *Socially Attentive Monitoring* (Kaminka and Tambe [1998]), agents use social diagnosis to diagnose team failures. SAM employs plan recognition to infer the mental state of other agents based upon their (observable) actions, using *model-sharing* (Tambe [1996]) – where team agents share operator models, which indicate their cohorts expected behaviour and infer beliefs – to reduce communication requirements. Failures are recognized by divergences between beliefs, goals or plans compared to other team-members; upon detecting divergence in behaviour, the agent will backtrack through it's own beliefs to determine the exact difference. SAM does not address situations where perception is *erroneous* rather than incomplete – i.e. such as where an agent's sensor reports incorrect data.

Model-Based Diagnosis utilises a system model – formed in terms of components, their interrelations and behaviour – to establish the cause of malfunctions. Roos and Witteveen [2005] propose an extension to agent-based plan diagnosis. Agent health states are modelled as an explicit set that can be related to specific failures through causal rules; the resulting causal diagnosis can be applied to explain observed errors, and for prediction. However, this approach is single-agent only, and does not consider failures from errors by *other* agents. It also does not consider failure response, although modelling of health states may be useful for reactive recovery.

Micalizio and Torasso [2007b] describe a distributed approach for monitoring and failure diagnosis of a multiagent plan (MAP) containing joint activities, by extending a model-based approach. The MAP is modelled in terms of activities, plus their causal links and precedence constraints. Agents are assumed to co-operate in service provision – i.e. agent i will provide some service for agent j – and distinguish different *types* of failure. *Primary* failure denotes failure of the agent's own activity; *secondary* failures represent consequent failures elsewhere in the MAP – i.e. where primary failure by i leads to secondary failure of (some set of) j's activities. This also distinguishes plan and agent diagnosis; the former is concerned with identifying (primary or secondary) activity failure, the latter with explaining the source of failures as some combination of functional faults.

Agents use a Plan Execution Monitoring (PEM) module (Section 6.1.4) to supervise activity execution; detecting failure where not all expected effects were achieved (Micalizio and Torasso [2007a]) – this does not consider additional *unexpected* effects as a source of failure and trigger for diagnosis and repair, however. Local activity failure initiates diagnosis to identify the root cause and relate that failure to the MAP; identification of secondary failures requires co-operation between members to communicate the details of their threatened local activities. Agent level diagnosis aims to infer possible causes for failure, including identification of agent health state. Plan diagnosis seeks to determine *causal* (violated causal links) or *fault* (sub-optimal health state) threats to other activities in the MAP.

Eventually, a set of missing goals is formed, indicating those which cannot be achieved due to the activity failure (of either threat type). In an approach given in Mi-

calizio and Torasso [2009], agents first attempt local recovery – planning to restore local state to a *safe status* (including releasing resource locks) such that other agents are not threatened by the failure, before forming a recovery plan to address the missing goals. If either phase fails, the agent aborts the plan and informs other team members; informing them that the failed agent has released resource locks, and will not (re)reserve those resources in future. In response, the other agents revise their plans to account for the MAP changes arising from that agent's failure.

The approaches surveyed in this section are not strictly defined in BDI terms, but may be applied for detection of belief or intention inconsistencies between agents involved in joint activity. They may detect the cause where failure *has* occurred, or predict it's occurrence through identifying belief divergences which risk incorrect adoption of intentions (and execution of activities). These approaches do not always define a response mechanism for detected issues, although resultant information may be useful for other handling mechanisms. Micalizio and Torasso [2007c] *do* diagnose both failure and propose a recovery strategy. However, they define failure as failure to achieve all expected effects – which does not account for exogenous change threatening subsequent activities, unless it removes a member of that effect set. As a reactive method, their approach does not consider whether failure risks debilitation and will not respond if expected effects were achieved but the agent was damaged during execution (i.e. threatening subsequent activity).

Our robustness approach will assume any failure diagnosis implementations exist as part of the general agent framework, outwith the BDI reasoning cycle, and can be reduced to mechanisms guarding against belief inconsistency. For example, detection of unexpected effects (as from Micalizio and Torasso [2007c]) can trigger belief updates and consequent revision of intentions (by our robustness-specific behaviour, or default BDI reasoning). As we are concerned with plan execution robustness, we regard *agent* diagnosis – i.e. detection of component failure, as described by Roos and Witteveen [2005] – as outside the scope of our approach, although agent health state information may lie within the *Belief* set accessible to our approach.

4.3 Sentinel Monitoring and Exception Handling

Hägg [1997] introduces *Sentinels*; specialized 'watchdog' agents that act to prevent undesirable states occurring or to preserve key functionality. A community of sentinel agents can be employed as a control system layer in a MAS. Sentinels can use a variety of information sources – communications monitoring, regular 'heartbeat' signals, or direct queries – to evaluate agent performance, detect belief inconsistencies between agents, and detect (or even anticipate) failure. As sentinels incur computation and communication costs, it is unrealistic to guard *all* functionality – instead, the system designer must identify critical functionality and specify sentinels accordingly for graceful degradation. The sentinel concept is a paradigm for implementing oversight and response, but what the latter entails may be domain specific and is left for the system designer to define.

A related approach to sentinels is generation and handling of *exceptions*. Klein and Dellarocas [1999] define exceptions as generated upon any departure from desired system behaviour; such as agent failure, communications problems or task execution issues. A standard approach towards exception handling is to build specific behaviours into individual agents – the *survivalist* approach. However, this risks increasingly complex and inflexible agents, with the majority of their behaviour defined for exception handling and recovery.

Klein *et al.* [2003] defines the *citizen* approach which, rather than giving individual agents specific handling behaviours, uses an *exception handling* (EH) service. The EH service holds knowledge of a set of generic exceptions, defined in terms of generic state, comparable to specific situations during runtime, and associated with a set of plan templates for responding to that fault. This serves as domain-independent, generic response information – removing the need for more specific behaviour to be provided for each individual. Agents joining the MAS provide the EH service with a representation of their behaviour, to allow pattern-matching against generic exception types. Depending upon the implementation, introduction of a new agent may lead the EH service to generate a sentinel to monitor for occurrence of failure types identified from the provided behaviour information. This sentinel may also transparently monitor communications of that agent, identifying and correcting corruption (Parsons and Klein [2004]).

Souchon *et al.* [2004] define an approach to apply exception handling and propagation concepts from programming languages (such as Java) to a MAS context. Here, exceptions are either detected internally or – in the case of delegation – communicated as messages to dependants. Agents are viewed in terms of providing services, with *role agents* representing sets of agents which hold the same capability. Role agents broadcast received service requests to their represented agents, and collect responses *or* exceptions to be (respectively) aggregated into a collective response or *concerted exception*. *Exception handlers* are associated with services, agents (i.e. covering all service exceptions) or roles. Handlers either perform corrective action (e.g. restoring state or sending partial results), propagate the exception (if it could not be handled), or retry execution (potentially after acting to modify the execution context). Exception handling searches for an appropriate handler for an exception; if a local handler cannot be found, the agent will propagate the exception to any dependant. The search continues until an exception handler is identified or the top-level agent reached; meaning the efficacy of this approach will rely upon appropriate provision of handlers.

Shah *et al.* [2006] describe an exception diagnosis process for market-based open MASs. Their approach assigns sentinels to agents joining the MAS; the agents are required to inform their sentinel about their goals, plans and mental state[1]. A hierarchical taxonomy of exceptions is modelled – upon detection of a fault, the diagnostic process explores this hierarchy to find the specific exception class. Exceptions are associated with abstract plans, instantiated and executed to confirm the correctness of that diagnosis. If more than one exception is diagnosed, determining the specific exception requires executing and considering the results of each possible exception diagnosis plans. This approach is concerned solely with diagnosis (using the exception hierarchy) and does not define recovery mechanisms.

Snyder *et al.* [2004] describe use of sentinels for failure detection in the *Cougaar* agent architecture, where failed agents are replaced with replicas (Section 4.5). Sentinels sit at the top of a monitoring hierarchy (*robustness community*) partitioned into node or Java Virtual Machine level monitors (which can restart failed sentinels), and (below) individual agent level monitors. In an asynchronous agent system, agent fail-

[1] Although this is justified as a method for agents to preserve autonomy by preventing sentinel introspection into their mental state, we note it still explicitly requires – 'forces' – information sharing.

ure is detected through inactivity – requiring a potentially unbounded wait (Fischer *et al.* [1985]). Cougaar employs *unreliable failure detectors* (Chandra and Toueg [1996]), permitted to diagnose false positive failures, provided that errors will be ultimately detected and corrected. Agent health monitoring is performed through a regular 'heartbeat' signal, passed to sentinels via agent monitors. The wait period for diagnosing failure is set based upon where the cost of *ensuring* correct diagnosis exceeds that of correcting an erroneous one (where the heartbeat arrives *after* failure diagnosis).

Cakirlar *et al.* [2008] define an exception handling approach which classifies exceptions over three levels; plan level (i.e. from activity failure), agent level (including system errors or unhandled plan exceptions), and finally multi-agent level (failures in dependencies due to agent level exceptions). Agents dynamically add goals upon detecting an exception; successful identification and execution of a plan for that inserted goal (through regular agent reasoning) handles that exception. Three types of exception handling goals are given – each must be defined and explicitly associated with a given goal;

- *exceptional* goals, if met, allow resumption of the original plan
- *sameAs* goals denote an *equivalent* goal to that met by the now-failed plan
- *inverseOf* goals 'roll back' post-execution failure state

This does entail a specification burden to define handling goals for each possible agent goal. Goals are attempted in precedence order; if an agent cannot find a plan to achieve an *exceptional* goal, it attempts to find a plan for the *sameAs* and then *inverseOf* goals associated with the failed agent goal. If an agent cannot recover from an exception, that exception is propagated to any dependant.

Sentinel monitoring is primarily a monitoring rather than response mechanism, and may be considered proactive or reactive depending upon the specific implementation. However, these approaches reduce agent autonomy by requiring the sharing of mental states with sentinels (through communication or invasive introspection). We have opted, for sake of generality, to assume any sentinel mechanisms are transparent (e.g. as in Parsons and Klein [2004]) and that their outcome will – in the BDI reasoning cycle – be reflected through receipt of Belief update events.

Exceptions signify divergence from desired behaviour, and can facilitate reactive robustness behaviour – although the work surveyed here has been primarily concerned with representing and communicating notifications of erroneous behaviour. Cakirlar *et al.* [2008] suggest a mechanism for adoption of *responsibility* within decompositional teams through propagation of exceptions of decreasing specificity (such as concerted exceptions in Souchon *et al.* [2004]) up the agent hierarchy. Our approach will require a similar process, with higher level agents in a team responding through local robustness behaviour when a lower level obligant is anticipated as unable to meet their obligation (despite any attempts to resolve issues at their local level). This team-level behaviour can intuitively be expressed through local agent level exception generation/handling, as opposed to using – likely infeasible – centralized approaches for assigning responsibility.

We opt not to utilize exception handlers, as these require a meta-organization (in the form of role agents) which may restrict the generality of our approach, and risk being dependant upon the system designer's anticipation of handler responses for uncertain, stochastic environments. Additionally, an anticipated future activity failure – as our proactive approach should address – is potentially less severe than the definitive, current problems typically entailing an exception. Agents may also be required to arbitrate between multiple anticipated threats, as their 'time window' of consideration would ideally extend beyond the narrow immediacy of *detected* failures. Our approach should allow the anticipation of multiple potential threats to planned activities, and provide agents with autonomy to prioritize their robustness response(s).

4.4 Role Filling Approaches

The behaviour expected of an agent can be defined by the roles it holds (Trzebiatowski and Miinch [2001])[2]. The organization of a MAS can be designed in terms of agent roles (required to achieve the system goal), their inter-relationships, and the conditions (such as capability constraints) for mapping roles to agents (Xu *et al.* [2007]). Role based approaches to robustness are concerned with reconfiguring agent-role assignments when an agent is no longer suitable for, or capable of, it's current role – such as following debilitation.

[2]Roles themselves can be considered analogous to social concepts such as norms

The *Organizational Model for Adaptive Complex Systems* (OMACS) by DeLoach *et al.* [2007] combines a centralized monitor agent with a role-filling approach, where a MAS of heterogeneous agents is defined in terms of goals and roles to be filled. Domain specific functions are used by OMACS to score the quality of an individual agent's capabilities and it's performance within given role. Reorganization of agent-role assignments is triggered by any event impacting (adding, failing or achieving) goals or agents (influencing quality of possessed capabilities), with a hill-climbing algorithm used to find the optimal set of agent-role assignments (based upon the quality scoring functions). A single Organizational Master (OM), equivalent to a centralized sentinel, performs this process. The OM requires total organizational knowledge, which – combined with it's singleton nature – does risk it becoming a central point of failure. OMACS relies upon fixed roles and utility functions – role-assignments can be modified, but not actual roles; for example, there is no possibility of *splitting* the responsibilities of an unmet role into new separate, individually assignable ones. Additionally, OMACS does not handle scenarios where there are insufficient agent resources to fulfil all roles.

Preisler and Renz [2012] propose another role-assignment approach, again based upon agent capabilities. Where an agent is no longer capable of it's assigned role – but other agents *are* – a role-swapping process allows exchanging of role assignments. A decentralized approach is employed, as multiple role swaps between agent pairs may be required to ensure all roles are filled. The general robustness of the system is characterised through the *redundancy rate*; i.e. the number of agents with the capability to perform a particular role. A higher redundancy rate entails more agents can potentially assume a role, and a greater chance of successful reconfiguration (e.g. a 10% rate for a role indicates one tenth of system agents can fill it). However, this approach again somewhat limits flexibility by treating roles as immutable.

Role-assignment approaches act to ensure a pre-specified (role-defined) organization exists; they may be considered proactive or reactive depending upon the triggering mechanism for re-organization (such as whether performed upon agent failure, or upon anticipating suboptimal performance). The actual robustness effect depends upon both the accuracy of role-assignment (correct assignment of roles to appropriate agents), and the organizational structure itself. These approaches will not detect or correct structural weaknesses in the MAS organization – such as central points of failure – or

if insufficient agents exist to fill all defined roles. This contrasts with the flexibility we seek through proactive plan modification – planned activities that require delegation also inherently define execution roles that must be filled by some obligant.

A method is required to assess the utility of an agent for a given role – such as quantitative scoring functions in OMACS (DeLoach *et al.* [2007]) or constraints requiring possession of specific capabilities (Xu *et al.* [2007]). This corresponds to our earlier requirement for agents to possess capability meta-knowledge to introspectively reason over plan activities. Qualitative estimation (e.g. in OMACS) can indicate not just whether agents can perform some assigned task – whether a role assignment or delegated activity – but to what level of *quality*. We require our capability model (i.e. to be used by our contributed approach) to include similar qualitative estimation – if granular estimation is impossible, this can be abstracted to a boolean indicating capability possession.

4.5 Replication

One common, robustness approach for software systems is to provide redundancy, allowing replacement of failed components with equivalents. This introduces costs in providing redundant resource, and in analysing and determining *how* to provide those resources within cost constraints. Redundant backups may be *warm* – brought online and synchronized with the last known state after failure of the original component – or *hot* – kept constantly synchronized.

Within agent systems, *replication* of agents can be used both for redundancy and performance improvement (i.e. parallelization). Deters [2001] describe an approach towards the latter in the form of the *DICE* multiagent framework, whilst observing that replication can also potentially improve fault tolerance. *Replica groups* are formed from a number of identically capable agents, or *replicates*; allowing redundancy and parallel processing or load-balancing. In *transparent* replication, service users are unaware of duplication within the replicate group – i.e. perceiving it as a single agent (resembling *holons*, defined by Schillo and Fischer [2003]). It can be difficult to identify replication requirements for dynamic, large scale systems in advance (Guessoum *et al.* [2005]) – the logical nature of agents can allow *dynamic* replication, where both identifying requirements for, and instantiation of, replicates are performed at runtime.

In their work, Deters [2001] note that memory and computational resource constrain scalability where, respectively, replicated agents are reactive (only act in response to direct messages) or proactive (i.e. in terms of autonomous goal adoption, as with BDI).

Guerraoui and Schiper [1997] describe two replication techniques, with potential for hybrid combinations. In *primary backup* replication, a single *primary* replica receives and handles client invocations. Other replicas are backups; the primary replica forwards requests and responses between client agent and backup replicas. For *active* replication, there is no centralized controller (i.e. no primary replica); client requests are sent to *all* replicas, with the client waiting until either (depending on the specific approach) it receives the first or all responses. In the active replication strategies, the failure of any replica is transparent to a client; in the primary backup case, the client is aware of failure of the primary replica, due to the resultant promotion of a backup replica into the primary role.

Fedoruk and Deters [2002] describe replication as a robustness mechanism for MASs. Replication may be either *heterogeneous* – replicates perform the same tasks, but may vary in functional semantics – or *homogeneous* – replicates share identical codebases, but risk sharing code faults. In both cases, creation of a replicate requires activity to ensure consistent internal state with the replicated agent; this may be more difficult in the heterogeneous case, if semantic differences lead to differences in data requirements or representation. *Proxies* may transparently manage replica groups; *redundant* replicates can be held in a dormant mode (potentially using a hot backup strategy) and reactivated to compensate for replicate failure or to manage increased load. Their experimentation observed that replica groups did incur a communications cost overhead from proxy duplication when forwarding messages to and from replicates, but concluded this cost was not excessive.

Kumar and Cohen [2000b] describe an approach considering teams of *middle*, or *broker*, agents – these perform tasks including routing requests and responses (i.e. akin to proxies), serving as service advertisers, or locating capable agents. In their *Adaptive Agent Architecture* (*AAA*) approach, teams of middle agents hold a Joint Intention (JI) to ensure broker functions are provided to some set of user agents. If connection is lost between some team member and it's user agents, the JI stimulates the other broker team members to (attempt to) connect to the user agents that were being served. Bro-

ker functionality will be consequently restored to user agents following an individual broker failure, provided at least one member of the broker team remains functional. This does risk computational or messaging overload on broker agents, if assuming the responsibilities of a large number of debilitated cohorts. The JI may also be extended to require a set number of brokers in the system at all times; upon loss of a broker team-member, and if below that threshold, the remaining team-members attempt to find another AAA agent – which can start a *new* replacement broker (assuming the required infrastructure support exists).

Snyder *et al.* [2004] describe robustness within the *Cougaar* architecture, which detects agent failure using sentinels. Failure requires the agent be replaced with a replica; agents maintain collections of backup replicas using either an *active* or *passive* strategy. Active replication entails replica(s) state being synchronized with the individual every time a task is performed, reducing the time to bring that replica online. The active strategy is ideal for scenarios with high failure rates or tight time constraints upon recovery, but carries significant resource cost as the original agent must synchronize replicas immediately after every activity. Passive replication uses *checkpointing*, where the entire system state is periodically persisted to non-volatile storage. Agent failure is addressed through restarting that agent (effectively re-initializing it) and restoring mental state using the last stored checkpoint. Additional communication may be performed to reconcile state inconsistencies between agents – for example, where a partially executed delegated task has state changes not recorded in the last checkpoint. As *unreliable failure detectors* are used to detect failure, incorrect replication cases (from incorrect failure diagnosis) must be detected and corrected.

Guessoum *et al.* [2005] define an approach for dynamic replication based upon *criticality*. They focus upon the *macro* organization of the system which emerges dynamically at runtime and cannot be anticipated in advance (i.e. to specify replica groups). Under their approach, the agents to replicate (and how many times) are determined based upon a combination of their criticality and available system resources. Criticality derives from the volume and type of messages sent between agents, indicating the dependencies upon that agent. Agents involved in more messaging activity are judged as more critical – i.e. more system agents would be impacted by their loss. Replication is performed by sentinels in response to agent failure, or as a preventative measure (to create redundancy or provide additional load handling). One issue is that,

particularly if revising the macro-organization graph at short intervals, criticality may not correlate with messaging – for example, where an agent is prioritizing performing a critical task ahead of messaging.

A dynamic replication approach is also suggested by de Luna Almeida *et al.* [2007], who define criticality based upon the current *plans* of agents. Plans are modelled as directed acyclic (AND/OR) graphs. Constituent activities have their criticality scored through (a designer specified) *absolute criticality function*, which does not account for the plans of other agents; this considers elements including the number of alternate agents capable of that activity, resource requirements or further domain specific factors. *Relative criticality* is also calculated for activities, this time considering other agent's plans, based upon the value their results hold for the system as a whole. Overall plan criticality is calculated using the criticality of constituent activities. Determination of which set of agents to replicate is viewed as an optimization problem, solved by identifying the set of replicated agents offering greatest global utility. Global utility is calculated by combining individual replica utilities (determined by criticality of the replicated agent combined with the probability of it failing). It is worth noting this approach fundamentally relies upon the criticality functions resulting in optimal replica allocations – placing corresponding requirements upon analysis and specification by the designer.

Replication can be proactive – by providing redundancy – or reactive – to replace failed agents; it is also not solely robustness-centric, and may be employed to improve performance through local balancing or parallelization (Deters [2001]). Replication may complement role filling approaches – e.g. replicating agents to fill unoccupied roles – and is similarly concerned with preserving the meta-organization for the overall MAS, rather than ensuring correct individual *behaviour* or maximizing goal achievement. Redundancy is often constrained by resource availability; the replication approaches discussed here often focus upon dynamic provision (including replacement) of agents to avoid the cost of (potentially unused) anticipatory provision. We opt to assume replication or redundancy in the agent system exists at an organization level, and will be transparent to the BDI reasoning cycle our approach focuses upon. We have required agents to hold knowledge regarding their capabilities – this information *could* also be employed by targeted replication.

4.6 Conclusion

This chapter discussed a number of definitions for robustness. Drawing from Hägg [1997], and through considering our focus upon avoiding plan (and activity) failure, we opted to define robustness as *maximizing goal achievement under perturbation*.

A variety of approaches towards robustness were also described, some of which address different aspects to our plan-centric contribution – allowing us to form assumptions regarding aspects of agent operations we specifically do *not* address[3]. These also suggested mechanisms for adoption of responsibility for, or communicative requirements of, robustness in distributed systems.

We formed the following assumptions and requirements;

- The efficacy of our approach is to be measured through goal achievement rate under perturbation; the latter defined as the rate of exogenous change.
- Our own approach as is assumed to lie within a general 'ecosystem' of robustness methods; we restrict our focus to intended plan execution within the BDI reasoning cycle.
- Approaches concerned with meta-organizational correctness, or handling sensory or communication corruption, are assumed as outside the scope of our consideration.
- Agent teams are assumed decompositional and hierarchical; we require an approach similar to exception propagation mechanisms to propagate responsibility when threats to delegated activities cannot be addressed by obligants.
- Propagation of *responsiblity* requires aggregation of threats, where appropriate.
- Dynamic, goal-duration organizations arise from dependency relationships formed for distributed plan execution; our approach must consider both obligant and dependant roles within activity delegation.
- The cost of capability specification, both in boolean and qualitative terms, is assumed to be partially justified through the utility of such information in other robustness approaches.

[3] We do not assume existing methods guarantee *perfect* efficacy – only that their effectiveness lies outside the scope of our contribution.

Chapter 5

Planning

Plans are critical in rational, goal-orientated behaviour. Although our approach is concerned with plan execution, the information required to detect and address threats to existing planned activities will likely mirror that required to *select* activities during plan formation – this chapter discusses plan representation and formation, before the following chapter considers methods for handling potential activity failures stemming from environmental uncertainty.

5.1 Planning and Plan Execution

A plan is a set of steps that, when scheduled according to ordering constraints and performed within a given initial state, achieve a particular goal. Figure 5.1 depicts a generalized automated planning process, based around use of a deterministic world model given by system $\sum$, where the next state is determined by the activities executed and (if applicable) exogenous changes. The *Planner* produces a Plan based on some initial state, objective (goal) and world model, which is scheduled and executed by the *Controller*. The Controller observes activity outcomes and, in the case of online planning, informs the Planner of Execution Status to allow plan revisal. Although the simplest goal specification is a set of states, others are possible; i.e. avoiding particular states, performing specific tasks, or optimizing some value (Nau [2007]).

Plan operators represent activities possible in the domain; typically defining preconditions and effect sets – i.e. state constraints required to be met before, and state changes resulting from, successful execution. The *qualification problem* (McCarthy [1958]) states that, in a realistically complex environment, it is impossible to enumer-

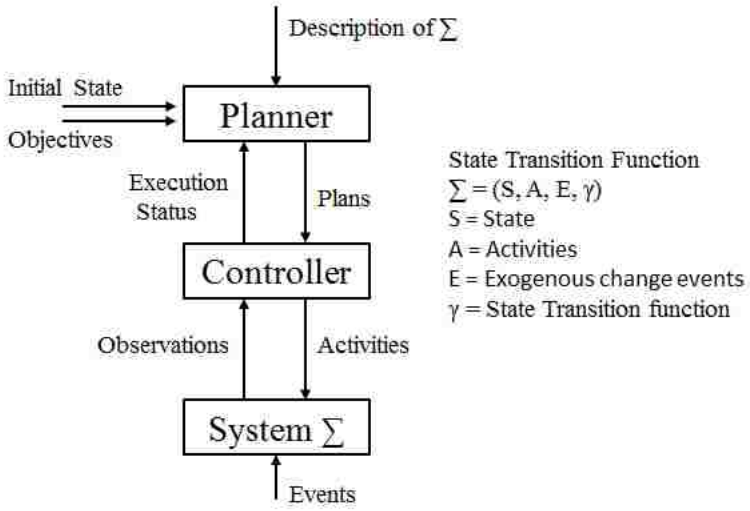


Figure 5.1: Generalized automated planning process from Nau *et al.* [2004]; ∑ represents a (necessarily abstracted) deterministic model of the world.

ate all state combinations that *may* prevent success – and that doing so would over-constrain an operator to the point of unusability. In practical terms, preconditions define – selected based on some degree of significance – states required to execute an activity without *guaranteed* failure.

Determining the sequence of activities required – the *planning problem* – is separate from the *scheduling problem* of determining when to execute each activity. Planned activities may execute to completion without (detected) error but not achieve their stated effects, if change can also occur through environmental events or the actions of other entities. The following sections overview classical planning and hierarchical planning approaches, in order to examine different types of plan structure and the information employed. We also describe multiagent planning approaches to consider the additional information required to reason over delegated activities.

5.1.1 Classical Planning

Classical planning (plan formation) assumes a deterministic, static and finite domain, with fixed goals and an implicit notion of time. Although classical planning is regarded as domain independent, these assumptions restrict the set of *plausible* domains (Nau [2007]). A deterministic state model consists of some finite set of activities A, finite set of states S (where a state is some set of propositions defining the condition of

the world), and a state transition function f. A *Discrete Control Problem* (DCP), as defined by Bonet and Geffner [2001a], solved by finding the activities to move from initial state s_0 to a goal state G, is modelled through the following:

- The state space S – the possible world states
- Initial state $s_0 \in S$
- Activities $A(s) \subseteq A$, which can be performed in each $s \in S$
- A deterministic transition function $f(s,a)$ defining the effect of executing $a \in A(s)$ in $s \in S$ (i.e. giving post-execution state s')
- A function $c(a,s) > 0$ giving the cost of performing a in s, employed to identify optimal solutions
- A set G of goal states, where $G \neq \emptyset$ and $G \subseteq S$

A DCP solution is an activity sequence $a_0, \ldots, a_n$ – i.e. a plan – that, when executed, results in the state sequence $s_0, s_1, \ldots, s_n, s_{n+1}$ where $G \subseteq s_{n+1}$. This requires a_i be performable in s_i ($a_i \in A(s_i)$), with s_{i+1} representing the outcome of executing a_i in s_i (given by $f(s_i, a_i)$). An *optimal* solution achieves G with minimal cost ($\sum_{i=0}^{n} c(s_i, a_i)$).

Planning is essentially a search problem, where the planner traverses the search space to find a plan executable in s_0 and ending with achievement of G (Hendler *et al.* [1990]). For example, *state space search* views the set of possible spaces as a directed graph; each node represents a world state, with directional arcs between state nodes representing the outcomes of performing particular activities in that state. A plan represents a *path* – with actions defined by the traversed arcs – from root node s_0 to the leaf node achieving G.

The *Stanford Research Institute Problem Solver* (*STRIPS*) (Fikes and Nilsson [1971]) has been widely used as a standard representation for classical planning problems. STRIPS defines a planning problem $P = \langle F, O, I, G \rangle$, where:

- F is a set of boolean variables
- I gives the initial state (i.e. s_0)
- G is the goal state (i.e. s_G)
- O is the set of operators (i.e. A)

We focus on the information used to select activities during plan formation. Each $o \in O$ defines an activity type in terms of a signature and three sets of atoms from F:

- Preconditions $Pre(a)$; required true to execute a

- Add effects $Add(a)$; added following execution of a
- Delete effects $Del(a)$; removed following execution of a

McDermott *et al.* [1998] defines the *Planning Domain Descriptor Language* (PDDL), which extends a STRIPS-like formalism with support for type definitions (i.e. for world objects or to constrain operator parameters). Unlike STRIPS, PDDL operators can have negative preconditions or effects – respectively requiring a condition be false, or causing it to be not true. Both preconditions and effects can have quantifiers (expressed numerical conditions); effects may be conditional, i.e. depend upon execution context. Later extensions of PDDL support numeric fluent values (Fox and Long [2003]), allowing definition of plan metrics (e.g. to minimize cost or execution time), and for durative effects in discrete (at the start, end or throughout execution) or continuous form (e.g. gradually decreasing fuel during execution).

One issue with classical planning is time complexity; Bylander [1994] state forming an optimal plan using STRIPS operators is NP-complete. *Heuristic* techniques can improve *common*-case performance, albeit with potential inefficiencies for worst-case scenarios. Heuristic functions estimate the (minimum cost) distance to the goal from a given state ('scoring' desirability of potential expansions), which is then used to arbitrate between search options. Heuristics should be *admissable* – i.e. never overestimate distance from the current search node to the goal (Pearl [1984]) – to generate optimal plans.

A* pathfinding employs a *domain specific* heuristic, favouring movement to locations with lowest Manhattan distance to the destination. Domain-*independent* heuristics solve a *relaxed* version of the problem to identify a lower bound cost estimate for the unrelaxed domain; for example, *Fast-Forward* (FF) (Hoffmann [2001]) and *Metric-FF* (Hoffmann [2003]) form the relaxed domain by removing operator *delete* effects (Bonet and Geffner [2001b] employ a similar relaxation).

5.1.2 Hierarchical Task Network (HTN) Planning

Hierarchical Task Network (HTN) planning (Tate [1977]) is a *domain configurable* planning method – i.e. using domain specific information to guide planning (Kandiyil and Gao [2012]). HTN planners offer speed improvements over classical planners through encoding domain-specific standard procedures as methods. This restricts the

planning search space and captures procedural knowledge (Sohrabi *et al.* [2009]) – but requires discovery and encoding of such knowledge.

Like classical planning, HTN planning represents activities as deterministic state transitions (Nau *et al.* [2004]), with world states defined as sets of atoms – but rather than achieve goal states, HTN planners aim to perform *tasks*. Erol *et al.* [1994] define a *goal* task in the form *achieve*[l] (for some literal l); a solution is a primitive task network with constraints influencing ordering and scheduling (i.e. a *partial order* plan), resolvable to a total order plan, and formed through iterative decomposition. Nau *et al.* [2004] define an HTN planning problem $P = \langle s_0, w, O, M \rangle$:

- s_0 is the initial state
- w is the *initial task network*, to be refined to a set of primitive tasks
- O is a set of operators, M a set of methods (also referred to as *expansions* or *refinements*), forming the domain $D = (O, M)$
- A solution to P is one that performs all tasks in w

M defines known (predefined) task decompositions. Each $m \in M$ can be described by $m = \langle name(m), task(m), subtasks(m), constr(m) \rangle$:

- $name(m)$ defines a signature $n(x_1, \ldots, x_k)$; n is a unique method symbol and $x_1, \ldots, x_k$ define variables which may appear in m
- $task(m)$ is the non-primitive task decomposed by m
- A task network is a pair $w = \langle U, C \rangle$:
  - U defines which subtasks ($subtasks(m)$) need to be performed
  - C defines constraints ($constr(m)$) upon U, such as for ordering, variable instantiation, or defining literals required true before or after

A task $t(r_1, \ldots, r_k)$ is *primitive* if t corresponds to an operator, and *ground* if all terms r are ground. Similarly, a task network is ground if all tasks within ($\{t_u \mid u \in U\}$) are ground. HTN planning algorithms use continuous selection and application of refinement methods (Fig 5.2) to replace (decompose) every *non-primitive* task network in a problem with a primitive task network; these primitive tasks can then be scheduled and performed, with a *plan* being a sequence σ of ground primitive tasks. The resultant plans are decompositional task hierarchies.

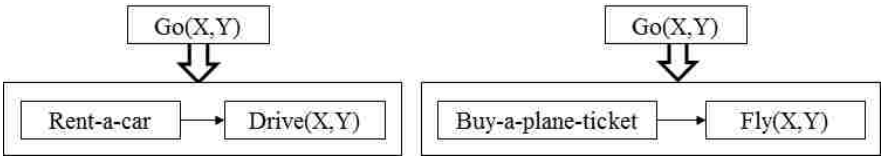


Figure 5.2: Example of possible task decompositions, based upon Erol *et al.* [1994], showing two possible refinements for the task to *Go* from X to Y.

The first true HTN planner, *NOAH* (Nets Of Action Hierarchy) (Sacerdoti [1975]), committed to an abstract solution at the top level of the task hierarchy before progressively decomposing successive levels. *NONLIN* (Tate [1977]) added the ability to backtrack at all levels of planning; 'retracing' steps to consider alternate decompositions following a faulty choice or inability to find a solution. NONLIN was succeeded by O-Plan (Tate *et al.* [1999]) and subsequently I-X (Tate [2001]), which supported mixed-initiative planning – allowing use of human expert domain knowledge to add constraints to the planning task and guide automated planning at key points.

These planners are notable for their practical use – the *Optimum-AIV* planner, based on NONLIN and O-Plan, was used by the European Space Agency to provide automated planning support for spacecraft production (Drabble *et al.* [1997]). I-X has also been used in domains including military coalitions (Allsopp *et al.* [2002]), small army unit co-ordination (Tate *et al.* [2000]), disaster response/rescue (Siebra and Tate [2006]) and non-combatant evacuation (Wickler *et al.* [2006]). The Simple Hierarchical Ordered Planner (SHOP) by Nau *et al.* [1999] and it's successor, SHOP2 (Nau *et al.* [2003]) represent further examples of HTN planners with widespread practical application; including within domains such as evacuation planning, terrorist threat evaluation, UAV control and manufacturing (Nau *et al.* [2005]).

5.2 Multiagent Planning

Distributed planning occurs when multiple agents participate in planning and/or plan execution. Cox *et al.* [2005] defines multiagent plan as a tuple $\langle A, E, CL, CC, NC \rangle$;

- A is a set of activities to be executed
- E are precedence links, establishing ordering constraints between activities
- CL are causal links, describing where activity effects provide a state required by another's preconditions

- *CC* and *NC* contain concurrency or non-concurrency constraints.

Each agent forms or holds a plan *P* for a task, and identifies the required *A* by decomposing the root task into subtasks. The resultant *A* contains non-decomposable (primitive) tasks; i.e. leaf tasks that can be scheduled (respecting constraints in *E*, *CC* and *NC*) and executed – with *CL* information of use in supporting reactive plan repair.

Durfee [2001] describes distributed plan formation as a 5-step process, subsequently generalized by de Weerdt and Clement [2009];

1. Allocate goals to agent
2. Refine goals into subtasks
3. Schedule subtasks by adding resource allocation and timing steps
4. Communicate planning choices (of prior steps) and resolve any conflicts
5. Execute the plans

Distributed plans may be formed using a centralized or distributed approach; the former may offer better plans by employing centralized global knowledge (including use within heuristic functions), but risks becoming intractable due to the state-space increase arising from reasoning over a large set of agents and associated activities (Jonsson and Rovatsos [2011], Nissim and Brafman [2012]), or infeasible due to the entire processing burden of planning being placed upon a single agent (Nissim and Brafman [2014]). A further disadvantage is that the centralized planner's assignment of activities may impair individual agent autonomy, compared to allowing individuals to form local plans (in a distributed planning process). Distributed planning and execution is likely to be employed in complex and realistic environments, with agents using local (specialised) knowledge to contribute parts of the distributed plan; this may also potentially improve efficiency through parallelization (Ephrati and Rosenschein [1997]).

The following section discusses some specific approaches for distributed plan formation. These approaches inform the capability meta-knowledge required for our robustness reasoning with regard to multiagent plans.

5.2.1 Private/Public Actions

Brafman and Domshlak [2008] describe an approach for MAP using a *public/private* action[1] concept. Atoms are *private* or *public* – a private atom is neither required as a precondition for, nor an effect of, any actions of another agent. The set of agent actions is partitioned into private/internal and public action sets – public actions have preconditions or effects containing public atoms, and consequently will require co-ordination. This partitioning is used to form the *agent-interaction graph* IG_Π (similar to a *causal graph* defined in Brafman and Domshlak [2006]). IG_Π shows agent relationships in terms of their action's abilities to supply or destroy states required by preconditions of others, indicating coupling in the domain; they show worst case complexity of planning as associated with the degree of inter-agent activity coupling. It is assumed loose coupling is a natural property of a MAS, where *'substantially autonomous'* agents execute more internal actions than coupled public ones.

Nissim *et al.* [2010] implement the MAP algorithm *CSP+Planning*, which splits planning into public and private aspects. The *public* aspect is expressed as a Distributed Constraint Satisfaction Problem (DCSP), solved by finding a minimal length sequence of public actions; this solution ensures all *public* atom preconditions are met and the goal is achieved. The *private* aspect is performed by individual agents using a local planner (*FF* by Hoffmann [2001]). That planner will identify sequences of internal actions, executed between public actions to establish *private* precondition atoms. This provides local consistency for the public action plan, ensuring private *and* public precondition atoms hold for the public actions.

5.2.2 Partial Global Planning

Partial Global Planning (PGP) is a framework for co-coordinating distributed problem solving, focused upon scheduling (Durfee and Lesser [1991]). PGP adopts the principle of '*co-ordination through local reasoning*', where plan information sharing between agents during plan formation and execution allows co-ordination to arise from *local* behaviour. Agents use shared information on their local plans to form Partial Global plans – these represent the holding agent's knowledge of the collective plan-

[1] Although we primarily use 'activity' in this thesis, 'action' is employed where necessary to maintain consistency with Brafman and Domshlak [2008] or if used by others employing or extending a public/private action model.

ning process for a given goal, and indicate which agents should be informed of the results of local plan execution. PGP changes may trigger local plan changes to account for co-ordination requirements (Decker and Lesser [1992]). The exchange of PGPs, combined with proposal based negotiation to resolve conflicts, allows gradual convergence upon a shared plan. PGP has optimal performance and non-optimal but adequate performance in dynamic environments. It offers coordinated behaviour through designed *local* behaviour of agents; showing neither centralized control nor total global knowledge are essential for distributed planning.

5.2.3 Generalized PGP and TÆMs

PGP was designed around sensor systems formed of homogeneous agents (Durfee and Lesser [1987]); *Generalized* Partial Global Planning (GPGP) views agent coordination as a distributed search of a dynamically evolving goal tree (Lesser *et al.* [2004]). GPGP aims to maximise overall quality attained by agent groups, accrued by achieving high level goals within time constraints – higher degrees of coordination lead to better overall quality and shorter execution time. GPGP was extended by *SHAC* (**SH**ared **A**ctivity **C**oordination), which separated modelling and implementation of coordination mechanisms from the planning problem and algorithm (Clement and Barrett [2003]).

Task structures in GPGP are modelled using the TÆMS (**T**ask **A**nalysis, **E**nvironment **M**odelling, **S**imulation) language (Decker and Lesser [1993]). TÆMS represents task-subtask (or goal to sub-goal) decompositions, similar to HTN representations (Vincent *et al.* [2000]), but annotates a continuous *quality accumulation function* (*qaf*) rather than AND/OR types. The *qaf* function defines task quality through combining quality of associated sub-tasks; for example, *q_min* defines the quality as minimum associated subtask quality – equivalent to an AND relationship. Alternatively, *q_max* is equivalent to an OR relationship, defining quality as the maximum of an associated individual subtask (Lesser *et al.* [2004]). TÆMS also models *enables* and *facilitates* relationships (plus converse equivalents) between tasks, defining ordering constraints. Tasks have durations and optional deadline constraints, which can be employed towards calculating their quality.

Each system goal has an associated TÆMS tree, indicating alternative methods (disjunctive decompositions) of achieving that goal; the *qaf* score can be used in select-

ing between multiple options. Agents use a TÆMS representation (initially) of their local activities; this evolves to include the activities (and resultant task relationships) of other agents upon receipt of information from them. The representation provides a partial global model of activities in the system (partial, as it will almost certainly only capture a subset of the global task tree). Coordination requires agents identify which sub-goals to pursue, when, and with what degree of effort; more than one agent may be assigned to a particular leaf node in the task structure.

5.3 Conclusion

This chapter covered automated planning in both a local and multiagent context. Planning and later modification both intuitively involve reasoning over the appropriateness of activities for some expected execution context, and identification (or formation) of causal link relationships. We formed the following assumptions and requirements:

- We assume use of deterministic plans, with activities modelled as state transitions.
- We do not assume or require a *specific* approach for plan generation in intention formation *or* our robustness behaviour, beyond noting heuristic and HTN approaches can improve the speed and viability of runtime planning for such.
- Our capability meta-knowledge model requires information equivalent to a STRIPS operator, to anticipate precondition violation and estimate the execution context for subsequent activities – similar to as used in plan *formation*.
- We require our capability model to provide quantitative estimation of activity quality, which can be applied to counteract the *qualification problem* – i.e. to indicate where preconditions hold yet activity success is not certain.
- Our capability model should also allow reasoning over whether as-yet undecomposed goals or subgoals can be achieved; this requires modelling similar to that of HTN refinements.
- An aggregation approach similar to that of TÆMS' *qaf* is required for qualitative estimation where capabilities represent plan options.
- To cover *multiagent* plans, our capability meta-knowledge model must represent capabilities accessible via dependencies upon others.

- We assume contracts arise from activity delegation; these must convey sufficient information to allow introspective reasoning about that activity by the dependant, similar to information sharing during multiagent plan formation.

- A decentralized robustness approach is necessary due to excessive knowledge and communication requirements for centralized approaches in realistic domains – we require *co-ordination through local reasoning*, as expressed by Durfee and Lesser [1991] for PGP.

Chapter 6

Plan Robustness under Uncertainty

Our contribution is motivated by the risk of activity failure – and associated debilitative consequences – due to exogenous change in realistic domains. This chapter discusses approaches for preventing or recovering from plan activity failure.

6.1 Preventing Failure in Uncertain Environments

This section focuses upon approaches to avoid plan activity failure in uncertain environments – whether by attempting to handle all possibilities within the formed plan, or using mechanisms that defer commitment to specific activities until execution.

6.1.1 Conformant Planning

Smith and Weld [1998] define conformant planning as finding a linear plan (activity sequence) to achieve a goal *regardless of world state* – i.e. covering both uncertainty over activity outcome and lack of sensory ability. Rather than attempting to resolve uncertainty through sensing, the planner seeks to 'force' the world into a certain state. The planner must account for *any* possible outcome (including side-effects) modelled in the operator specifications, and form plans applicable for any possible initial state. Palacios and Geffner [2006] describe an example conformant plan, where a robot in a n width grid is assured to reach the rightmost side by performing n moves right. Although we regard conformant planning as primarily concerned with managing uncertainty, this approach can improve robustness through avoiding the plan failures stemming from such - and thus aiding goal achievement by plan-executing agents.

Conformant planning can be modelled by extending the DCP model (Section 5.1.1) to reason over the space of *possible* belief states; where the initial state is no longer assumed known and activity outcomes may be non-deterministic. The initial state s_0 is extended to cover a set S_0 of possible initial states; $f(a,s)$ is similarly replaced by $F(a,s)$, which maps to a set of *possible* effects of a (i.e. $s' \in F(a,s)$). A solution to a conformant planning problem is a plan that achieves, with certainty, the goal for any $s \in S_0$ and for any possible $F(a,s)$.

As conformant planning considers a multitude of possible states, it is significantly more difficult than classical planning – Turner [2002] show conformant planning as $\sum_3^P$-complete for plans of polynomially-bounded length, dropping to $\sum_2^P$-complete if activities are deterministic and executable. Son *et al.* [2005] suggest one approach for reducing complexity to NP-complete by approximating a single initial state using *0-approximation* (Baral and Son [1997]) to determine initial state beliefs based upon where a given state is constantly true (or false) in all possible initial states (beliefs only true in *some* initial state possibilities are treated as *unknown*). However, this approach is unable to capture non-trivial disjunctive inference (Son and Tu [2006]).

Palacios and Geffner [2006] describe a method for solving *some* non-trivial conformant problems through forming equivalent classical problems, to be solved by a classical planner. Atoms are introduced to represent conditional beliefs; $L \mid X$ represents that if X holds, then (given certain invariants) L must hold (X can represent a disjunction, e.g. if $X_1 \vee \ldots \vee X_n \subset L$ and $X_n \subset L$, L holds). This allows the planner, given knowledge of X_i, to conclude whether L holds. Their approach is, however, not applicable to *all* conformant problems and assumes activities are deterministic – meaning uncertainty must lie *only* in the initial state – excluding where environmental uncertainty includes exogenous change during plan execution.

The increased possible state space arising from exogenous change during execution significantly complicates conformant planning, and – combined with the general complexity of such planning approaches – likely renders this type of approach infeasible for failure avoidance in realistically complex environments. One further risk is that it may simply be impossible to form a conformant plan in certain domains; to extend the earlier example of Palacios and Geffner [2006], a robot may ensure it is rightmost by executing n moves right on an n width grid – but this is not practical if moving right

too many times risks damage from hitting a boundary wall.

6.1.2 Contingent Planning

Contingent or *conditional* planning handles uncertainty and partial observability by inserting conditional branches into the plan. Decisions on which branch to execute are deferred until execution, after any sensing activity, to employ more accurate information than available at planning time. Bonet and Geffner [2000] define contingent planning as a *non*-deterministic control problem. A solution (a contingent plan) is a graph where nodes equate to some belief state b, arcs denote the state transition for a performed in b ($a(b)$), and each node has a successor b_a^o corresponding to the beliefs resulting from $a(b)$ – where a path can be found to achieve G, accounting for uncertainty through sensing activities and selection of conditional branches.

In one example, *Cassandra* (Pryor and Collins [1996]), forms partial-order plans including distinct information gathering activities. Cassandra distinguishes preconditions for validity and conditions for selection; defining when an activity is *possible* versus when it is *necessary*. Exogenous events are assumed *not* to occur; sources of uncertainty are also assumed as known.

A balance has to be struck regarding plan branches – too few restrict flexibility, but too many risk execution time being dominated by branching and sensing. Dearden *et al.* [2002] implemented a utility function to determine branch placement based upon (probabilistically estimated) likely failure points. Albore *et al.* [2007] observe contingent plans risk exponential growth with the number of possible observations (sensed effects) following activity execution. They partially address this issue using successive relaxations to a conditional problem to form a conformant and then classical problem, but solely determines the *next* activity to perform rather than forming an entire plan – meaning this is not viable if advance identification and reservation of resources is required.

Conditional planning allows a degree of fault prevention to be built into plans; but the extent of that robustness depends upon sufficient branch coverage for possible failure cases. A further issue arises if enumerating all conditions is intractable, or where the planner lacks this knowledge.

6.1.3 Markov Decision Processes

Markov Decision Processes (MDPs) model activity in stochastic domains, and can be used to form *policies* guiding agent behaviour. An MDP can be modelled as $\langle S, A, P, R, C, \gamma \rangle$, where;

- S defines the state space – a finite set of possible states
- A gives the (finite) set of activities
- P gives a transition probability $P(s,a,s') \rightarrow \mathbb{R}$ indicating the likelihood of reaching state s' by executing a in state s; these represent the *Markov Assumption* that the next state derives solely from s and a
- R is a reward function $R(s) \rightarrow \mathbb{R}$, giving the utility for being in s
- C is a cost function $C(a,s) \rightarrow \mathbb{R}$ indicating the cost of performing a in s[1]
- γ is a discount function $\gamma \rightarrow$[0:1]

An MDP solution is a *policy* π, where $\pi(s) \rightarrow a$ gives the optimal activity a to be executed in state s. Policies are generated using a function defining the utility of performing a in s; $V(s,a) = R(s) - C(s,a)$. The *history* h gives the sequence of activities executed prior to s through following π, allowing determination of the policies' cumulative value;

$$V(h|\pi) = \sum_{i \geq 0} \gamma^i R(s_i) - C(s_i, \pi(s_i))$$

A solution to an MDP is an *optimal* policy, i.e. giving maximum utility over all other possible policies. When translating a classical planning problem into an MDP representation, goal and non-goal states can be given non-zero and zero rewards respectively. The discount function γ reduces rewards associated with later states, bounding the maximum total activity cost for following π.

MDPs assume complete knowledge – that the world is fully observable and states known. This assumption is removed by *Partially Observable* MDPs (POMDPS), which reason over observations rather than states. Observations indicate current state; O defines a finite set of observations, and $P_a(o|s)$ the probability of observing $o \in O$ when state s is reached from activity a. Decision making uses the history of previous observations to form a probability map, allowing the actual state to be inferred and a solvable MDP defined.

[1] In some formalizations, an MDP definition may have only an R or C function determining utility; we have opted to define both in line with Nau *et al.* [2004].

Boutilier [1996] suggests a method for *Multiagent* MDP (MMDP) based planning. He suggests that, given a common reward function, agents will form the same individual policies. Co-ordination can be reduced to be only being required where agents have *multiple* optimal joint activities in a given state, and is viewed as an n-player game aiming to converge on a Nash equilibrium – such that all agents select the same, optimal, joint activity. Convergence may be achieved through use of conventions – manually specified or identified through reinforcement learning – to identify the policies of involved agents. This simplifies planning, as agents only need to consider a subset of co-ordination 'games' rather than compute a global coordination policy covering the whole MMDP. However, their assumption of full observability – of agents involved in a problem and their possible activities – may not be feasible for realistic environments.

While policies can offer optimal behaviour, complexity issues render *identifying* them intractable as state space increases. This is exacerbated for POMDPs, where state space is further expanded due to the probabilistic nature of observations. In contrast, Schut *et al.* [2002] show BDI agents are able to handle domains that are relatively simple yet intractable for MDPs, and with approximate performance to MDP (albeit depending on the time costs of runtime planning).

Attempts to improve MDP tractability typically involve abstraction – simplifying state spaces at cost of policy optimality (Boutilier and Dearden [1994]) – or determinization to employ classical planning within the policy generation process. Guestrin *et al.* [2001] developed an approach using *factored* MDPs – representing the MDP as a dynamic Bayesian Graph – for multiagent planning; they suggest this approach reduces computational complexity to tree-graph width (of a co-ordination graph used for inter-agent negotiation) from the exponential complexity of MDP approaches. The *ReTrASE* (Regressing Trajectories for Approximate State Evaluation) MDP solver uses determinization to support state space aggregation, and was shown to have superior performance to leading planners on several IPPC (International Probabilistic Planning Competition) domains, although incomplete Probabalistic PDDL support prevented evaluation in *all* domains (Kolobov *et al.* [2009]).

Work has sought to reconcile both MDP and BDI approaches. Simari and Parsons [2006] suggest mapping between policies and intended plans, extracting the latter by

projecting future activities selected through a policy (assuming the maximum probability state transition occurs). They also present a converse method for forming policies from existing deterministic plans, for where the state search space is too large for MDP solution. Pereira *et al.* [2008] further extend this with an approach to form deterministic plans from offline-formed POMDP policies.

Aside from tractability issues, MDP approaches risk transition probability information being unavailable or impractical to learn. MDP specifications are also non-intuitive, restricting their *practical* usability. Meneguzzi *et al.* [2011] suggest a method to map more intelligible HTN domains onto MDPs – although this defines probabilities based upon state presence within operator preconditions, rather than probabilities in the *environment*. We argue MDP approaches are unlikely to be feasible in the complex realistic environments our contribution targets. Although tractability issues associated with MDP and POMDP approaches can potentially be addressed with abstraction or approximation techniques, we assume the degree of abstraction required for a realistic, complex domain would overly compromise the optimality of any generated policy, making the outcome no more ideal than deterministic planning (if not worse).

6.1.4 Continual Planning

Continual planning treats plan revision as a continual process by interleaving planning, execution and monitoring. In the most extreme case this extends to *reactive* or *dynamic* planning, where only a single next activity to execute is determined in each given instant. For example, Schoppers [1987] describes synthesis of *universal plans* that define conditional rules for selecting which activity a robot should perform for any given situation. This resembles the use of MDP Policies, with similar difficulties stemming from enumerating all possible states (in universal plans) or the cost of determining the next activity after every execution.

desJardins *et al.* [1999] suggest agents employ continual planning in dynamic and partially observable environments, where time constraints prevent formation of a complete plan, or where goals may change over time. Intended plans can be incrementally extended during execution, potentially including sensory activity, until the intended goal is achieved, invalid or impossible. For example, Pellier *et al.* [2014] suggest an approach based upon *Moving Target Search* (MTS) algorithms. MTS algorithms are

employed in domains such as where an agent follows some moving entity – requiring constant plan modification to account for the target's unpredictable movement.

A typical approach is to define plans containing *abstract* activities or subgoals, refined during execution; such as within the Procedural Reasoning System (PRS) (Ingrand *et al.* [1992]) and *Jason* (Bordini and Hübner [2006]) agent framework. These systems ultimately use hierarchical plan structures; the initial plan is an upper 'layer' of abstract steps, with specific decomposition during execution (and using more current knowledge). Continual planning approaches risk *shorter* term refinements introducing effects that inadvertently stymie the longer term goal; Clement and Durfee [1999] state any abstract plan must still be specific enough to avoid (the majority of) conflicts between refinements. A secondary risk is the loss of necessary resources to contention, due to failing to identify and reserve them in advance.

Brenner and Nebel [2009] present an approach postponing 'unknown' parts of the planning operation, allowing execution to begin without a complete plan. Their *Multi-agent Planning Language* (MAPL) models the presence or absence of knowledge; operator definitions can represent knowledge requirement preconditions and the knowledge gathering effects of sensing activities. Active information gathering – planned sensing – allows planning to be resumed once required information is known. The postponement of planning decisions does risk agents being caught in computational or logical 'dead-ends'; they argue the alternative is failing to act entirely *or* facing (computationally intractable) contingent planning to cover *all* possible circumstances. Their continual planning algorithm iterates through three phases; (re)planning for goals from the current state, executing plans, and perceiving world state changes.

MAPL defines *Assertions* – representing goals or composite activities that cannot be currently refined, combined with *replanning conditions*[2] that define knowledge required to perform that refinement. This allows the planner to both form plans with abstract future activities, and plan to gather the information required to refine them. Plan monitoring is used to determine if the plan has become obsolete due to the effects of an assertion expansion or world state changes; appropriate plan repair or replanning can then be employed using current knowledge.

[2]In this context, 'replanning' refers to the performance of further planning options to refine that assertion.

Continual planning risks shorter-term decisions stymieing the longer term goal; including failure to identify and secure required resources. This can be partially mitigated against in approaches such as used by the *Jason* BDI framework (Bordini and Hübner [2006]). *Jason* agents use libraries of pre-defined plan recipes, which can include subgoals which are only refined upon execution based upon the agent's beliefs at that time. Such approaches allow resource reservation by intermixing specific and abstract activities within intended plans; our robustness approach should should allow reasoning over as-yet undecomposed composite activities or subgoals, particularly as use of this type of plan model within *Jason* evidences it's viability.

6.2 Handling Plan Activity Failure

It is unlikely to be feasible or tractable to form plans that entirely prevent failure in realistically complex and uncertain environments. This leads to a likely requirement for agents to handle plan failure; this section describes several recovery techniques.

6.2.1 Reactive Plan Repair and Replanning

Activity failure can mean preconditions of subsequent activities in the same plan are no longer met, or that goal states are not established as expected. In the literature, the term 'replanning' has varied meaning and usage. Talamadupula *et al.* [2013] defines *replanning as restart* to cover generation of an entirely new plan from the point of failure, and *replanning to reduce computation* as a *minimal modification* of an existing plan. For simplicity, we use *replanning* to refer to the former, and *plan repair* for the latter – reflecting terminology used by Fox *et al.* [2006] and van der Krogt and de Weerdt [2004]. van der Krogt and de Weerdt [2004] describe plan repair as having two aspects, *refinement* and *unrefinement*, corresponding to addition (plan extension) or removal of activities – where only the former applies to replanning. In our BDI context, we use replanning to refer to formation of a new $plan_i$ for a $goal_i$ in a post-failure state, and repair as modification of an existing $plan_i$.

According to Fox *et al.* [2006], plan repair offers greater efficiency and stability in terms of information retention than replanning. Greater stability reduces both unnecessary reservation of resource (i.e. those released from use following total replan-

ning) and the overhead for communicating information regarding plan modifications (Komenda *et al.* [2012]). However, Nebel and Koehler [1992] argue that experimental results showing superior computational efficiency for plan repair do not hold in theoretical worst case scenarios, *if* there is an explicit goal to retain a maximum part of the original plan. Selection between plan repair or replanning approaches to cover *distributed* intentions may be driven by considering responsiveness (time spent to modify or replan) against communications costs (increasing inversely to stability).

Plan causal structure information describes causal links between planned activity, denoting where effects of some activity contribute states required by preconditions of some *subsequent* plan activity – allowing distinction between *side*-effects and those relevant to ongoing execution (states may have *multiple* contributors, i.e. be established by multiple activities). Reece and Tate [1994] describe use of this information to synthesize *protection monitors*, which detect where required causal effects have *not* been established following activity execution (and no alternate contributor activities exist) – allowing invocation of repair. A parallel causal structure represents causal links established by *prior* activity and required by future activity, and is used to avoid or address interference with preserved parts during plan repair.

Drabble *et al.* [1997] describe an approach for plan repair within O-Plan (Tate *et al.* [1999]), where two tables store causal link information. The *Table Of Multiple Effects* (*TOME)* records the effects of activities, which may occur at the start or end of execution. The *Goal Structure Table* (*GOST*) records causal dependencies between activities. Finally, the *TOME and GOST Manager* (TGM) invokes plan repair where a causal link does not hold. Failure of an activity to establish an effect does not necessarily require repair, if *multiple contributors* exist (Tate [1977]).

Their repair algorithm is specified in three parts. First, 'necking' the plan establishes where the last activities completed and inserts a *neck point* – a dummy activity denoting the insertion point for repairs. This also identifies activities scheduled for next execution; the *execution fringe*. If repair was triggered by activity failure, missing effects required by casual links are identified using the GOST; if no alternative contributors exist, a restorative plan is generated and inserted after the neck point. Alternatively, if planning was triggered by exogenous change, a world event activity representing that change is inserted into the plan after the last executed activity. That

change is consequently represented in the TOME, allowing – with the GOST – determination of any impact and repair requirements. Even if the plan is not affected, persistence of that event within the TOME allows detection of any *subsequent* impact. If a repair plan is required, the *end* of the world event activity serves as the neck point.

van der Krogt and de Weerdt [2005] suggest an approach utilizing individual plan repair to form multiagent plans between self-interested agents. Here, agents plan for a particular individual goal, delegating sub-tasks they cannot achieve locally using a blackboard-style auction (and requiring agents to share their capabilities). Plan repair is used to adapt local plans, followed by necessary auctions for added tasks, repeating until a complete plan is formed. This does not assume agents are collaborative, but relies solely upon local repair. Collaboration can still occur through the auction approach – there may be social benefits for self-interested agents (i.e. reciprocal aid when *that* agent requires assistance) to counteract the costs of performing some subtask for another. Their experimental results, evaluated in a logistics domain, suggest efficiency gains in plan repair over replanning[3] and, significantly, reduced decommitment costs due to more limited change to intra-agent dependencies.

Boella and Damiano [2002] describe a plan repair algorithm for BDI agents in environments where exogenous change or non-deterministic (unexpected) activity effects can contradict intended plans. Agents monitor for differences between expected and actual world state, invoking repair where utility is reduced in the latter. The repair algorithm, based on a refinement planning principle, traverses *up* the abstraction hierarchy of the original plan from the activity with violated preconditions. Refinements are 'retracted' until a new (re)refinement with acceptable utility is found *or* no acceptable refinement is found for any level.

Fritz and McIlraith [2007] annotate plan activities with information gathered during refinement planning. These annotations are used to determine whether the current plan remains both valid (preconditions hold) and optimal (no *better* plan can be found given current state) following exogenous change or failure. Replanning occurs only if and when a more optimal plan *can* exist – reducing the computational cost of (their described alternative) replanning upon *every* divergence between expected and actual

[3]This contradicts Nebel and Koehler [1992], who focused upon *worst-case* analysis rather than experimental results.

state. Although aimed at improving replanning efficiency, this method is also applicable for repair; the focus is upon *triggering* remedial behaviour rather than the specific method of performing it.

Talamadupula *et al.* [2013] describe *replanning for multi-agent scenarios*, where replanning or repair operations are constrained by commitments to others; e.g. to maintain or establish states (such as safety responsibilities) or observe time/cost restrictions. They suggest representation of such constraints as *soft* goals – i.e. non-mandatory for success, but with associated rewards for achievement (and converse costs). The failure or exogenous change stimulating repair may render commitments to mandatory 'hard' goals as impossible to meet; soft goals allow MAP replanning/repair to focus on meeting *as many* constraints as possible (i.e. maximize net reward), without being overly constrained by being required to meet *all*.

Komenda *et al.* [2012] define, and subsequently evaluate (Komenda *et al.* [2013]), a notable MAP repair approach. Their algorithm focuses upon definition of planning problems and insertion of generated plans (using *MA-PLAN* by Nissim *et al.* [2010], which employs the public/private approach of Brafman and Domshlak [2008]) to repair a MAP. Three approaches were detailed (here, $P[0,..,k]$ denotes the failed plan consisting of activities a_0 to a_k, with the failed activity being some interim activity a_i);

- *Back on track repair* – forms a repair plan P_{back} that establishes the 'missing' effects associated with the failed activity. This is inserted as a prefix to the remaining plan activities, giving repaired plan $P' = P_{back} \bullet P[i,\ldots,\infty]$ (where $P[i,\ldots,\infty] \neq \emptyset$).
- *Lazy Repair* (LR) – attempts to execute the remaining original plan activities following the failed activity, where the *executable remainder* is denoted as $P[k\ldots\infty]$ (a_k is the activity following the failed activity), before forming a new *suffix* plan (P_{lazy}) to achieve any missing goal states; i.e. $P' = P[k\ldots\infty] \bullet P_{lazy}$
- *Repeated Lazy Repair* (RL) recognizes that LR may see, with repeat failures, repeated concatenation of a repair plan. RL drops any existing repair suffixes before appending a new repair suffix. For example, failure at a_{k_1} and subsequently a_{k_2} would see the repaired plan evolve from $P[k_1\ldots\infty] \bullet P_{lazy-1}$ to $P[k_2\ldots\infty] \bullet P_{lazy-1} \bullet P_{lazy-2}$. RL instead would drop the previous P_1 repair, giving repaired plan $P[k_2\ldots\infty] \bullet P'_2$ – on the basis P'_2 would be shorter than the combined $P_1 \bullet P_2$.

Evaluation in a number of domains shown that plan repair offered reduced communications overhead over replanning in more tightly coupled domains. All three repair algorithms had superior execution length (executed less joint actions) over replanning, with Repeated Lazy Repair showing best performance.

This thesis focuses on BDI agents which we assume to co-operate in multiagent plan execution. This motivates adoption of plan repair over replanning due to lower associated communications costs. While Nebel and Koehler [1992] argue against the *computational* efficiency of plan repair, they do so from the basis of an *explicit goal* to maximize plan retention, and for the worst case – this may not hold for most common case scenarios (such as indicated by experimental results, such as by Fox *et al.* [2006]), or if there is flexibility over the required plan stability.

The approaches reviewed here are defined from a reactive standpoint – i.e. following detected failure, typically defined as where expected activity effects have not arisen, and are vulnerable to where the debilitative *consequences* of that failure hinder recovery. Approaches such as given by Reece and Tate [1994], Boella and Damiano [2002] and Fritz and McIlraith [2007] can be seen as proactive due to detecting plan invalidity from violated preconditions prior to activity execution – a situation that may occur from activity failure or parallel exogenous change. We can adopt similar methods for our proactive approach – i.e. detecting if preconditions of activities within the intended plans of a BDI agent are likely to fail, and responding accordingly.

We have also previously required qualitative evaluation functions within our capability meta-knowledge model, to determine where the *utility* of selected activities has been reduced. Our requirements therefore exceed the boolean model of these approaches, which only consider whether an activity has *failed* or not, to consider where *risk* of failure increases – and requiring an associated mechanism to indicate when such risk merits *pre-emptive* plan repair.

Having adopted a plan repair approach, *HTN* plan repair methods can provide a model for our desired multiagent behaviour. Hierarchical Task Networks resemble the decompositions arising from activity delegation within hierarchical agent teams (Wickler *et al.* [2009]). We view local modification of plans by obligants as similar to re-refinement HTN plan repair; suggesting a distributed plan repair approach based on

this local behaviour. Exception propagation (Section 4.5) style mechanisms can control *when* agents in a decomposing hierarchical team perform plan changes, to restrict distributed plan changes to some minimal subset.

6.2.2 Plan Execution Monitoring

Plan Execution Monitoring (*PEM*) detects and responds to divergence between the actual world state during execution and that assumed by (i.e. during the formation of) a plan. PEM approaches may incorporate plan repair or replanning, and have been employed in partially observable, highly dynamic domains such as robot football (Mendoza *et al.* [2015]) or disaster scenarios (Jarraya *et al.* [2013]).

Wilkins [1985] describes a *replanning* module employed within the System for Interactive Planning and Execution monitoring (SIPE). Given a plan, world state and some unanticipated situation SIPE seeks to transform a threatened plan into an executable one with minimal changes[4]. SIPE first discovers the current situation and identifies resultant problems – such as missing knowledge or violated casual links – with detected problems then addressed through selection of a *replanning action*. This action may re-instantiate an activity with alternate variable values, insert a conditional branch, or replace a threatened activity with a new subgoal (stimulating planning to form and insert a new (sub)plan).

IPEM - Integrated Planning, Execution and Monitoring - interleaves planning and execution, responding where activity preconditions are violated by failure or exogenous change (Ambros-Ingerson and Steel [1988]). IPEM utilizes continuous planning to support problems otherwise unsolvable due to partial (initial) knowledge; an initial partial plan is refined using IF-THEN rules that define transformations for detected flaws. Flaw types include false preconditions, conflicts between potentially parallel actions or unexpanded/non-primitive actions; responses include inserting or re-ordering actions, or insertion of sub-plans. The IPEM scheduler uses a priority ordered queue (*agenda*) of tasks to resolve potential conflicts between fixes – each task consists of a flaw and set of candidate fixes, both ordered using a domain specific heuristic. If a flaw cannot be addressed, IPEM iteratively backtracks using a stored history of planning decisions; if necessary, resulting in complete re-planning. Like SIPE, IPEM utilizes

[4]Although described as a replanning module, this *minimal change* approach matches our earlier distinction for plan repair.

hierarchical decomposition to facilitate local repair.

The *Continuous Planning and Execution Framework*, or CPEF (Myers [1999]), uses *monitors* that, when triggered, evoke responses ranging from user alerts, to plan repair or invocation of standard operating procedures. Various monitor types exist – *failure* monitors define responses to activity failures[5], *knowledge* monitors detect the absence of required information, and *assumption* monitors detect differences between observed world state and that assumed by plans. Assumption monitors are similar to protection monitors in Reece and Tate [1994] and, similarly, can be synthesized.

CPEF models various types of failure. *Precondition* and *action* failure respectively occur due to absence of some precondition state(s) before execution, and absence of a stated effect afterwards. *Unattributable* failures occur where some (automated or manual) assessment deems the current plan unacceptable, even if no failure occurs. As some domains permit individual failures to occur, such as due to inbuilt redundancies, *Aggregate* failures types capture where a *combination* of failures constitutes a more significant event. Plan repair is employed by CPEF, when triggered by monitors, using a principle of minimal modification and maximum stability – as CPEF uses hierarchical plans (using the SIPE-2 planner by Wilkins [1991]), backtracking re-refinement is used, terminating when parents of failed plan nodes are successfully re-refined.

The PKS (Planning with Knowledge and Sensing) system uses a *knowledge based* approach to planning, where the planner considers how knowledge state – rather than possible environment state – is changed through activities. PKS models activities in terms of knowledge requirements (with preconditions equating to queries) and effects (what is known after execution). This entails storage of information in five databases (Petrick and Bacchus [2002]); conveying facts which are known (of boolean or continuous types), which will become known (i.e. sensed) *during* execution, those whose values are unknown but are known to come from a disjunctive set (allowing inference of conditional branches), and of *Local Closed World* knowledge (Etzioni *et al.* [1994]) – where sensing *can* return exhaustive information, even when the open world assumption applies in *general*).

[5]CPEF supports both direct and indirect execution – the latter being observation of some *other* system executing the plan.

Goals are represented through information queries. PKS generates conditional plans to account for *potential* knowledge cases – for example, branching to cover both cases where a variable is (discovered as) true or false. These can be linearized into execution paths with conditional rules applied to perform both backwards and forwards inference (Petrick and Bacchus [2003]), allowing definition of temporal constraints (e.g. that some state should hold at the end or over the duration of a plan). In one example, PKS was used in controlling a bartender robot (Petrick and Foster [2012]). This facilitated social dialogue, as representation of activities as knowledge transitions allowed interaction between the human customer and bartender robot to be modelled in planning terms. The robot responded to missing information by reforming its plan to include knowledge gathering; e.g. repeating a drinks order request if unable to understand the customer's response.

Another example of PEM is found in *I-Globe* (Komenda *et al.* [2009a]), which integrated I-X mixed-initiative planning (Tate [2001]) with the A-Globe agent platform (Šišlák *et al.* [2005]) and the AGENTFLY multiagent UAV control system (Sislak *et al.* [2012]). The experimental domain (Fig 6.1) presented a dynamic environment where I-Globe agents hunted terrorists (using police and intelligence gathering UAV agents) and responded to attacks (using ambulance and fire brigade agents). Failure in activity could lead to potentially severe costs – i.e. failing to intercept a terrorist could lead to subsequent attacks and loss of life – meaning a strictly reactive approach offered insufficient efficacy. I-Globe supported real-time replanning through *decommitment* rules associated with obligations, which defined conditions for an agent dropping an intention and adopting an alternative plan instead (Wickler *et al.* [2009]). A notion of capability was defined, giving agents knowledge of activity preconditions, with decommitment rules (similar in effect to protection monitors) linking violation of preconditions to the triggering of recovery plans.

Whilst plan repair can change plans in response to threat – whether in reaction to activity failure, or proactively where preconditions are detected as violated before execution – plan execution maintenance offers a method for *invoking such behaviour*. The PEM approaches described are concerned with *post* execution states, placing them as more of a reactive approach. They can also be considered proactive, when detecting violation of activity preconditions prior to execution. However, this risks drawbacks similar to continuous planning due to the immediacy of required activity – being able to

detect and address threats further in advance may offer greater flexibility and options, and is a motivator of our focus upon an explicitly proactive approach.

Figure 6.1: Screenshot of an I-Globe scenario (Komenda *et al.* [2009b]).

Our approach requires PEM behaviour, with a proactive focus, and defined within the context of BDI agent and multiagent reasoning – i.e. using agent mental state components, as a specific part of reasoning, and also covering joint activity. We can also adopt aspects of the surveyed approaches which naturally extend to our multiagent context and previously identified requirements. Detection of precondition violation is requisite (such as by SIPE, IPEM or through CPEF assumption monitors), although we also wish to employ qualitative reasoning to account for the qualification problem.

The concepts of *aggregation* in CPEF, or agenda formation in IPEM, may be of value in allowing categorisation and prioritization of threats in our approach. As we are considering *anticipation* rather than reaction, there is a possibility of plans containing multiple activities at risk of failure over the longer (non-immediate) time period. This

is similar to re-refinement plan repair or propagation of exceptions (Section 4.3), in that multiple threats in a subplan can represent a singular combined threat to the associated subgoal or composite activity.

6.2.3 Determinization with Replanning

Determinization approaches form a classical problem from a non-deterministic domain specification (i.e. where operators have non-deterministic effects); the determinized planning problem is solved using a classical planner – taking advantage of classical planning optimizations – with the resultant plan translated back to probabilistic operators for execution. These planners recognize that determinizations will be inexact, and compensate by replanning where the *actual* effects following execution diverge from those expected (i.e. by the determinized domain).

FF-Replan (Yoon *et al.* [2007]) is one such example; this planner won the IPPC-2004 competition, and – although not formally entered – outscored the winners of IPPC-2006 in a significant number of domains. FF-Replan maps probabilistic operators to deterministic operator specifications, before employing the *FF* planner (Hoffmann [2001]) to solve the determinized problem. The initial version of FF-Replan utilized *single-outcome* determinization, where the highest probability outcome was selected as the effects specification of an equivalent deterministic operator; latter versions adopted *all-outcomes* determinization (i.e. treating all possible probabilistic outcomes as equal), forming a deterministic operator to represent every possible outcome combination. FF-Replan monitors state during execution, replanning upon any divergence between actual and expected post-execution state.

FF-Replan was later extended by FF-Hindsight (Yoon *et al.* [2008]) to employ *hindsight optimization* (Chong *et al.* [2000]) for determinization. FF-Replan's single-outcome determination assumes the most likely outcome occurs – if a less probable outcome occurs (resulting in replanning), it may render it costly to achieve the goal. FF-Hindsight instead forms multiple determinizations to consider alternate potential outcomes, selecting activities that *with hindsight* (from the plans formed using these determinizations) provides a starting point to reach the goal cheaply. FF-Hindsight offered improved performance over FF-Replan in IPPC-04 and 06 domains, and in 'probabilistically interesting' domains suggested by Little and Thibaux [2007] in ear-

lier criticism of FF-Replan. This approach had high computational expense, scaling with the state-space size and applicable action set, although Yoon *et al.* [2010] identified further enhancements aimed at improving scalability.

PAC-PLAN (*ProbabilitiesAreCosts-Plan*) offers another example of determinization (Jiménez *et al.* [2006a]). Here, all-outcomes determinization is used to form deterministic *alias* actions (plans can be converted back from alias actions to the original probabilistic actions). Each alias action has a cost equating to *risk of failure*, given as $risk_i = -log(prob_i)$, where $prob_i$ gives the probabilistic likelihood of that operator's specific effects. The deterministic planner (in their evaluation, LPG-TD-1.0 by Gerevini and Serina [2002]) employs metric-minimization based upon this risk value, corresponding to minimizing risk of failure.

Determinization and replanning both attempt to handle uncertainty (by considering probabilistic operators) and incorporate plan execution monitoring (i.e. to trigger replanning under unexpected effects in FF-Replan). However, these approaches focus upon the possible outcomes of *successful* execution; i.e. where preconditions hold and the activity succeeds, but with effects of varying probabilities. For a determinization approach to account for the possibility of *failure*, the resultant *consequences* would need to be known and provided as determinized effects; this would require an *all-outcomes* approach, as the most probable outcome of preconditions holding (as per the qualifications problem) is success, meaning any single-outcome approach would omit failure case effects when forming determinized operators.

Representing the failure case in an all-outcomes or weighted operator (as in *PAC-Plan*) is counter-intuitive, as such operator representations effectively mean planning for *failure* – whilst failing to capture that the activity preconditions are associated with a significantly *lower* probability of failure than where they do not hold. Finally, any determinization approach attempting to handle both failure and success outcomes of activities is arguably entering the realm of conformant planning, and facing the same issue of intractability. This means these types of approaches, whilst able to handle uncertainty in the form of unexpected effects, cannot be seen as strictly concerned with failure recovery. Additionally, planning for such uncertainty may be less effective if post-execution state is strongly influenced by exogenous change.

However, PACPlan does suggest a method for weighting – through metric values – deterministic operators to account for the probability of success. This provides a viable method to incorporate quantitative quality estimations from capability meta-knowledge into our proposed approach, *if* using a capable runtime planner. Approaches such as FF-Replan evidence the need to consider divergence from assumed state when executing deterministic plans in realistic, uncertain environments; reinforcing our motivation to proactively avoid failures caused by such divergence.

6.3 Conclusion

This chapter described techniques for avoiding or reacting to failure in uncertain environments, where activities may lead to unexpected post-execution states (including from exogenous change). We first surveyed various approaches that attempt to prevent failure by accounting for uncertainty to avert unexpected circumstances (in the MDP or conformant planning case), or by generating plans offering adaptive flexibility (through conditional branching, or deferring decisions). All of these approaches, however, face issues limiting their viability in realistic environments.

MDP and conformant planning methods become intractable in such environments, particularly with the possibility of exogenous change; abstraction can improve MDP tractability, but reduces policy optimality. Contingent planning is similarly likely to become intractable in the event of complex environments, due to a combinatorial explosion in the number of branches required to account for possible exogenous change before, during, or after activity execution. Finally, continual planning can ward against uncertainty by deferring decisions – but risks inadvertent long term failure.

In general, we form assumptions that:

- Agents in a realistic environment cannot form intended plans which entirely prevent threats from unexpected execution contexts.
- Robust agents must counteract divergences between planning-time (i.e. assumed) and execution-time activity execution context.

The second section of this chapter considered *response* to failure; including performing replanning or plan repair, use of plan execution monitoring to trigger such behaviour, and the combination of both in determinization approaches such as FF-

Replan. Our approach will be required to provide similar behaviour, but within BDI agent reasoning in an explicitly proactive manner.

Plan repair is preferable to replanning in a distributed context due to reduced communications cost. We noted a relative lack of existing work specifically covering *multiagent plan repair*, with the exception of Komenda *et al.* [2012]. Existing techniques such as HTN plan repair and exception propagation may be applicable towards such behaviour, as identified through our previous requirements. This supports the value of our contribution, which should provide proactive, pre-emptive (of failure) plan modification – i.e. plan repair – in a MAS.

We can form the following requirements and assumptions from our discussion of plan robustness techniques:

- We require agents to monitor plan execution, detecting where exogenous change threatens the next activity to execute *or* it's successors in the intended plan.
- Our approach must identify and respond to both violation of preconditions and loss of quality.
- A plan repair approach is required for local (agent level) plan repair, based upon reduced communications overhead over replanning.
- We require minimal plan modifications as a *soft* goal, allowing flexibility to reduce computational cost (i.e. to mitigate the relative inefficiency stated by Nebel and Koehler [1992] when maximum retention is an explicit, hard goal), whilst attempting to provide the communications cost benefits of greater stability.
- We assume the distributed plan executed by a decompositional hierarchical team can be equated to a Hierarchical Task Network, where delegation of activities to plan forming agents is effectively equivalent to HTN task refinement, and wish to replicate HTN re-refinement plan repair through distributed robustness behaviour.

Chapter 7

Behavioural Design

In this chapter, we first detail the *Cargoworld* domain serving as our detailed motivating example. This is subsequently used to describe our behavioural design requirements – firstly the assumed MAS behaviour in non-failure circumstances, and secondly the desired response when intended plans are threatened by exogenous change.

7.1 The Cargoworld environment

Chapter 2 introduced the *Cargoworld* domain. Before specifying desired behaviour, we detail Cargoworld to provide specific motivating examples – including defining world states, possible activities and agents within the MAS.

7.1.1 Domain Predicates and Operations

We list predicates for Cargoworld in Figure 7.1, defining potential worlds states and, ergo, information potentially held in agent *beliefs*. Contradictory states are assumed mutually exclusive; e.g. *Truck1* cannot have percepts indicating it is both *healthy* and *mortal*, or be simultaneously at multiple junctions. Entities in the environment have associated percepts identifying their type; for example, *Truck*s will be identified as present through percepts *Truck*(*Truck1*), *Truck*(*Truck2*), and soforth. This allows representation of specific entities, and their type(s), within planning operators.

Predicate	**Variables**	**Meaning**
busy	*ag* – agent	*ag* is currently performing some activity
healthy	*ag* – agent	*ag* is at optimum health
damaged	*ag* – agent	*ag* has suffered debilitation but remains functional (with reduced activity quality)
mortal	*ag* – agent	*ag* is mortally damaged and unable to act
atJ	*ag* – agent *j* – junction	*ag* is located at *j*
onR	*ag* – agent *r* – road	*ag* is currently *at some point* along *r* (between *r*'s endpoints)
overJ	*ag* – agent *j* – junction	*ag* is currently flying above *j*
flying	*ag* – agent	*ag* is currently (flying) in the air
airport	*j* – junction	Indicates there is an airport at *j*
loaded	*ag* – agent *c* – cargo	*ag* is carrying *c*
carryingCargo	*ag* – agent	*ag* is carrying a cargo item
cargoNeeded	*j* – junction	Request for cargo to be delivered to *j*
cargo	*c* – name	Defines an item of cargo exists with identifier *c*
cargoAt	*c* – cargo *j* – junction	*c* is currently located at *j*
stuck	*ag* – agent	*ag* is stuck in it's current location (i.e. skidded off-road) and cannot move until it frees itself
resting	*ag* – agent	Corrolary to the *stuck* percept; *ag* cannot free itself whilst in a resting state. When an agent becomes *stuck*, it must rest for a period of time before recovery. This prevents *stuck* being a zero-consequence state which can be immediately recovered from using a *free* activity.
toxic	*j*1 – junction *j*2 – junction	The road connecting *j*1 and *j*2 has been contaminated with toxic substance(s)
toxicRd	*r* – road	Identifies a toxic road by id *r*
dangerZone	*j* – junction	*j* is dangerous and cannot be used by non-APC agents
blocked	*j*1 – junction *j*2 – junction	The road between *j*1 and *j*2 has been rendered unusable until cleared
blockedRd	*r* – road	Counterpart to *blocked*, defining the specific road identifier *r*
windy		Indicates environmental conditions are currently windy, impacting flight activities

Figure 7.1: Environmental State atoms within *Cargoworld*

Numerous activities can be performed, each corresponding to the use of an entity effector and representing a directed state change (Figure 7.2). We prepend an additional argument to indicate the performer(s) for a delegated activity within agent plans – i.e. *move*($Truck1$, A, B) indicates a *move*(A, B) is delegated to $Truck1$ – this is given for readability and does not represent an implementation requirement.

7.1.2 Failure Sources

We define three generalized *types* of activity failure which may occur in a stochastic dynamic environment, and give examples within a Cargoworld environment. These definitions do not consider *programmatic* failures due to incorrect implementation, or misspecification of plans or planning domains.

- **Preconditions failure** refers to failures stemming from preconditions not holding; i.e. where exogenous change between plan formation and execution prohibits success of an activity. For example, if *Truck1* intends to *move*(A, B), but $A \rightarrow B$ becomes blocked by a landslip.
- **Non-deterministic failure** refers to where the world state did not prohibit success (i.e. preconditions held), but introduced additional *risk* that lead to failure. For example, road $A \rightarrow B$ may become *slippery* from rainfall – although still nominally traversable, the more hazardous conditions lead to *Truck1* sliding off of the road and failing to *move*(A, B). This type reflects scenarios where failure is possible but not *certain* – reflecting the 'hidden' uncertainties within a realistic domain reduced to deterministic terms.
- **Exogenous (change) failure** is where an exogenous event *during* execution causes immediate failure – e.g. *Truck1*'s engine explodes, forcing it to stop.

Preconditions and Non-deterministic types represent *preventable* failure – we argue these risks can anticipated using (actual or predicted) execution context knowledge, combined with agent *meta-knowledge* regarding their activity execution capabilities. Exogenous failure represents cases which cannot be readily anticipated in advance; although this type of failure is an inevitable risk in continuous, stochastic environments, we argue there remains value in anticipating and preventing the former types. Our motivation assumes non-exogenous failure scenarios are sufficiently frequent to justify pre-emptive measures; our eventual design should also allow complementary reactive measures (which *can* respond to exogenous failure).

Signature	Arguments	Purpose
move	*ag* – agent *r* – road *o* – junction *j* – junction	*ag* moves along *r*, from *o* to *j*. *o* and *j* must be connected by *r*, and *ag* must be either at *o or* be located at some point on *r*. This activity is performable by any road vehicle, although specific preconditions may vary.
load	*ag* – agent *c* – cargo *j* – junction	The agent *ag* loads (picks up and holds) *c*; both *c* and *ag* must be co-located at *j*. Helicopters cannot load (or unload) cargo whilst flying.
unload	*ag* – agent *c* – cargo *j* – junction	*ag* unloads *c* and deposits it at *j*; *ag* must be carrying *c* and at *j*.
takeOff	*ag* – agent *j* – junction	Helicopter *ag*, landed at *j* – which must hold an airport – becomes airborne *over j*.
land	*ag* – agent *j* – junction	Helicopter *ag*, which must be airborne over *j* and where *j* must hold an airport, lands at *j*.
fly	*ag* – helicopter *o* – junction *j* – junction	Helicopter *ag* flies directly from *o* to *j*; *ag* must already be airborne.
secureArea	*ag* – APC *j* – junction	The APC agent *ag* removes the dangerZone at *j*; *ag* must already be present at *j*.
unblock	*ag* – bulldozer *r* – road *o* – junction *j* – junction	*ag* clears the blocked *r* (which must connect *o* and *j*), moving from *o* to *j*. *r* must not be flooded or toxic, and *o* and *j* must not have associated dangerZones.
decontaminate	*ag* – Hazmat *r* – road *o* – junction *j* – junction	*ag* moves from *o* to *j*, decontaminating (the connecting road) *r* in the process. *r* must not be flooded and blocked, and neither *o* or *j* can have dangerZones. *ag* must also be initially located at *o*.
consume	*ag* – agent *j* – junction *c* – cargo	*ag* informs *j* it can consume *c* in order to satisfy an existing request.
free	*ag* – agent *r* – road *o* – junction *j* – junction	*ag* frees itself from being stuck on *r*, connecting *o* and *j*. *ag* must not be resting, and *r* itself must be traversable by that agent. The preconditions for *free* allow definition of planning goals to establish required conditions and free agents from being stuck.

Figure 7.2: Possible operations within *Cargoworld*

7.2 Agents within a Cargoworld MAS

We define two classes of agent. *Physical* agents correspond to, and control, entities within the environment which can directly influence environment state. *Logical* agents do not have this association, and achieve goals through dependencies upon other agents (which directly or indirectly lead to activities by physical agents). In our Cargoworld example, each world entity (Section 2.4.2) has an associated physical agent representing it within the MAS.

We first describe *logical* agents and their potential interactions with others:

- The **LogisticsHQ** provides 'top level' (strategic) control; receiving cargo requests, identifying cargo, and then selecting a capable agent to deliver that cargo. Physical agents do not have peer-to-peer visibility; consequently, the *LogisticsHQ* also serves as a broker to 'expose' functionality to others. For example, a *Truck* wishing to clear a road will form an *unblock* dependency upon the *LogisticsHQ* – which, in turn, selects and dispatches an appropriate *Bulldozer*.
- **MilitaryHQ** acts as a tactical level controller, serving as a broker for use of *APC* and *Hazmat* agent types by *LogisticsHQ* and vice-versa. In the former case, *MilitaryHQ* is responsible for forming dependencies upon *APC* or *Hazmat* agents (as appropriate) to secure a given junction or decontaminate a given road, including performing of a specific physical agent. In the latter, *MilitaryHQ* acts as a proxy; *APC* or *Hazmat* agents form dependencies upon *MilitaryHQ*, which forms an equivalent dependency upon *LogisticsHQ*. For example, if an *APC* needs to clear a road for travel the resultant dependency chain would be *APC* $\rightarrow$ *MilitaryHQ* $\rightarrow$ *LogisticsHQ* $\rightarrow$ *Bulldozer* [1].

Our design will not assume any authority structure; we are concerned with the team meta-organizations arising from dependency relationships, rather than constraints upon their formation. However, realistic systems frequently utilize organizational hierarchies such as Strategic-Tactical-Operational layers (described in Killion [2000]), as these represent a proven method for decomposing and organizing solutions to complex problems. We represent this in our example MAS by visibility constraints upon capabilities, restricting the possible dependency relationships an agent can form (Fig-

[1] We refer to cases such as this as *indirect* dependencies, i.e. *APC* indirectly depends on *Bulldozer*.

ure 7.3[2]). For example, *Truck1* cannot delegate to *Bulldozer1*, and must use *LogisticsHQ* to unblock a road. *Truck1* would not know the identity (or type) of any further agent(s) used – this semantic knowledge restricted to *LogisticsHQ*.

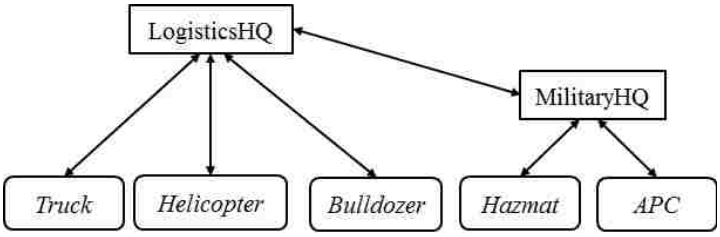


Figure 7.3: Example multiagent hierarchy; arrows indicate an agent can form dependencies upon another.

The heterogenity in *Cargoworld* is simple enough that a fully peer-to-peer agent system (with no capability visibility constraints) could be employed. We specify an effective hierarchy to serve several purposes. Separation of knowledge (of both delegation structures and semantic details) and responsibility reflects both real world command-chain concepts and agent-knowledge specialisation that exists and which often motivates a multiagent approach (Sycara [1998]). The relationships between agents are also more complex due to restriction of semantic knowledge to specific agents; this mandates dependency formation, mirroring the likely necessity of dependency formation in real world scenarios.

7.3 CAMP-BDI Behaviour

This section describes generalized behaviour examples of a CAMP-BDI MAS within a *Cargoworld* environment, extending from fault free execution to the desired proactive behaviour for potential failure cases.

7.3.1 Normal Agent Behaviour

Before considering maintenance behaviour, we describe multiagent activity covering successful delivery of cargo without requiring failure mitigation. We reference the geography previously defined in figure 2.6, where a cargo request has been generated

[2]Where required, we refer to specific individuals as numbered instances; i.e. *Truck1* and *Truck2* are *Truck* type agents. The numerical designation is dropped where we only need to refer to a single instance of that type, or that type in general.

at E, and a cargo object ($cargo1$) exists at K. *Truck1* and *Truck2* are at A and M, respectively.

We now discuss typical multiagent activity, starting with adoption of a *deliverCargo*(E) goal by *LogisticsHQ*. This requires dependency formation between agents, allowing task delegation. We use the terminology *dependant* to refer to the agent delegating a task, and *obligant* to refer to the agents *performing* it.

The following sequence summarizes our assumed behavioural model for a MAS performing this delivery task in *Cargoworld*;

1 *LogisticsHQ* forms an intention to perform a plan achieving *deliverCargo*(E); this goal is satisfied by removal of the state *cargoNeeded*(E).
Truck1 and *Cargo1* are selected as task-performing obligant and the utilized cargo resource (we define intentions in the form **goal**:*plan*) :
deliverCargo(E): *moveTo*(*Truck1*, A, K), *load*(*Truck1*, *Cargo1*, K), *moveTo*(*Truck1*, K, E), *unload*(*Truck1*, *Cargo1*, E), *consume*(*Cargo1*, E)

2 A dependency is formed upon *Truck1* to perform *moveTo*(*Truck1*, A, K). *Truck1* forms a route plan to reach K from it's current location, to be executed upon the dependant's request;
moveTo(*Truck1*, A, K):*move*(A, G), *move*(G, J), *move*(J, K)

3 Acceptance of further dependency requests from *LogisticsHQ* results in the following *goal*:*plan* pairs being formed by *Truck1*, executed (adopted as intentions) upon the dependant's request;
load(*Truck1*, *Cargo1*, K):*load*(*Cargo1*, K)
moveTo(*Truck1*, K, E):*move*(K, L), *move*(L, I), *move*(I, E)
unload(*Truck1*, *Cargo1*, E):*unload*(*Cargo1*, E)

4 *LogisticsHQ* executes its plan for *deliverCargo*(E), requesting *Truck1* perform each delegated activity in turn. *LogisticsHQ* waits for each dependency (delegated activity) to complete; once *Truck1* confirms successful execution, *LogisticsHQ* progresses it's (dependant) intended plan onto the next activity.

5 *LogisticsHQ* completes by using *consume*(E, $Cargo1$), which uses $cargo1$ to satisfy the request from E.

The following series of atomic activities are consequently performed:

1 *move*(*Truck1*, *A*, *G*)
2 *move*(*Truck1*, *G*, *J*)
3 *move*(*Truck1*, *J*, *K*)
4 *load*(*Truck1*, *Cargo1*, *K*)
5 *move*(*Truck1*, *K*, *L*)
6 *move*(*Truck1*, *L*, *I*)
7 *move*(*Truck1*, *I*, *E*)
8 *unload*(*Truck1*, *Cargo1*, *E*)
9 *consume*(*LogisticsHQ*, *Cargo1*, *E*)

We assume multiagent activity requires advance formation of dependency contracts. This is not an inherent requirement of the BDI approach, but we argue a logical requirement for distributed activity – to protect against agent or resource contention and facilitate information sharing (such as when establishing mutual beliefs). Our approach – discussed in the following chapters – extends this assumption to communicate maintenance-relevant information.

This section described the (assumed) MAS behaviour under normal conditions; we will next describe the desired behaviour under potential *failure* scenarios.

7.3.2 Behaviour to prevent Preconditions Failure

Truck1 is currently on road $A \rightarrow G$ (Figure 2.6), travelling route $A \rightarrow G \rightarrow J \rightarrow K$ to load *cargo1* at K. $J \rightarrow K$ becomes blocked by a landslip, violating preconditions for travel along $J \rightarrow K$. Upon arriving at G (and beginning its next reasoning cycle) *Truck1* should detect likely failure of the planned future *move*(*Truck1*, *J*, *K*) and identify this as being due to a violated precondition.

Truck1 should consequently modify the intended plan. For example, *Truck1* can form a dependency upon *LogisticsHQ* to unblock $J \rightarrow K$; *LogisticsHQ* subsequently selects and delegates this task to *Bulldozer1*. Execution restores precondition states for using $J \rightarrow K$, before *Truck1* reaches J, avoiding this anticipated failure (Figure 7.4).

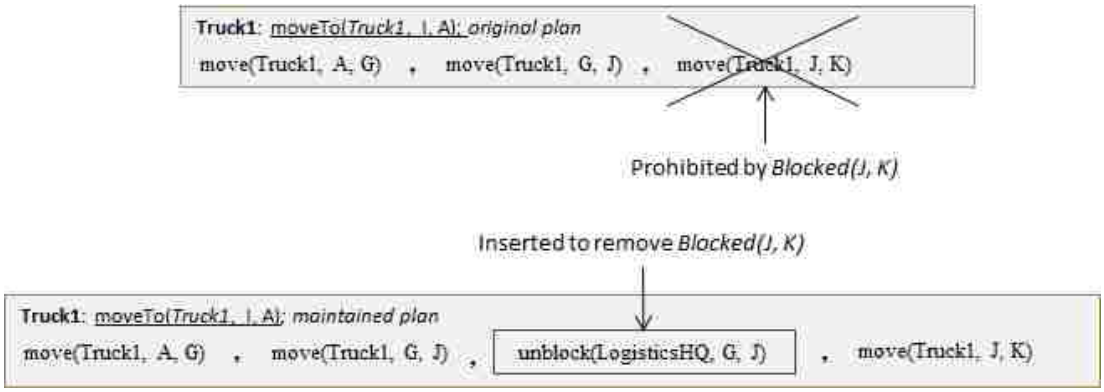


Figure 7.4: Plan modification to prevent preconditions failure; *unblock* is inserted to re-enable the planned *move*, after an exogenous event blocks $J \rightarrow K$.

7.3.3 Behaviour to prevent Non-deterministic Failure

In these scenarios, changes (compared to original assumptions at intention formation / plan selection) in the anticipated execution context of an activity render it sub-optimal. For example, *Truck1* is carrying cargo from K to E, along route $K \rightarrow L \rightarrow I \rightarrow E$, when a rainstorm results in $I \rightarrow E$ becoming *slippery*. This does not prohibit travel down that road, but does increase the risk of failure when doing so; instead, *Truck1* forms an alternate path $K \rightarrow H \rightarrow C \rightarrow D \rightarrow E$, using only dry roads (Figure 7.5).

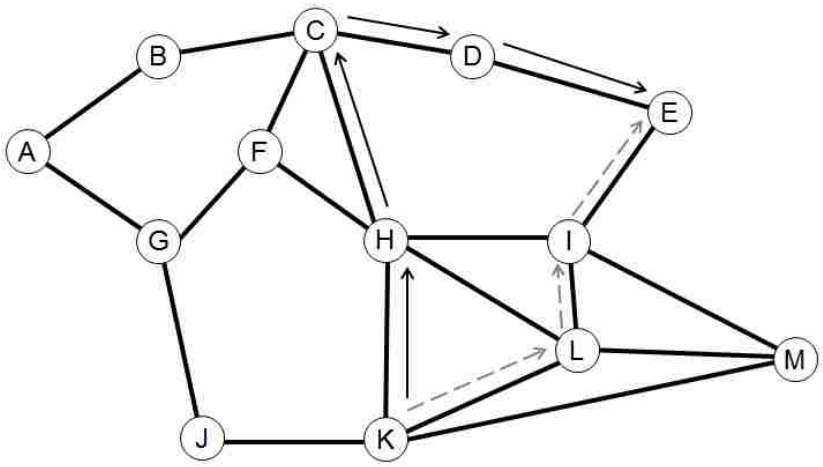


Figure 7.5: Modified route adopted by *Truck1* (solid arrows), located at K, to pre-emptively avoid slippery $I \rightarrow E$. The original planned route is shown in dashed arrows.

This does incur additional cost through extra movement – maintenance must ideally balance *extent* of risk against the costs of changing plans (both in terms of activities and

computational resource). Avoiding failure *earlier* may also have some efficiency benefits over recovery; in this example, *Truck1* would incur additional costs backtracking to H if it delayed maintenance until set to execute *move*(*Truck1*, I, E).

We can extend this behaviour to scenarios where an agent is unable to prevent preconditions failure; if an agent cannot re-establish preconditions for an activity, it should instead replace that activity with one (or more) whose preconditions *can* be established. In certain cases it may even be preferable to replace an activity *regardless* rather than insert prior precondition-establishing activities – the latter can lead to continual and incremental growth of plan length and complexity, purely to enable a single activity which may actually only hold limited significance towards the intended goal.

7.3.4 Distributed Maintenance Behaviour

We also consider behaviour of an agent *team*, performing a distributed plan, where an individual member is unable to perform adequate maintenance. One such example arises where *Truck1* is carrying cargo along the route $K \rightarrow L \rightarrow I \rightarrow E$, but suffers partial damage en-route to K – increasing failure risk for the future *move*(*Truck1*, L, I).

Truck1 cannot recover independently by repairing itself. At this point, the dependant *LogisticsHQ* should be made aware *Truck1* is at risk of failure for its obligation to *moveTo*(*Truck1*, K, E) – and that *Truck1* has attempted and failed to mitigate that risk. This requires *LogisticsHQ* to reconsider the (dependant) intended plan.

The *moveTo*(*Truck1*, K, E) activity cannot be directly substituted due to an inherent reliance upon the debilitated *Truck1* for that goal. This also threatens *unload*(*Truck1*, *cargo1*, E), whose preconditions require *Truck1* to have moved to E. *LogisticsHQ* is required to adapt its intended plan with consideration of the associated goal, aborting *moveTo*(*Truck1*, K, E) and subsequent activities before forming a new plan and dependencies such that:

1 *Truck2* moves from M to K to rendezvous with *Truck1*:
moveTo(M, K)
2 *Truck1* unloads *Cargo1* at K:
unload(*Truck1*, *Cargo1*, K)

3 *Truck2* loads *Cargo1* at K:
$load(Truck2, Cargo1, K)$
4 *Truck2* travels to the delivery destination:
$moveTo(Truck2, K, E)$
5 *Truck2* unloads cargo at E:
$unload(Truck2, Cargo1, E)$
6 The cargo is released for use (delivered) at E:
$consume(Cargo1, E)$

This places additional requirements beyond the localized, individual agent maintenance case. Obligants must communicate information allowing dependants to identify where delegated activities are threatened, and adopt responsibility if obligant maintenance was unable to address that threat. This requires synchronization to ensure individual maintenance is first attempted at lower levels of a dependency hierarchy, to minimize disruption to a distributed plan. It is desirable to avoid centralized control(s), as information and computation requirements in large, realistic systems typically render these infeasible.

7.4 Summary

This chapter described the *Cargoworld* domain introduced in Chapter 2, to detail domain predicates, entities and activities. We also described an example MAS for operating within this domain. *Cargoworld* uses a transport paradigm to provide a comprehensible motivator for agent behaviour which can be related to real-world practical applicability. Agent heterogeneity provides increased options for failure mitigation and planning (making such decisions non-trivial), and allows more detailed consideration of multiagent plan execution.

We used *Cargoworld* to first describe assumptions about MAS behaviour, and then desired maintenance behaviour; the latter covering both preconditions violation and loss of planned activity quality (expressed as an increase in the risk of failure, as is relevant to our robustness concern) due to exogenous change after a plan is intended. This provides a generalized, abstracted view of the maintenance process, and defines a number of requirements for out subsequent design:

- Agents must *perform introspective reasoning* to identify threats to planned ac-

tivity, identify appropriate plan modifications, and ultimately avoid anticipated failure.

- A method is required for *communication of necessary information* to support introspective reasoning, building upon dependency contract formation.
- Due to realistic scenarios requiring distribution of knowledge and ability across various agents, a *decentralized* approach to distributed maintenance is necessary – including communication of information between the obligant(s) and dependant to support adoption of responsibility by the latter.
- Proactive maintenance may carry greater computational cost than reactive behaviour, due to its anticipatory nature. The frequency of maintenance should be able to be balanced against the consequences of failure for particular activities, particularly in non-deterministic failure cases.
- Finally, we can intuit it is desirable to support flexible modification of maintenance behaviour during runtime, to adapt to observations of maintenance efficacy.

Chapter 8

CAMP-BDI Supporting Architecture

This chapter contributes our *supporting architecture*; special case *Beliefs* used to support introspection and later plan modification by our algorithms, and which also provide representation models for communication during distributed activity. We use this architecture to support our subsequently designed behaviour for *CAMP-BDI* agents – BDI agents which are *Capability Aware*, and which use that capability knowledge to *Maintain Plans*.

8.1 Mental State Components within the BDI agent Model

We first define the standard mental state components of Beliefs, Desires and Intentions. **Beliefs** represent assumed knowledge of an agent regarding itself and the environment, and can be defined as a set of positive and negative state atoms;

$$B = B^{+} \cup B^{-}$$

The CAMP-BDI supporting architecture provides a subset of B used to support our algorithms. In this thesis we refer to these components independently due to their specific purposes within our approach.

Desires are a set of potentially conflicting goals, individually valid given current beliefs, from which the agent selects intended goals.

$$\forall d \in D\text{: } d\text{=}\{\ g_1, \ldots, g_n\ \}$$

Each individual d represents states to be achieved or removed, defined as being either explicitly positive, explicitly negative, or implicitly negative[1]. We refer to a desire as as to achieve a goal g=$g^+ \cup g^-$; g^+ defines literals required to be present in B to achieve g, and g^- those required absent. For example, a goal for *Truck1* to move from A to B, and not be mortally damaged at the end, gives $g^+ = \{atJ(Truck1,B)\}$ and $g^- = \{atJ(Truck1, A), mortal(Truck1)\}$. The individual desire is met when $g^+ \subseteq B$ and $g^- \not\subseteq B$.

Desires are active *achieve* goals, as defined by Braubach *et al.* [2005]. An agent may hold a *goal* set covering multiple types (including *maintain* goals described by Dastani *et al.* [2011]), used by standard BDI reasoning. If *reactive* maintenance goals exist, we regard them as stimulating desires to (re)achieve protected propositional states when necessary. *Proactive* maintenance goals (Duff *et al.* [2006]) prohibit certain states being established – responsibility for respecting them will lie within the plan identification mechanism used for intention formation and, in our design, later co-opted for maintenance planning.

The **Intentions** of an agent are defined by a selected, non-conflicting, subset of Desires - conceptually, $I \subseteq D$. The exact definition of an intention varies within the literature; this thesis adopts the definition employed by Simari and Parsons [2006], where an intention combines *both* goal and plan;

$$\forall i \in I : i = \{goal_i, plan_i\}$$

The $goal_i$ corresponds to a selected desire; $plan_i$ represents the plan (to be used) to achieve $goal_i$. This explicit association allows agents to reconsider how they achieve goals, when those goals remain valid despite threats to a specific *means*. The Normative Agent Architecture (NoA) (Kollingbaum and Norman [2003]) and the B-DOING architecture (Dignum *et al.* [2002]) similarly distinguish selected goals within agent mental state, albeit focusing upon the various influences upon rational reasoning.

We define a plan p as a linear sequence of n activities;

$$p = \{a_1, \ldots, a_n\}$$

[1]We make a closed world assumption where the absence of a positive atom within B can be treated as that atom being *implicitly* negative. For example, a vehicle agent does not require the explicit belief $\neg at(A) \in B$ if $at(A) \notin B$.

Activity a represents a state transition $F(a,s)=s'$; s' is the outcome of successfully executing a in state s. An activity can represent either an atomic action – i.e. use of an agent effector – or a goal performed through decomposition to an executable sub-plan (i.e. as in hierarchical task networks). A p as a *primitive plan* if every $a \in p$ is atomic. Where $a \in plan_i$, we refer to $goal_i$ as the *parent goal* of a.

We assume all activities can be captured by deterministic STRIPS planning operators (Fikes and Nilsson [1971]). However, we do *not* assume all factors influencing the success of a are represented through deterministic preconditions, but rather the most significant *prohibitory* ones (McCarthy [1958]).

Finally, we refer to the set of obligations held by an agent (to perform some activity upon request) as Ob, and the dependencies as Dp. These are more fully detailed in section 8.4, but also referenced in preceding sections as information sources.

8.2 Capabilities

Our approach is founded upon *pre-emptive* behaviour – that agents respond to exogenous change by identifying whether intended plans are negatively impacted, and modifying those plans – if necessary – to compensate. This requires introspection by agents regarding their ability to perform activities. The *capability* model provides meta-knowledge to both assess viability of an activity in a given world state, and to modify plans in recompense. A common model is employed for capabilities (discussed subsequently), encapsulating semantic knowledge requirements within the implementation – this aids generality, communicability and re-usability.

8.2.1 Existing Approaches towards Capability Modelling

Self-awareness is an important aspect in design and implementation of an intelligent agent; including representation of agent ability, and its impact upon which goals can be achieved – Xuan [2006] suggests rational agents must be able to reason over activity utilities. Morgenstern [1986] describes planning and acting agents as requiring knowledge of their possible activities and achievable goals; an agent has *know-how-to-perform* an activity if it is aware of the constraints allowing or prohibiting execution, *can-perform* an activity if it can be executed in the current situation, and has *know-how-*

to-achieve for a goal where it *can-perform* an activity achieving the required effects. These concepts are extended into planning in terms of *can-execute* a plan, and where an agent *can-plan* to achieve a goal (by executing the activities itself and/or delegation).

Singh [1999] defines similar concepts of *know-how* as determining what goals can be achieved;

> *An intention can lead to success when it is held long enough, is acted upon, and when the agent has the requisite know how.*

Two types of action describe the *know-how* of an agent. *Basic* actions are primitive, atomic activities. *High level* actions represent procedural knowledge – sequences of lower-level actions that can be performed, where constituent actions may be high level (i.e. subgoals) or basic. High level actions must eventually resolve to a set of basic actions; i.e. plans must eventually resolve to some set of effectors for an agent to be deemed capable.

One predominant focus of existing work lies upon modular, reusable capabilities, to allow composition of agents from encapsulated capability objects. Busetta *et al.* [2000] define resuable capability models serving as 'building blocks' for creating agents; each capability defines a subset of relevant beliefs, plans and handled triggering events for the plans. These may be composed of further sub-capabilities; agents are viewed as defined by graph-like capability structures rather than sets of plans and beliefs. Nunes [2014] suggests three types of relationships between such modular capabilities; association (directional or bidirectional dependencies between capabilities), composition (knowledge is *shared*, such that a capability is aware of beliefs and/or achievable goals within another), and inheritance (a capability reusing and extending anothers constituent components).

Braubach *et al.* [2006] extend the modular model of Busetta *et al.* [2000] to address a number of perceived issues. They use inclusion of initial beliefs to parameterize capability instances, and suggest an approach for dynamic modification of agent capability sets. Capability *loss* is treated as unpreventable and irreversible – limiting response to deciding whether to abort intended plans or reactively handle resultant failure. Similarly, Padgham and Lambrix [2005] offer a model focused upon modular reuse, which limits failure handling to predefined recovery plans or dropping intentions entirely.

Padgham and Lambrix [2005] extend the concept of accessible worlds (Rao and Georgeff [1995]) to include *capability-accessible* worlds – where desires and intentions are restricted to the goals an agent is *capable* of achieving. A somewhat analogous approach (if we view capabilities as representing plan information) is suggested by Waters *et al.* [2014]; although they do not explicitly define a capability concept, they suggest an intention selection mechanism where agents favour execution of intended plans with least *coverage*[2]. Agents maximize overall intention throughput, by selecting the lowest coverage intention in each reasoning cycle – recognizing such intentions are less likely to be executable in future circumstances.

8.2.2 Capability Model

A capability c, for activity a, is defined with the following fields;

$$c = < ag, s, g(a), pre(a), \mathit{eff}(a), conf(a, B_a) >$$

- ag: identifies the agent that performs a – this may be different from the agent holding the capability object.
- s: a signature with name n and t parameters ($s = n(v_1, \ldots, v_t)$); the combination of s X ag can be used to uniquely identify a given c. For example, an activity to *move* along road $A \rightarrow B$ would correspond to s=*move*(*?from*, *?to*), where *?from* and *?to* denote variable names ground to become A and B.
- $g(a)$: the goal achieved by (succesfully) performing a, whose terms can be ground by s, defined as the sets of states which must be added ($g^+(a)$) and removed ($g^-(a)$)for $g(a)$ to be achieved. Goal states are used to disambiguate the defined purpose of an activity; i.e. performing *move*(A, B) always adds state *at*(B), but with varying *side*-effects depending on whether ag is a road or aerial vehicle. We assume multiple capabilities with the same s share a common $g(a)$; meaning s can be used to determine the *purpose* of an activity.
- $pre(a)$: a set of preconditions (belief atoms), ground to a, that define the conditions under which a can be achieved – use of c to perform a is not *guaranteed to fail* iff $pre(a) \in B_a$ (where B_a provides the – believed – execution context of a).
- $\mathit{eff}(a)$: the complete set of post-effects of successfully using c to perform a, ground using s. This can be considered equivalent to the combined set of add

[2]Thangarajah *et al.* [2012] defines the concept of coverage as representing the breadth of situations in which a plan can be executed; a plan with *high* coverage has less constraints (i.e. 'covers' more scenarios) than one with low coverage.

and delete effects of a STRIPS operator – we refer to states added as $eff^{+}(a)$ and $eff^{-}(a)$ respectively. As $g(a) \subseteq eff(a)$,the *side-effects*[3] of c are $eff(a) \setminus g(a)$.

- $conf(a, B_a)$: a X $B_a \rightarrow$ [0:1]; a "*confidence*" function estimating the scalar quality for performing a through use of c, in execution context B_a. This allows reasoning whether exogenous change has decreased optimality of a, but without violating preconditions – E.g. confidence for $move(A, B)$ (where $A \rightarrow B$ is a road) is less where $B_a \ni slippery(A, B)$ than where $B_a \ni dry(A, B)$.

A capability can be generalized as stating that, for a executed in B_a, ag can achieve $g(a)$ with some level of quality – indicating likelihood of success – estimated by $conf(a, B_a)$, provided $pre(a)$ holds in B_a, with post-effects as defined by $eff(a)$

8.2.3 Typology

We define the *type* of a capability using two overlapping categories – locality and complexity.

8.2.3.1 Locality

Capabilities are *internal* or *external* depending upon whether the capability represents knowledge of *ag*'s ability to perform an activity *itself*, or of some *other* agent that can perform it.

8.2.3.1.1 Internal Capabilities A capability is *internal* where *ag* is the same agent as that holding the capability object; i.e. the agent can alter world state itself. For example, a *Truck* agent may have internal capabilities concerned with travelling down a road or loading cargo.

8.2.3.1.2 External Capabilities *External* capabilities represent where *ag* is *another* agent – i.e. providing meta-knowledge regarding achievement of $g(a)$ through delegation to *ag*. The need to utilise other agents to achieve goals frequently motivates adoption of an MAS approach, and entails dependencies between agents. We assume agents *advertise* their internal capabilities appropriately, to allow their use by other agents[4].

[3] We assume activities are selected based upon their associated $g(a)$ states.

[4] We do not assume any specific mechanism for transmitting or the contents of advertisements, beyond that they minimally will contain the information of the external capability representation.

An external capability object represents a capability advertised by another. Using the same representational fields as internal capabilities allows maintenance reasoning to regard these externally performed activities in the same manner as internally performed. We assume advertised information is understandable by recipients; i.e. that preconditions and effects refer to states which can be sensed by the recipient. We argue it is intuitive that agents would only use advertised capabilities where they could understand such information – as this is required to identify what a delegated activity would achieve, and under which conditions.

However, some additional constraints do exist upon use of external capabilities, as the more detailed semantics of performing that activity may only be locally known. Although our approach does not mandate a specific planning approach, including for distributed planning, external capability preconditions and effects may exclude *private* state atoms (Brafman and Domshlak [2008]) – i.e. limiting communicated information to the states required to be provided by, or which can be provided to, other agents (assuming that goal-required atoms are themselves always public). *Confidence estimation* will also likely require more specific and detailed semantic knowledge than might be readily communicated (Section 8.2.5). External capabilities consequently only represent *general* case ability of the advertiser for a given activity; although more specific information *can* be provided using *contracts* (discussed in Section 8.4).

8.2.3.2 Activity Complexity

We define *primitive* capability knowledge as equivalent to know-how of *basic* actions, and *composite* to *higher level* know how (as defined by Singh [1999]). An example of a primitive capability is to *move* down some road; an example composite being knowledge to *travel* along some particular route (potentially entailing multiple *move* activities).

Our capability model is *polymorphic*; the semantics of how an activity is performed (or goals met) are encapsulated within the capability object to allow generalized maintenance reasoning. However, it is still necessary to provide some semantic indicators for certain reasoning cases – e.g to prioritize primitive internal capabilities over composite or external ones, to minimize complexity of agent activity and interdependency.

8.2.3.2.1 Primitive Capabilities

A capability is primitive if *ag* achieves $g(a)$ through a single atomic activity. All plans executed by an agent will eventually entail use of primitive capabilities, in order to actually change the world. Agent plans – both on the individual and distributed level – result in an acyclic graph, terminating in leaf nodes corresponding to either primitive activities (a change to the world directly performed by the agent) or use of an external capability (i.e. a dependency upon some other obligant). We do not require the dependant know what type of capability the obligant actually uses to perform a delegated activity (obligation) – i.e. the activity is indivisible from the perspective of the *dependant*, reducing the semantic knowledge to be communicated between agents.

This views plans as having HTN-like decompositional structures, potentially across multiple delegation relationships; the *SharedPlans* (Grosz *et al.* [1999]) and *Planned Team Activities* (Kinny *et al.* [1992]) models for distributed planning and execution both adopt similar viewd of distributed plans as acyclic graphs based upon agent decomposition. The TÆMS (Task Analysis, Environment Modelling, Simulation) modelling language (used in Generalized Partial Global Planning by Decker and Lesser [1993]) also holds a tree-structured view of distributed plans, noted by Lesser *et al.* [2004] as similar to the HTN model.

8.2.3.2.2 Composite Capabilities

BDI agents employ plans to achieve their selected desires, typically through use of a plan library. *Composite* capabilities represent the plan options *ag* has for achieving $g(a)$. Plans represented in a library are assumed to have associated selection preconditions and defined (achieved) goals, used to guide plan selection when forming intentions .

Each plan in an agent's plan library will be associated with *exactly* one composite capability. Each composite capability is associated with *at least* one plan – meaning a composite represents the set of options for $g(a)$[5]. Composite capability preconditions are the disjunction of selection preconditions for all represented plans – defining conditions where *at least one* known and selectable plan exists. The effects field is restricted to represent the goal state to be achieved – exact side-effects will only be known when a *specific* plan is selected for execution. Once a specific plan *is* selected, maintenance

[5] i.e. *n*:*1* multiplicity for *plan*:*capability* mapping, where $n>0$.

logic can identify exact effects through the capability (effects) knowledge associated with that activity sequence.

This description of composite capabilities is based upon use of predefined plan libraries, as is common in many existing BDI frameworks. Composite capabilities can also represent the ability to *form* a plan, but require domain-specific implementation of preconditions and confidence functions to represent constraints upon plan formation. Finally, we regard composite capabilities as mutable in response to addition, deletion or modification of (relevant) plans within the plan library.

8.2.3.3 Denoting Agent Type by held Capabilities

We refer to *agents* as one of two types based upon their capability set. A *primitive* agent has at least one internal primitive capability; *logical* agents can *only* achieve world state changes through delegation (using external capabilities). This distinction indicates whether an agent corresponds to (controls) some entity situated in the operating environment, or is purely organizational.

8.2.4 Matching capabilities to activities

In order to perform reasoning regarding an activity, CAMP-BDI agents must identify the appropriate capability; this section describes that process, performed by the *getCapability* function. The capabilities held by an agent (C) can be subdivided into primitive, composite and external type sets; i.e. $C = C_{internal} \cup C_{external}$ ($C_{internal} = C_{primitive} \cup C_{composite}$). The capability mapped to an activity is that which offers the most specific and relevant information about performing that activity (Algorithm 2). If the activity is delegated, and ambiguity exists over the obligant(s), then the external capabilities associated with the most likely obligant(s) are returned.

Where an activity is delegated and has an existing dependency contract, the external capability is retrieved directly from that contract (section 8.4); otherwise, the 'best fit' capability is determined. Capabilities are first matched by locality, with agents assumed to favour internal capabilities to avoid communications cost and uncertainty over the success of dependency formation. The algorithm first attempts to identify a matching capability from $C_{primitive}$ or $C_{composite}$ – an activity cannot be associated with *both* types of capability.

Algorithm 2: The *getCapability* function; we assume agents cannot intend to perform an a they do not possess capability knowledge for.

Data: a – an activity, which may correspond to achievement of a goal
B_a – the estimated execution context for a
Result: The capability for performing a, or **null** if the agent is incapable
$C_{primitive} \leftarrow$ set of internal, primitive capabilities held by the agent;
$C_{composite} \leftarrow$ set of internal, composite capabilities held by the agent;
$C_{external} \leftarrow$ set of external capabilities held by the agent;
if $a \in Dependencies$ **then**
    // Assumes dependency contracts are only formed if the agent has no local capability for a
    $contract_a \leftarrow contract$ for a in $Dependencies$;
    return $contract_a$.externalCapability;
else if $(a \notin Dependencies)$ & $(a \in C_{internal})$ **then**
    // Matches an internal capability
    return $c \in C_{internal}$ where $c.s = a$ and $c.ag =$this agent;
else if $(a \notin Dependencies)$ & $(a \in C_{external})$ **then**
    // External capability is required
    if *a has a defined obligant* **then**
        // a defines a specific intended obligant - referred to as ag_{obl} - so can match to a *specific* capability
        return $c \in C_{external}$ where $c.s = a$ and$c.ag = ag_{obl}$;
    else
        // Otherwise, assumes the best possible obligant would be employed to execute a
        return getBestExternalCapability(a, B_a);

If an agent does not hold internal capabilities for a, it must delegate that task. If an obligant has been previously identified (but a dependency contract not yet formed), the obligant name allows identification of a corresponding external capability. If *no* obligant is specified, *getBestExternalCapability* (Algorithm 3) iterates through all external capabilities with an s matching a, returning that offering the best estimated confidence (matching assumed obligant selection criteria). If a is a joint obligation – requires use of multiple agents – a new external capability is formed, representing the combination of most capable obligants. If no capability can be found, or insufficient obligants exist, an 'artificial' capability is generated which indicates that activity cannot be performed (i.e. with preconditions as *false*, *conf* as 0, and effects as $\emptyset$). This aims to reflect the constraints and confidence associated with forming a *team* to execute that delegated task.

The *mergeCapabilities* (Algorithm 4) function merges a set of multiple capabilities(C) – into a single 'synthesized' capability; $mergeCapabilities(C) \rightarrow c_{merged}$. The s and $g(a)$ parts of c_{merged} can be taken from (will be common across) any member of C. Preconditions are formed as the intersection of all preconditions from members of C, and effects as the union of all effects – i.e. all obligants must have their individual preconditions met to perform a joint activity, and all obligant effects are expected to be achieved upon successful completion.

The *conf* function represents a more complicated case, as semantics of the individual capabilities in C may vary. We treat the merged capability as holding a *reference* to the individual joint obligant capabilities; when a call is made to $c_{merged}.conf$, sub-calls are made to the *conf* functions of each $c \in C$. The resultant individual values can be aggregated to return a single value; the exact aggregation function will match that of *plan* confidence estimation (Section 8.2.6) – effectively treating a joint obligation as a 'plan' where the constituent activities execute in parallel.

Algorithm 3: The *getBestExternalCapability* function

Data: a – an activity or goal

B_a – an estimated execution context for a

Result: The highest confidence external *capability* for a, or **null** if none found

$n \leftarrow$ number of obligants required for a;

$C_{external} \leftarrow$ set of all external capabilities;

/* Let C_a be a list of capabilities, ordered by ascending confidence */

$C_a \leftarrow \emptyset$;

for *each* $c_{external} \in C_{external}$ **do**

  $conf \leftarrow c_{external}$.confidence($a$, B_a);

  Add $c_{external}$ into C_a, in confidence order using $conf$;

if *Size of* $C_a < n \parallel \emptyset$ **then**

  // Create capability indicating a is impossible

  $c_{impossible}$ = new Capability();

  $c_{impossible}.s = a$;

  $c_{impossible}.ag = \emptyset$;

  $c_{impossible}.pre = false$;

  $c_{impossible}.eff = \emptyset$;

  $c_{impossible}.conf = 0$;

  return $c_{impossible}$;

else if $n = 1$ **then**

  return top entry in C_a;

else

  $C_{topSet} \leftarrow$ top n entries in C_a;

  return *mergeCapabilities*(C_{topSet});

Algorithm 4: The *mergeCapabilities* function

Data: C – a set of capabilities, ordered by generalized confidence

Result: A capability formed by merging all members of C

$c_{merged} \leftarrow$ new (blank) Capability;

$c_{merged}.s \leftarrow\ c.s$ for any $c \in\ C$;

$c_{merged}.g(a) \leftarrow\ c.g(a)$ for any $c \in\ C$;

for *each* $c \in c_{merged}$ **do**

$c_{merged}.pre(a) \leftarrow c_{merged}.pre(a) \cap c.pre(a)$;

$c_{merged}.eff(a) \leftarrow c_{merged}.eff(a) \cup c.eff(a)$;

$c_{merged}.conf(a, B_a) \leftarrow$ function referencing all $c.conf(a, B_a)$ for all $c \in\ C$;

return c_{merged};

8.2.5 Confidence estimation

The qualification problem (McCarthy [1958]) argues that, to avoid over-constraining an operator to the extent of being unusable, deterministic preconditions can only represent the *most significant* states constraining an activity. Some states may not have significant enough effect to justify representation as preconditions, yet still influence the outcome. Confidence estimation provides a scalar representation of activity quality (in the range 0 to 1), intended to account for where the execution state decreases the likelihood of success, without being significant enough to represent as preconditions. This is somewhat similar to the use of qualitative scoring for optimal scheduling by He and Ioerger [2003], or quality scoring functions within OMACs (DeLoach [2009]).

The confidence function considers all states *known* to impact execution, including those of lower significance. This information is encapsulated within the function, abstracting away any need for the maintenance algorithm to consider precise semantics (knowledge is local to the *capability*, rather than the agent). An intuitive implementation is probabilistic estimation, but such granularity is not necessary; the function simply has to provide a comparable, representative result – allowing abstraction or generalization if necessary due to knowledge limitations.

Use of a numerical value also supports simplified, enumerated states – e.g. *definitely failing* = 0, *probably failing* = 0.35, *risk of failure* 0.75, *definite success* = 1 – and comparison between different capabilities, without requiring knowledge of capa-

bility semantics. This particularly applies when considering *external* capabilities – e.g. *Truck* and *Helicopter* agents *move* in semantically different ways, but their quality can be compared through their advertised confidence.

Two classes of confidence estimation are described; generic and specific. *Generic* estimation is used if a is unground, and provides a generalized estimation of *ability* for performing that *type* of activity. *Specific* estimation uses a ground a to allow more detailed semantic reasoning, giving more accurate estimation for that specific activity. This also allows estimation to account for where certain states only impact specific activities. For example, a *Truck* can have a general confidence in *move*(?*from*, ?*to*) based upon current health, location, and/or weather condition states – but the confidence for *move*(A,B) is significantly influenced by the current state of road $A \rightarrow B$. We next discuss approaches for estimation, which vary with capability type.

8.2.5.1 Primitive Capability Estimation

Primitive capability confidence estimation is domain and agent dependent, as use of these capabilities equates to directly manipulating the environment. For example, a *Truck* agent controls a Truck vehicle situated within the environment, with primitive capability confidence deriving from the physical condition of that entity and world. As confidence for plans and external capabilities ultimately originates from the confidence of primitive, internal capabilities, this does render efficacy of confidence estimation dependent upon the designer.

Primitive capability confidence will be specific to the environment, capability and/or holding agent type; one example approach is to use conditional rules (such as *if slippery*(?*from*, ?*to*), then *confidence* $= 0.7$), or arithmetical operations based upon individual confidence effects associated with various states. Another more generalized method is to use (time-weighted) historical results. Singh *et al.* [2010], Haigh and Veloso [1998] and Jiménez *et al.* [2006b] all used approaches of probabilistic estimation derived from historical records, for respective purposes of learning plan selection (context) conditions, forming control rules for robot behaviour, and guiding variable instantiation(s). However, this would require failure to occur in order to infer negative states – contradicting our desire to prevent failure in the first place.

8.2.5.2 External Capability Estimation

Confidence estimation for external capabilities uses a fixed value, received in the most recent advertisement for that capability (i.e. from *ag* as defined by the capability). As estimation employs agent beliefs, and as knowledge is often decentralized in a MAS, the recipients of advertisements would likely be unable to for the necessary beliefs if advertisements *did* contain detailed calculation information. Restriction to a fixed value allows semantic knowledge requirements to be limited to the *advertiser* alone, reducing overall knowledge requirements across the system.

This approach effectively involves a 'push' approach – external confidence values must be generalized, as the advertiser cannot be aware of how and when potential dependants would use that external capability (i.e. assume the activity instantiations or execution contexts for future obligations upon them). External capability holders could alternatively form query requests for specific confidence estimates; we have assumed this would entail excessive communications costs, although such a request approach could still be implemented if necessary for a particular domain.

Specific values *can* be provided within established dependency contracts as the precise execution context of activities becomes known (Section 8.4). This offers some mitigation against the generality of advertisements, as well as reinforcing the desirability of forming contractual dependencies as early as possible (to aid maintenance reasoning as well as reserve agent resource).

8.2.5.3 Composite Capability Estimation

In composite capabilities, the confidence value represents the quality of available plans for a in execution context B_a. The estimation function (Algorithm 5) iterates through plans represented by the capability and estimates their individual confidence through the *calculatePlanConfidence* function. General and specific cases are similar, although the latter is able to ground plan (selection) preconditions with the arguments of a to filter out unselectable plans.

A specific implementation is required where composites represent runtime plan formation, to estimate the confidence for any plans the agent *can* generate to achieve a given the initial state B_a. Simply performing planning and evaluating results upon each

confidence call (with formed plans being cached or discarded) would be too computationally expensive to justify, although there may be some utility from forming *relaxed* plans for such a purpose, similar to within domain-independent heuristic planners. The overall capability confidence is the highest individual plan confidence; we assume the agent uses a similar rationale for plan selection.

Algorithm 5: The *confidence* function for a composite capability

Data: a – the goal achieved by the plans represented in this capability
B_a – the execution context for a
Result: $[0..1]$ – a scalar indicator of quality
$conf \leftarrow 0$;
$P \leftarrow$ set of all Plans represented by this capability;
if *a is ground* **then**
    `// Filter by preconditions`
    for *every* $p \in P$ **do**
        if *(preconditions*$^+$ *of* $p \not\subseteq B_a$*)* $\|$ *(preconditions*$^-$ *of* $p \subseteq B_a$*)* **then**
            `// Remove from consideration`
            $P \leftarrow P \setminus p$;
`// Find the best plan's confidence`
for *every* $p \in P$ **do**
    $conf_p \leftarrow$ calculatePlanConfidence(p, B_a);
    if $conf < conf_p$ **then**
        $conf \leftarrow conf_p$;
return *conf*;

8.2.6 Calculating Plan Confidence

Our concern lies primarily with *use* of estimated confidence values, rather than their calculation (which may vary with domain properties and plan libraries). Regardless, it is useful to outline a number of potential approaches for the *calculatePlanConfidence* function; $conf(p,B)$, where p is a plan and B represents the execution context of the first activity.

These approaches share a commonality in that their result ultimately stems from confidence in either primitive or external capabilities (with estimation general or spe-

cific depending upon whether the evaluated plan is ground). As these approaches involve iterating – in execution order – through every $a \in p$, capability knowledge is required both to estimate confidence in a and determine it's effects. In the latter case, the effects of a_n are applied to B_{a_n} (execution context of a_n), to estimate the execution context ($B_{a_{n+1}}$) of the following a_{n+1}.

8.2.6.1 Calculate by minimum activity confidence

One approach is to use minimum activity confidence;

$$conf_{min}(p, B) = \min_{a \in p} conf(a, B_a)$$

This defines plan confidence (Algorithm 6) through the worst *individual* activity (similarly to the *q_min* quality metric of TÆMS). This also reflects our maintenance approach (discussed in the following chapter), where plans are maintained based on individual confidence of their constituent activities – meaning the confidence value directly indicates if p contains at *least* one threatened activity. Finally, if estimating confidence to evaluate against a minimum threshold, α–β pruning can be used; returning immediately upon ensuring confidence will be below that threshold. This offers *potential* optimisation, depending where the first terminating low-confidence activity lies in p, for certain operations such as filtering out low-confidence plans.

Algorithm 6: The *calculatePlanConfidence* function, using minimum constituent activity confidence

Data: p – A plan of n activities; $p = \{a_1, \ldots, a_n\}$
B_a – The estimated execution context for a_1
Result: $[0..1]$ – a scalar indicator of quality
B_a – Updated with the effects of executing p

```
conf_p = 1.1;
for each a ∈ p do
    Capability c_a ← getCapability(a);
    conf_a ← c_a.confidence(a, B_a);
    if conf_a < conf_p then
        conf_p ← conf_a;
    // Update execution context for the next activity
    B_a ← B_a ∪ c_a.effects+(a);
    B_a ← B_a \ c_a.effects−(a);
if conf_p = 1.1 then
    // If cannot find any capabilities, assumes zero confidence
    return 0, B_a;
return conf_p, B_a;
```

8.2.6.2 Calculate by averaged activity confidence

Calculated confidence by minimum activity allows determination of whether or not a plan would require maintenance. However, it can be argued as less representative when comparing plans – a plan with a single, slightly more significant, activity at risk would be regarded as inferior to one with multiple individual points of slightly lower risk. An alternate option is to use an averaged value (Algorithm 7);

$$conf_{average}(p,B) = \frac{\sum_{i=1}^{n} conf(a_i, B_{a_i})}{n} \text{ where } p = \{a_1, \ldots, a_n\}$$

Unlike $conf_{min}$, α–β pruning cannot be used for optimization within $conf_{average}$. One further concern with an averaged approach arises is that it does not account for the *length* of plans. Although this principle fits with our desire for semantic independence (that using a confidence value shouldn't require knowledge of capability semantics, including the length of plans represented by a composite), an average value does not

capture how many *individual* activities are at risk. We can partially mitigate this using a maximum limit on the number of activities assessed from the plan – albeit this would not prevent issues where plans had *less* activities than that threshold.

Algorithm 7: *calculatePlanConfidence* based upon average activity confidence

Data: p – A plan of n activities; $p = \{a_1, \ldots, a_n\}$
B_a – The estimated execution context for a_1
Result: $[0..1]$ – a scalar indicator of quality
B_a – Updated with the effects of executing p
$conf_p \leftarrow 0$;
$count \leftarrow 0$;
for *each* $a \in p$ **do**
  Capability $c_a \leftarrow$ getCapability(a);
  $conf_a \leftarrow c_a$.confidence(a, B_a);
  $conf_p \leftarrow conf_p + conf_a$;
  $count \leftarrow count + 1$;
  `// Update execution context for the next activity`
  $B_a \leftarrow B_a \cup c_a.\text{effects}^+(a)$;
  $B_a \leftarrow B_a \setminus c_a.\text{effects}^-(a)$;
if $count = 0$ **then**
  `// No activities, no possible failure`
  return $1, B_a$;
return $(\frac{conf_p}{count}), B_a$;

One potential risk with an averaged approach lies where the impact of a single, very low confidence activity is 'diluted' by multiple higher confidence activities – meaning the calculated value fails to represent near-certain failure stemming from a single threatened activity. This is addressed by use of a minimal activity approach (i.e. $conf_{min}$) – another approach may be to use a *weighted* average, giving more immediate activities greater significance;

$$conf_{weighted}(p,B) = \left(\frac{\sum_{i=1}^{n} conf(a_i, B_{a_i}) \times w_i}{\sum_{i=1}^{n} w_i} \right), \text{where } p = \{a_1, \ldots, a_n\}$$

The weight given to each activity confidence decreases further into the plan; for example, $w_i = \frac{n-i}{1+i}$ for a p of n activities, with i denoting the currently evaluated activities position. This means confidence more effectively represents whether a plan will initially require maintenance, although it also increases the risk of failing to represent *later* activities at risk of failure.

We can argue this is less of an issue with longer plans, or higher frequencies of exogenous change. It is more likely the *estimated* execution context of later activities will be invalid at execution, reducing the accuracy of estimated confidence values using that assumed context. A weighted approach offers one method to focus estimation upon these earlier activities with greater certainty over their execution context.

8.2.6.3 Analysis

The most appropriate calculation method will itself depend upon the exact planning approach, and how tightly the formation of maintenance plans is constrained (with regards to confidence). The $conf_{min}$ approach is suited to strict planning constraints prohibiting *any* low-confidence activity. However, $conf_{min}$ may not be appropriate if this constraint is unrealistic, and *some* use of lower confidence activities is inevitably necessary; in that case, either $conf_{average}$ or $conf_{weighted}$ are more appropriate, as these offer methods to judge whether a generated maintenance plan provides an *overall* confidence increase.

Confidence estimation entails knowledge requirements, particularly for primitive capabilities. We argue these can be subsumed within domain modelling, as forming a planning domain *or* plan library requires identification of states and their impact upon activity success, to define suitable activity preconditions. This consideration does have to be weighted against the reality that confidence estimation will have greater accuracy when using more specific and detailed information about the impact of states upon success than might be necessary for a purely deterministic domain model.

Primitive capability estimation requires domain specific implementation, meaning responsibility lies with the system designer to ensure efficiency and termination. External capability estimation simply returns a fixed, advertised value, making it of trivial complexity – complexity is offset to the internal capabilities of the advertiser (generat-

ing the advertised estimate).

Composite estimation is more complex, as composites represent multiple plans. Estimation effectively results in a branching tree structure, where the final leaf nodes correspond to the potential primitive plans (Figure 8.1)[6]. Confidence estimation becomes essentially a tree-traversal operation, with complexity $O(n_{plans} \times n_{activities})$ – n_{plans} is the maximum number of complete primitive plans, and $n_{activities}$ the maximum size (number of activities) of any individual primitive plan.

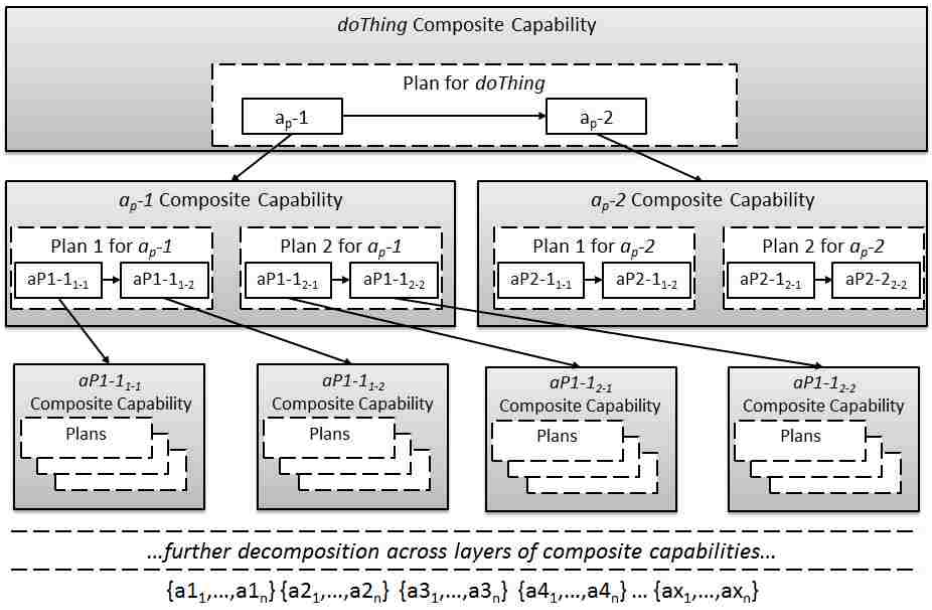


Figure 8.1: An example of progressively branching confidence estimation calls, where a composite capability contain plans which themselves contain composite activities. Ultimately, this 'tree' of calls resolves to confidence estimation for leaf activities.

Composite confidence estimation requires each activity have an associated, estimated, execution context; requiring duplication of the current belief set in memory. In the worst case, effects of an activity a_n are the complete inversion of its execution context B_{a_n}, meaning $B_{a_{n+1}}$ cannot be more efficiently stored through reduction to only the individual state transformations required to reach $B_{a_{n+1}}$ from the initial B_{a_0}.

Our estimation algorithm assumes cyclical references do not occur, or are detected

[6] A primitive plan is one composed *only* of ordered primitive activities, as results from the sequential decomposition or refinement processes of approaches such as HTN planning.

and prevented, such that infinite referential loops do not occur (i.e. where capability c_a uses estimation for c_b, which has plans containing activities matching c_a, etc). If this assumption holds, composite confidence estimation terminates so long as primitive and external confidence estimation terminates for leaf activity nodes. Avoiding cyclical loop circumstances is simplified by defining external capabilities through fixed values, reducing calculation scope to the local (advertising) agent. Finally, domain and implementation dependent optimizations may exist to reduce complexity.

A copy of B_a has to be kept for every instance where a is met by a composite capability, to provide the execution context for each evaluated plan. The number of copies – the extent of memory usage – will scale linearly with the number of primitive plans (i.e $O(n_{plans})$), as each plan requires an associated B to provide (progressively estimated) execution contexts for it's constituent activities.

This memory use may be a concern with tightly bounded agent memory resources; one alternative is to use a two-stage approach, where the agent forms the complete set of potential primitive plans *before* evaluating each in turn – restricting memory requirements to two copies of B_a (a 'backup' of the initial execution context, and one for the currently evaluated plan). This would, however, entail either the composite estimation process be entirely defined by that top-level capability, or agents implement some form of semantic knowledge sharing between capabilities to correctly identify the entire set of possible primitive plans (as multiple decomposition options, at multiple levels, may exist). Our tree-based approach to plan confidence calculation – where composites only know the confidence of a plan activity, not *how* that estimate was generated – decouples and encapsulates such knowledge, allowing potential capability-specific optimisations.

8.3 Maintenance Policies

Policies provide a mechanism for dynamic regulation of system behaviour without requiring modification of the underlying implementation (Tonti *et al.* [2003]) – for example, to specify constraints and/or relaxations upon (fixed, inviolate) planning operators or agent goals[7]. Wark *et al.* [2003] describes policies applied to individual agents, system components, or larger sets of either. Both policy contents and their ap-

[7]This type and use of policy is entirely separate from those formed as a solution to MDPs.

plied scope can be altered during runtime, allowing dynamic modification of system behaviour over varying levels of specificity.

An example of policy application is found in the *Coalition Search and Rescue Task Support* (*CoSAR-TS*) experiment (Uszok *et al.* [2004], Tate *et al.* [2004]). CoSAR-TS examined team planning and execution operations within a search-and-rescue (SAR) environment, consisting of a realistic geography formed of several (fictional) countries (Arabello, Agadez, Binni and Gao). Policies were implemented using the *KAoS* (Knowledgable Agent-orientated System) ontology (Bradshaw *et al.* [1997]) and employed for dynamic adaptation of planning goals and operators by the I-X/I-Plan planning and execution system (Dalton *et al.* [2006]).

For example, Arabello possessed superior hospitals, but denied access to it's airspace for helicopters originating from Gao – constraining vehicle selection. Dalton *et al.* [2006] describe application of policies to add such constraints dynamically, allowing easy modification if Arabello (or other countries) subsequently altered flight restrictions. Policies both added flexibility to, and facilitated reuse of, agents by allowing the addition of agent, environment and situation specific constraints to generalized planning domains as required.

CAMP-BDI uses policies to define variables whose values influence agent maintenance behaviour, with their application ranging from specific agent-capability pairs to the entire system. Policy values can be used to tailor sensitivity of maintenance trigger conditions (and corresponding overall frequency of maintenance activity) against the severity of failure consequences, or to account for computational and time constraints upon agents. They can also be updated if knowledge about agent capabilities and the environment changes over time.

8.3.1 Contents

A maintenance policy ϕ contains the following fields;

- **Threshold** (*Th*): value (where $0 \leq Th < 1$) defining minimum acceptable confidence for an activity a; a is regarded as threatened if $\phi.Th < c_a.\text{conf}(a, B_a)$.
- **TriggerConditions** (*TC*): a set of state atoms, including absence conditions (i.e. $TC = TC^{pres} \cup TC^{abs}$; where TC^{pres} is the set of positive or negative states

which should hold, and TC^{abs} the positive or negative states which should be absent),serving as additional explicit maintenance trigger conditions. Regardless of confidence, a will be regarded as threatened if $TC^{pres} \subseteq B_a$ and $(TC^{abs} \setminus B_a) = TC^{abs}$. This field specifies states which always entail maintenance; either as an optimisation to avoid confidence calculation, or in response to knowledge learned during runtime.

- **Priority** (Pr): value used to prioritise addressing of threatened activities (taking precedence over plan ordering). This allows focus upon certain activity types (when mapping the policy to their associated capability). Prioritisation can be applied if it is advantageous to address threats to a particular type of activity earlier, such as where resources are likely to required (i.e. for advance reservation).
- **DropConditions** (DC): state atoms, defined as for TC. Maintenance planning is treated as intractable for correcting a threat to a if $DC \subseteq B_a$ (for example, if *Truck* is mortally damaged, it cannot address a threat to a). This forces the maintenance algorithm to maintain the *parent* of a – effectively generalizing the problem whilst avoiding execution costs for futile lower scope (a-specific) maintenance behaviour.

Although we define both TC and DC as sets of atoms – effectively disjunctive statements – future work may expand representation to improve flexibility, with one possible approach being to define these fields using conditional rules, applied only for specific activity instances.

8.3.2 Matching to Activities

When maintaining an activity a, the agent must identify the applicable maintenance policy ϕ_a. We assume an advertising and receipt mechanism exists to distribute policy information, such that agent ag has a set of known policies $ag.\Phi$ (including knowledge of which capabilities and/or agents any $\phi \in ag.\Phi$ is associated with). Algorithm 8 defines the *getPolicy* function, used to retrieve ϕ_a, which employs a precedence order when multiple policies can apply to a.

Policies can be associated with specific activity types (i.e. matched to capability); we also assume a *default* maintenance policy $\phi_{default}$ exists. For intentions, where $a \in plan_i$, both a and the associated $goal_i$ may have different policies associated with them. This risks inconsistencies in policy field values, and scenarios where maintenance can

Algorithm 8: The *getPolicy* function

Data: a – an activity / goal

Result: The policy ϕ_a associated with a

if $a \in Dp$ **then**

```
// A contract was formed with an obligant to perform a
```

$contract_a \leftarrow$ dependency contract for $a \in Dp$;

$\phi_a \leftarrow \text{contract}_a.\phi$;

$goal_i \leftarrow$ goal for $plan_i$, where $a \in plan_i$;

$\phi_{goal} \leftarrow$ getPolicy($goal_i$);

return merged(ϕ_a, ϕ_{goal});

else if $a \in Ob$ **then**

```
// a is an obligation
```

$contract_a \leftarrow$ obligation contract for $a \in Ob$;

return $contract_a.\phi$;

else

```
// Locally performed a
```

if $\nexists \phi \in \Phi$ *where* ϕ *applies to* a **then**

```
/* No policy exists for a in the set of known policies,
   identify if there is a plan containing a and if that
   has an associated goal                                  */
```

if $\exists$*plan where* $a \in plan$ **then**

$goal \leftarrow$ goal met by *plan*;

return getPolicy(*goal*);

else

return $\phi_{default}$;

else

$\phi_a \leftarrow$ most recent applicable $\phi \in \Phi$;

if $\exists$*plan where* $a \in plan$ **then**

$goal \leftarrow$ goal met by *plan*;

$\phi_{goal} \leftarrow$ getPolicy(*goal*);

return merge(ϕ_a, ϕ_{goal});

else

return ϕ_a;

restore confidence to meet the threshold defined in a's maintenance policy threshold, but that in $goal_i$'s maintenance policy.

To counter any potential inconsistencies, the maintenance policy returned by *getPolicy* uses *merge* (Section 8.3.3) to combine the policy mapped to a with that mapped to a's goal (either $goal_i$, or the subgoal met by the plan containing a). In hierarchical plans, recursion is used to merge together policies for the chain of decomposed activities, eventually forming a policy which combines that of a and every parent (sub)goal up to and including $goal_i$.

Priority is given to policies originating from obligation or dependency contracts; as these may contain more-specific values defined during contract formation. Otherwise, policies are associated with capabilities; if multiple policies apply, the most recent is used. We assume any policy overlap will stem from failure to review older policies or assignments during policy addition or modification – the most recent policy will likely reflect the most accurate knowledge.

8.3.3 Merging Policies

The *merge* function (Algorithm 9) combines n individual policies into a single merged policy (ϕ_{merged}). ϕ_{merged} disjunctively combines the input policy set, representing a *most constrained* policy. Where the input ϕ set represents the maintenance policy for a and those for a's hierarchical parent activities (subgoals), ϕ_{merged} is defined such that any condition requiring maintenance (triggering conditions) of a parent activity will *also* trigger maintenance of a.

Algorithm 9: The *merge* function

Data: $\phi_1, \ldots, \phi_n$ – n policies, to be merged into one

Result: The merged policy ϕ_{merged}

$\phi_{merged} \leftarrow$ new empty policy;

$\phi_{merged}.Th \leftarrow \min_{x=1\ldots n}(\phi_x.Th)$;

$\phi_{merged}.TC \leftarrow \phi_1.TC \cup \ldots \cup \phi_n.TC$;

$\phi_{merged}.Pr \leftarrow \max_{x=0\ldots n}(\phi_x.Pr)$;

$\phi_{merged}.DC \leftarrow \phi_1.DC \cap \ldots \cap \phi_n.DC$;

return ϕ_{merged};

The confidence threshold of ϕ_{merged} is the *lowest* value for policies ϕ_1 to ϕ_n; priority the *highest*. $\phi_{merged}.TC$ is the disjunction of *all* trigger conditions in the passed in policies; i.e. if TC holds for any one of the input policies, it should also hold for ϕ_{merged}. $\phi_{merged}.DC$ is the *intersection* of all DC fields – i.e. for ϕ_{merged} to denote the associated a cannot be maintained, *no* parent goals (where these 'contribute' input policies to *merge*) must be able to be.

8.4 Contracts

We assume agents co-operate to achieve goals, with long-term planning required to identify and reserve access to necessary agent capabilities and material resources. Broersen *et al.* [2002] define desires (and, by extension, resultant intentions) as internally or externally motivated, based upon whether they originate from the agent or another (dependant). We will refer to an intention as externally motivated if $goal_i$ corresponds to an accepted obligation.

Joint activity requires agents to share relevant information regarding their beliefs and plans – such as within Joint Intentions (Levesque *et al.* [1990]) and Joint Responsibilities (Jennings [1992]) theory. This often requires agents form agreements between each other, to represent information regarding joint activities and mutual beliefs; we refer to these as *contracts*.

CAMP-BDI assumes contracts are formed between obligants and dependents; we use these to convey information about *specific* activities beyond that in generalized capability advertisements or policy definitions. Contracts are viewed as formed as far in advance as possible once a plan is known; to both protect against contention and allows proactive use of the information within. We refer to the sets of obligation and dependency contracts held by a CAMP-BDI agent as Ob and Dp respectively; these can be viewed as a subset of B.

8.4.1 CAMP-BDI specific fields

CAMP-BDI maintenance behaviour requires dependency contract *contract* to contain the following fields[8];

- The (ground) **activity** to be performed (*contract*.α). The obligant(s) must hold a corresponding *internal* capability, the dependants a corresponding *external* capability (used to identify obligants). For brevity, we refer to the contract where α=a as $contract_a$.
- A set of **casual links** (*contract*.*CL*); add and delete sets ($CL = CL^+ \cup\ CL^-$) representing the cumulative effects of prior activities (to be) performed in the dependant $plan_i$. Obligants can estimate the execution context B_α for α by combining this information with current beliefs; i.e. $B_\alpha = (B \cup CL^+) \cap (B \setminus CL^-)$.
- An **external capability** (*contract*.c_α); a instance of the capability model from Section 8.2.2, with field values being set by the obligants(s) to reflect how they intend to perform α, and using a B_α estimated with the contents of *CL*. Whilst c_α's confidence function utilises a static value, this can be set by the obligant(s) as the result of *specific* estimation using internal capability knowledge.
- A maintenance **policy** (*contract*.ϕ) reconciling the potentially differing policies of the involved parties in the joint activity. This can be formed by merging dependant and obligant policies using the *merge* function (Section 8.3.3).

The dependant defines the contents of *a*,and *CL* in *contract*. Both dependants and obligants contribute to the formation of ϕ. The c_α field originates from the obligant(s) performing α; if α is a joint activity (performed by multiple agents), $contract.c_\alpha$ combines their individual capability specifications. Where *n* agents contribute capabilities $c_{\alpha_1} \ldots c_{\alpha_n}$, c_α is formed such that;

- *ag* represents a *set* (i.e. team) of obligants. This resembles a *holon* (Schillo and Fischer [2003]), where the obligant team is treated as a single, unified, agent performing α.
- Both *s* and $g(\alpha)$ should be common across all dependants and obligants.
- $pre(\alpha)$ is the intersection of all preconditions, $c_{\alpha_1}.pre(\alpha) \cap \ldots \cap c_{\alpha_n}.pre(\alpha)$; i.e. the relevant internal capability preconditions must hold for all obligants in order to perform α.

[8] We define the fields required by our maintenance algorithms here, although the *complete* set of contractual information may vary with specific implementations.

- $eff(\alpha)$ is the union of all effects, $c_{\alpha_1}.eff(\alpha) \cup \ldots \cup c_{\alpha_n}.eff(\alpha)$ - the combined effects of all obligants.
- $conf(a, B_\alpha)$ is a fixed value, formed from $c_{\alpha_x}.conf(\alpha, B_\alpha)$ for $x = 0, \ldots, n$ using the same approach as plan confidence estimation (Section 8.2.5).

All agents involved in performing α hold the same contract, with $contract_\alpha$ stored (as appropriate) in their *Ob* or *Dp* set. Similarly, α must be met by an internal or external capability of that agent, respectively depending on whether it is an obligant or dependant. For generality, we do not mandate specific methods for establishing dependencies – provided resultant contracts define these fields.

8.4.2 Usage and Execution

Obligants are expected to form a plan for α in advance; this can be stored for execution upon request, and is referred to here as a *suspended* intention (with $goal_i$ the – externally motivated – goal to perform α). Although generalized reasoning can be performed using the relevant composite capability, pre-emptively forming $plan_i$ allows obligants to both form their required dependencies and perform maintenance in response to exogenous change even before α begins execution.

This permits multilevel distributed plans, as obligants establish dependencies for their cached plans (and so forth, for sub-obligants). It also allows obligants to continually act to ensure mutual belief in their ability to perform α, whilst updating the external capability within the contract. However, there is an additional communications cost of sending contract updates upon maintenance changes to a suspended plan – we argue this to be a justifiable trade-off as required resources can identified and reserved further in advance, and more accurate information can be employed in dependant maintenance.

The dependant is quiescent during *execution* of α, waiting for obligants to complete before continuing the dependant $plan_i$. As we wish to minimize semantic knowledge requirements upon dependants, obligants do not share their specific $plan_i$ for performing α. This reduces the information communicated (particularly for dependency hierarchies), but means dependants are not aware of any fluent states during execution of individual activities within obligant plans. This restricts agents from executing dependencies in parallel, as maintenance cannot account all potential interactions between

multiple simultaneous delegated activities without such knowledge. Examining ways to support maintenance reasoning with parallel activities, without requiring detailed semantic knowledge about fluent states, is an area for future investigation.

8.4.3 Contract Policies

Formation of a dependency contract sees the dependant hold α within $plan_i$, and the obligant(s) form (to execute on request) an intention where $goal_i$ corresponds to α. As policies can be associated on an agent basis, with a risk of ambiguity if the agents involved have different applicable policies, the $contract.\phi$ field is formed by using *merge* (Algorithm 9) to combine all dependant and obligant(s) policies. This process, including exchanging of relevant policy information between agents, is assumed to lie within the specific implemented contract formation protocol.

The contract policy ensures consistency of maintenance behaviour. If the dependant receives updated confidence for α (assuming maintenance always occurs before any contract *EC* updates are communicated), the obligant(s) must have already identified and attempted to handle maintenance tasks at their own $plan_i$ level, due to their shared *Th* values. This ensures maintenance changes impact a minimal subset of a distributed plan, as higher hierarchical level maintenance changes will only occur if the – more specific – obligants are unable to restore preconditions and confidence about that shared *Th*. The distributed maintenance process is detailed in Chapter 10, including adoption of maintenance responsibility by agents.

8.5 Conclusion

This chapter contributes a supporting architecture, to provide the information required for our maintenance reasoning. These data objects are regarded as a subset of the *Beliefs* of a CAMP-BDI agent; although our focus (and specification) lies upon use of such information to improve robustness of selected intentions, future work may investigate how such information (with particular regard to capabilities) could also aid desire and intention selection.

Three knowledge components are provided; capability objects, contracts, and policies. *Capabilities* provide agents with knowledge to both support introspective reason-

ing about intended plans and form planning goals when performing proactive maintenance. *Contracts*, assumed to be a pre-existing requirement for multiagent dependency formation and execution, convey capability-formatted dependency information from obligant(s) to a dependant. This allows the latter to use the same capability-driven introspective reasoning for delegated activities, but offsets semantic knowledge requirements to the obligant (as part of the latter's provision of the external capability field).

Finally, *Maintenance Policies* aids flexibility by allowing runtime modification of maintenance behaviour-driving variables. These allow reaction to observed maintenance behaviour (such as reducing sensitivity in the event of excessive planning) or following changes in environmental knowledge. Policies, combined with a merging process and their specification in contracts, also serve a purpose in guiding distributed maintenance as a decentralized process – discussed in Chapter 10.

Chapter 9

The CAMP-BDI Maintenance Algorithm

The previous chapters described desired maintenance behaviour and defined a supporting architecture. This chapter describes our core contribution; the *maintenance algorithm* used by agents to anticipate threats to planned activity and perform corrective plan modification. We refer to agents employing this algorithm, and utilizing the previously described supporting architecture, as *CAMP-BDI agents*.

9.1 CAMP-BDI Agent Reasoning Cycle

We define the reasoning cycle of a CAMP-BDI agent in Algorithm 10 by extending the generic BDI algorithm given by Rao and Georgeff [1992]. Explicit maintenance, contract formation and contract update steps are inserted; the latter two reflecting assumptions over contract-driven multiagent behaviour whilst also facilitating distributed maintenance (discussed in Chapter 10). Contract formation is defined within the reasoning cycle to delineate its temporal relationship to maintenance.

We refer to *maintaining* an intention i as the process of first identifying activities under threat in $plan_i$, followed by addressing those threats through modifications to $plan_i$ (aimed at ensuring achievement of the associated $goal_i$). The maintenance process is formed of an initial *agenda formation* step (defining a set of identified threats, or *maintenance tasks*), followed by *task handling*; these two steps are combined into the *maintain* function. This division into discrete steps supports the potential and separate future examination of different approaches to diagnosis (agenda formation) and handling (addressing agenda contents).

Algorithm 10: The CAMP-BDI reasoning cycle, with changes from the generic algorithm (Rao and Georgeff [1992]) in black text. Those relevant to local maintenance behaviour are **underlined**; others are relevant to the distributed case and described in Chapter 10.

```
initializeState();
while agent is alive do
    msg_Ob ← extractObligationMaintainedMessages(eventQueue);
    B ← updateBeliefs(eventQueue, B);
    D ← optionGenerator(eventQueue, I, B);
    I ← deliberate(D, I, B);
    i ← updateIntentions(D, I, B);
    if i ≠ null & i not waiting on a dependency to complete then
        maintain(i, B);
        formAndUpdateContracts(i, B);
        execute();
    else if (Dps ≠ ∅) & ( msg_Ob ≠ ∅) then
        maintainDependencies( msg_Ob);
    else if Obs ≠ ∅ then
        maintainObligations(B, Obs);
    getNewExternalEvents();
    I ← dropSuccessfulAttitudes();
    I ← dropImpossibleAttitudes();
    I ← postIntentionStatus();
```

The maintenance process occurs after the agent selects an intention (i) – i.e. after beliefs have been updated. This avoids maintaining *all* possible intentions before selecting i, which has obvious computational overhead (particularly for large sets of potential intentions). Maintenance changes to *un*selected intentions also risk becoming redundant – or suboptimal – due to execution post-effects of the *selected* i. We assume selection of i is driven by the desirability of $goal_i$, and that maintenance modifications – even if $plan_i$ complexity increases – will never invalidate the decision to select i.

A further assumption is that agents generally will pursue intentions to completion. Interleaving independent (separately motivated[1]) intentions make it more difficult to anticipate future activity threats, as the execution context for future activities is defined less by the effects of predecessors in the same $plan_i$ than by those of *other* intended plan's activities. We do not *prohibit* such interleaving, but frequent interleaving will reduce the utility of maintenance – particularly with more temporally distant activities, due to increasing uncertainty over exactly which plans and activities will be executed (and which post-effects will occur to create future execution contexts).

The *maintain* function, given intention i, examines threats to activities in the associated $plan_i$ based upon current beliefs (including regarding agent capabilities). If threats exist to activity success, *maintain* (attempts to) modifies $plan_i$ in mitigation. There are several conditions for attempting maintenance; selection of an intention, receipt of a post-maintenance message from an obligant (dependency triggered maintenance), or – if no intention was selected – the existence of suspended intentions used to meet obligations. We address the first scenario in this chapter, and the remainder in the following chapter. Maintenance (Algorithm 11) first forms an *agenda* – a priority ordered list of maintenance tasks, where each task identifies an activity in $plan_i$ at risk of (potential) failure. For simplicity of reference, we assume $plan_i$ contains only activities still to be executed, rather than preserving those *already* executed.

Algorithm 11: The *maintain* function

Data: i – An intention, formed from a goal and plan ($goal_i$, $plan_i$)
B – The (estimated) execution context for the first $a \in plan_i$
Result: **True** if i's $plan_i$ was modified
$handled \leftarrow$ false;
$MT \leftarrow \emptyset$;
$MT \leftarrow$ formAgenda($goal_i$, $plan_i$, B, MT);
while $\neg handled \& \neg\, MT = \emptyset$ **do**
  $mt \leftarrow$ remove highest priority member of MT;
  $handled \leftarrow$ handleTask(mt, i);
return *handled*;

[1]Rather than where a dependent intention has an indirect dependency upon *another* intention it holds (or has committed to perform in future). For example, *LogisticsHQ* may form a dependency upon *Truck1* to deliver cargo, with *Truck1* forming a sub-dependency upon *LogisticsHQ* to clear a required road.

If a non-empty agenda is formed, *maintain* will iterate through it in priority order; terminating when a maintenance task is successfully handled *or* none can be. Handling a task entails modification of $plan_i$ through identification and insertion of an appropriate subplan (*maintenance planning*); this potentially invalidates *other* tasks in the agenda (i.e. if the inserted activities' effects remove threat conditions, or if the threatened activity is removed) – hence the termination. If further activities in $plan_i$ remain under threat, these will be handled in subsequent execution cycles; agenda prioritization attempts to ensure the most urgent threats are handled immediately, with lower priority tasks regarded as deferrable. This provides a specific termination condition, restricting *maximum* iterations to the agenda size (itself bounded to plan size). An alternative approach could continuously iterate through agenda formation and task handling until no tasks are identified (or maximum iteration limit reached) could be employed. This would risk significant computational cost, potentially delaying activity execution to the extent of risking exogenous change *during* those iterations.

The following sections detail maintenance tasks, followed by the task identification and handling algorithms. For externally motivated intentions (obligations), the dependant must be messaged after maintenance with updated information; this process is detailed in the following chapter.

9.2 Maintenance Tasks

Maintenance Tasks hold key information regarding threatened activities, used to determine requirements for appropriate handling. A maintenance task mt_a, concerning threatened activity a, is represented by the following:

$$mt_a = \langle type, a, B_a, conf_a \rangle$$

The *type* field guides goal specification for maintenance planning, and the insertion of the generated maintenance plan. We define three types, derived from considering activities in terms of STRIPS operators (Fikes and Nilsson [1971]):

- *Preconditions* tasks indicate anticipated preconditions failure; maintenance should seek to preserve a by inserting a maintenance plan that re-establishes required states before execution of a.
- *Effects* tasks denote the agent has unacceptably low confidence in a.

- *Effects_for_pre* tasks are a special subclass of *effects* task, indicating preconditions of a do not hold, maintenance has failed in previous attempts to restore those preconditions, and that a is due for imminent execution. This permits relaxation of confidence constraints upon maintenance planning, to (attempt to) avoid definite failure at the cost of accepting weaker maintenance plans. We assume *potential* non-deterministic failure of maintenance plan activities does not carry more severe consequences than definite preconditions failure – if not, failure could be permitted where reactive recovery methods are available.

The other fields provide information applied in plan formation. B_a provides the execution context for a, providing an initial state specification for maintenance planning; $conf_a$ provides the previously estimated confidence for a, allowing comparison against generated maintenance plans.

The maintenance task object does incur an additional memory cost due to the duplication of an agent belief base in B_a. This can be an issue where agents have restricted memory resources, particularly for large agendas. Here, we focus upon the *use* of B_a in maintenance, and leave the semantics of storage or generation for further domain specific research. We can, however, suggest two possible methods to address this issue in a practical implementation. Firstly, if sufficient computational capacity exists, the agent can simply re-calculate B_a upon demand using sandbox simulation to progress the current B – walking through preceding plan steps (to a) in execution order and adding effects appropriately. Alternatively, *mt* could be limited to the set of *changes* ($\triangle\ B_a$) – i.e. the execution-ordered effects of preceding activities. This would allow B_a to be formed by applying $\triangle B_a$ to the current B. This reduces memory requirements, provided $\triangle B_a$ is smaller – contains less literals – than the resultant B_a. Use of such an approach would, however, need to consider the computational overhead of calculating and subsequently applying $\triangle B_a$.

9.3 Agenda Formation

The *formAgenda* function (Algorithm 12) performs the diagnostic stage of maintenance; i.e. $formAgenda(plan_i, B_i) \rightarrow MT$, where MT is the agenda (maintenance tasks) for $plan_i$, and B_i is the execution context for the first activity in $plan_i$.

Algorithm 12: The *formAgenda* function

Data: g – a set of goal states

p – a plan of n activities for achieving g

B_a – estimated execution context of the first activity in p

agenda – priority ordered list of maintenance tasks; passed in empty in the initial (top-level) call

Result: *agenda* updated with maintenance tasks for p

B_a updated with post-effects of p (used by recursion)

$B_{start} \leftarrow$ copy of B_a;

for *each activity* $a \in p$ **do**

  if *a is abstract* **then**
    return *agenda*;

  $c_a \leftarrow$ getCapability(a);

  if c_a = *null* **then**

    $agenda = agenda \cup$ new MaintenanceTask(*effects*, a, B_a, 0);

    Update B_a with effects of goal a;

  else if *(*c_a *primitive* $\|$ c_a *external* $\|$ *(*c_a *composite* $\&\neg$*decomposed))* **then**

    existingDependencies $\leftarrow$ $false$;

    if *a or any subsequent activity* $\in Dp$ **then**

      existingDependencies $\leftarrow$ $true$;

    $agenda \leftarrow$ identifyLeafTask(g, a, B_a, *agenda*, *existingDependencies*);

    Update B_a with c_a.effects(a);

  else if c_a *composite* & *decomposed* **then**

    $p_a \leftarrow$ subplan decomposing a;

    agenda, $B_a \leftarrow$ formAgenda(g, p_a, B_a, *agenda*);

  agenda $\leftarrow$ consolidate(g, p, *agenda*, B_{start});

  return *agenda*, B_a;

Agenda formation uses sandbox simulation; the algorithm takes a copy of initial beliefs B and iterates through the leaf activities of $plan_i$ in execution order (a leaf is any activity not currently refined by a subplan – including undecomposed composite activities in continually planning agents). For each leaf activity, capability knowledge is used to determine if that activity is at threat – inserting a corresponding maintenance task if so – and (by applying capability-defined effects) estimate the execution context for the following activity. Each call of *formAgenda* returns both the agenda for a given plan – with any new tasks inserted – and B_a representing the post-execution state following the last activity. The latter is not used when called initially from *maintain*, but provides subsequent activity execution context when recursive calls are being made.

The *identifyLeafTask* function(Algorithm 13) is called to evaluate leaf activities:

$$identifyLeafTask(B_{a_n}, a_n, MT, existingDependencies) \rightarrow MT', B_{a_{n+1}}$$

i.e. for leaf activity a_n, execution context B_{a_n}, *identifyLeafTask* returns MT modified to include any maintenance task for a_n and estimates – using capability knowledge for a_n – the execution context $B_{a_{n+1}}$ for any following activity.

The function *getCapability* (Section 8.2.4) identifies the capability associated with a_n. The resultant c_a is used to identify threats to the successful execution of a_n, either due to violated preconditions or lack of quality (denoted by confidence). The $existingDependencies$ boolean is true if a_n or any subsequent $a_{n+x} \in plan_i$ has an associated existing dependency contract; indicating modifications replacing a_n and following activities would entail cancellation of existing contracts.

Preconditions tasks are generated where preconditions failure of a is anticipated *and* a achieves part of $goal_i$, or a or some following leaf activity(s) in the current subplan represent an existing dependency. These conditions seek to preserve a due to a being (potentially) necessary to achieve $goal_i$, or to avoid potential agent resource availability issues (if a cancelled dependency later turns out as still required). Preconditions tasks will never be generated if a has already *begun* execution, as preconditions must have already been satisfied.

Algorithm 13: The *identifyLeafTask* function

Data: a – an activity

B_a – estimated execution context of a

g – the $goal_i$ achieved by the $plan_i$ containing a

agenda – the current agenda of maintenance tasks

dependencies – true if a or successors in the source $plan_i$ corresponds to a dependency contract

Result: *agenda* updated with maintenance task for a, if identified

B_a updated with post-effects of a

$c_a \leftarrow$ getCapability(a);

$\phi_a \leftarrow$ getPolicy(a);

$contributesToGoal \leftarrow (((c_a.g^+(a) \cap g^+) \cup (c_a.g^-(a) \cap g^-)) \neq \emptyset)$;

$preconditionsHold \leftarrow (c_a.\text{pre}^+(a) \in B_a)$ & $(c_a.\text{pre}+-(a) \notin B_a)$;

if $\neg preconditionsHold$ & $\neg$ *currently executing* &

(contributesToGoal $\|$ *dependencies)* **then**

  $agenda = agenda \cup$ new MaintenanceTask(*effects*, a, B_a, ϕ_a, 0));

else

  $conf_a \leftarrow c_a.conf(a, B_a)$;

  $triggerConditionsMet \leftarrow (\phi_a.TC^{pres} \subset B_a)$ & $((\phi_a.TC^{abs} \setminus B_a) = \phi_a.TC^{abs})$;

  if $\neg preconditionsHold$ **then**

    $agenda = agenda \cup$ new MaintenanceTask(*effects_for_pre*, a, B_a, 0));

  else if $(conf_a < \phi_a.Th) \| triggerConditionsMet$ **then**

    $agenda = agenda \cup$ new MaintenanceTask(*effects*, a, B_a, $conf_a$));

  else if $c_a.type = external$ & $a \notin Dp$ & *contract formation has been attempted and failed for* a & $B_a = B$ **then**

    $agenda = agenda \cup$ new MaintenanceTask(*effects*, a, B_a, 0));

$B_a \leftarrow B_a \cup c_a.eff^+(a)$;

$B_a \leftarrow B_a \setminus c_a.eff^- a$;

return *agenda*, B_a;

As handling preconditions tasks is additive to $plan_i$ – through insertion of activities to re-establish required states – conditions for their generation are relatively constrained, to avoid continual iterative expansion of plan length where it may be simpler to instead insert a substitute. For example, if following route $A \rightarrow G \rightarrow J \rightarrow K$ where A, G, and J are dangerous, it is likely more efficient for *Truck1* to form an alternate route to K rather than insert *secureArea* activities for each dangerous location (which would increase distributed plan complexity as well as local $plan_i$ length).

Effects tasks are generated by using the maintenance policy (ϕ_a) mapped to a. An effects type maintenance task is generated if confidence in a (estimated by $c_a.conf(a, B_a)$, where B_a is the – estimated – execution context of a) is below $\phi_a.TH$, trigger conditions $\phi_a.TC$ hold in B_a, or preconditions do not hold (i.e. zero confidence) but it is not necessary to preserve a. These tasks are also generated where the agent has been unable to form a dependency contract for a, and a is next to execute. We assume attempts to execute a delegation-requiring task without an existing contract lead to immediate rejection and failure; the effects task, when handled, acts to trigger reconsideration of the plan such that it can be evaluated whether a is necessary and (based upon external capability knowledge) considered a valid dependency option. Finally, **effects_for_pre** tasks are generated if preconditions do not hold, previous maintenance attempts have failed *and* a is about to execute (i.e. will fail).

Generated tasks are added in the agenda MT. MT is ordered first by task priority (i.e. $\phi_a.Pr$), then by precedence in the plan – if no high priority task is defined in MT, the first task will correspond to the activity closest to execution. We allow (policy defined) out-of-order prioritisation to consider activities where successful maintenance change is likely to have temporal restrictions – for example, if maintenance changes for a particular activity type are (believed) likely to introduce dependency requirements, it may be preferable to form necessary contracts earlier to avoid contention.

9.3.1 Task Consolidation

A plan or subplan may have multiple threatened activities, resulting in multiple maintenance tasks. As the algorithm terminates upon successful task handling, unresolved issues may not be addressed until subsequent maintenance. The *consolidate* function (Algorithm 14) compensates by aggregating multiple tasks into a single task, allowing

them to be addressed through a single handling call rather than over multiple maintenance iterations (albeit with reduced task specificity, as unthreatened parts will be effectively aggregated with those in the subplan that *are* threatened).

Algorithm 14: The *consolidate* function

Data: g – The goal for the p
B – estimated execution context of $a_1 \in p$
agenda – priority ordered list of maintenance tasks for this level
Result: *agenda* updated with consolidated maintenance tasks
if *agenda contains* $\leq$ *1 task* **then**
  return *agenda*;
$conf_{min} \leftarrow 1$;
for *each* $mt \in agenda$ **do**
  if $mt.type = preconditions \parallel mt.type = effects\_for\_pre$ **then**
    $type \leftarrow effects\_for\_pre$;
    $conf_{min} \leftarrow 0$;
  else if $mt.conf < conf_{min}$ **then**
    $conf_{min} \leftarrow mt.conf$;
$\phi_g \leftarrow$ getPolicy(g);
$consolidatedTask \leftarrow$ new MaintenanceTask(*effects*, g, B, ϕ_g, $conf_{min}$);
$agenda \leftarrow \emptyset$;
$agenda \leftarrow agenda \cup consolidatedTask$;
return *agenda*;

The *consolidate* function is called following each recursive *formAgenda* call for a subplan:

$$consolidate(goal, B, agenda) \rightarrow agenda'$$

A modified $agenda'$ contains (if appropriate) the results of aggregating tasks within *agenda* into a single maintenance task. The *goal* represents the hierarchical parent goal met by a plan or subplan (a composite activity or, at the top level, $goal_i$) and B it's execution context; these are used to form the merged maintenance task.

We reason it is preferable to replace a subplan entirely in a single maintenance operation than perform multiple changes over iterative maintenance loops. This reduces the number of individual disruptions (if not necessarily the overall *scope*) to a distributed plan compared to multiple iterative calls. Consolidation also acts to aggregate the cost of individual maintenance planning operations into the handling of the single consolidated activity. It would be possible to adopt a rule where multiple tasks result in replanning the entire $plan_i$ – but this would cause unnecessary disruption if threatened activities lay solely within an isolated sub-part of the complete (hierarchical) plan.

Consolidation does increase the likelihood of dependency cancellation(s) from subplan replacement. Whilst this suggests additional communications costs from cancelling existing – and potentially forming new – dependencies, iterative revision of a subplan from handling multiple agenda tasks would also require communication of updated contract information. In our algorithm design, we opt to view it as better to risk this initial communications cost and limit disruption of distributed plans to as few iterations as possible, rather than face multiple lesser changes (and communicated contract updates) over sequential reasoning cycles.

9.4 Task Handling

Given a maintenance task *mt* *handleTask* (Algorithm 15) performs preventative modifications to the $plan_i$ containing $mt.a$, returning *true* if $plan_i$ was changed (*mt* addressed):

$$handleTask(mt, i) \rightarrow boolean$$

The exact semantics for handling *mt* depend on type, and are defined within Sections 9.5 and 9.6. The most significant element of behaviour in this function is handling of *preconditions* type tasks; if preconditions cannot be established for $mt.a$, an equivalent *effects* task is generated and (attempted) handled. This allows potential substitution of $mt.a$, recognizing that preconditions maintenance is more constrained due to the (likely) high specificity of precondition definition and at higher risk of failing in plan modification than effects equivalents.

Algorithm 15: The *handleTask* function

Data: *mt* – A maintenance task

i – The intention requiring maintenance; $i = \{goal_i, plan_i\}$

Result: **boolean** – true if $plan_i$ is modified and *mt* addressed.

```
handled ← false;
if mt.type = preconditions then
    handled ← handlePreconditionsTask(mt, i);
    if ¬handled then
        mt ← new MaintenanceTask(effects, mt.a, mt.B_a, mt.conf_a);
    else
        return handled;
return handleEffectsTask(mt, i);
```

9.4.1 Forming Planning Operator Sets From Capabilities

Both *handlePreconditionsTask* and *handleEffectsTask* functions attempt to generate a maintenance plan. This requires specification of a *planning problem*. We define a planning problem $pp = \langle O, I, G \rangle$; where O is a set of operators, I an initial state and G a goal (a set of atoms)[2]. A plan solves *pp* if it can achieve all states in G, using the activities defined by O, and starting in state I.

The *formOperators* function (Algorithm 16), given initial state B_{init}, forms a set of capabilities which can be used to form O. B_{init} is used in filtering the returned capability set; only capabilities with *general* confidence in B_{init} above the *Th* value of their associated maintenance policy will be returned. This helps prevent, albeit without guaranteeing prevention of, generation of low confidence plans – as activities in a formed plan may still have unacceptable *specific* confidence, which will depend upon plan semantics and the execution context for each activity.

This filtering is most effective where general confidence estimation provides a *minimum* value – i.e. any *specific* activity cannot have lower confidence. This is most viable where confidence is more dependent upon generalized agent state than specific

[2]Following the STRIPS planning problem definition given by Fikes and Nilsson [1971] as $\langle F, O, I, G \rangle$, where F is a set of boolean variables. We omit F as this is assumed constant for – and consequently common to – all *pp*s formed by our maintenance algorithms.

Algorithm 16: The *formOperators* function

Data: B_{init} – the initial state for planning, expessed as a Belief set.
threshold – a minimum confidence threshold.

Result: C_{op} – forms a set of capabilities which can be employed as (to define) an operator specification.

$C_{op} \leftarrow \emptyset$;
for $c \in C$ **do**
  if $c.confidence(c.s, B_{init}) < threshold$ **then**
    $C_{op} \leftarrow C_{op} \cup c$;
return C_{op};

activity semantics. However, this 'minima' approach could risk over-constraining the capability set when forming operators; for example, the confidence for a *move* activity would stem from the worst-case road (lowest-confidence possible instantiation) in the environment – possibly leading to rejection of the capability as an operator source on the basis of a single negative state road (whose use might not be required, or easily avoided, by any eventual plan).

Filtering by a minima general confidence guarantees resultant plans will have sufficient confidence – but risks missing plans where activities can have sufficient *specific* confidence, regardless of their general confidence estimate. Requiring a definitive minimal value also complicates implementation, by requiring total knowledge of possible execution contexts and instantiations to identify that worst case. Consequently, we regard general estimation as indicative; it does not guarantee specific confidence below a set threshold, but does help guard against it. The *formOperators* operation is thus regarded as helping *guide* maintenance planning by removing some infeasible operators. This represents a middle ground between not filtering operators (risking generation of deterministically 'legal' plans with nonetheless low confidence), and excessive filtering (where the capability set is too restricted for any plans to be found).

This requires generated maintenance plans to be evaluated and potentially rejected – entailing costs of confidence estimation and (retrospectively) of generating a rejected plan. One benefit of introducing acceptance criteria is allowing decoupling of planner semantics from agent implementation. As maintenance behaviour evaluates generated

plans (and selecting operators), the chosen planning method is not required to hold responsibility for, or knowledge of, agent capabilities or plan confidence constraints beyond those required and specified in the planning problem.

As no specific method is mandated for planning, we do not define a specific approach for forming operator specifications from capabilities. It is intuitive, though, that capability preconditions and effects knowledge can be translated to STRIPS-style (Fikes and Nilsson [1971]) equivalents (with composite capabilities comparable to refinements in HTN planning). However, filtering *is* of reduced utility if the planner is *probabilistic*, as this may be able to directly model confidence as denoting the probabilistic chance of achieving the defined effects states. The exact implementation of *formOperators* cannot be entirely divorced from the specific planning implementation; this section primarily illustrates it's general viability through an example design.

9.4.2 The Maintenance Planner Component

Handling a maintenance task *mt* entails identification and insertion of a *maintenance plan* into the $plan_i$ containing $mt.a$. The *formPlan* function generates this maintenance plan (pp_p):

$$formPlan(pp) \rightarrow p_{pp}$$

Here, *pp* defines a planning problem $pp = \{pp_{operators}, pp_{init}, pp_{goal}\}$, where:

- $pp_{operators}$ is a set of capabilities, as returned by the *formOperators* function described in the previous section.
- pp_{init} specifies an initial state, using the same representation model as agent *Beliefs*.
- pp_{goal} is a goal specification, again employing the same state representation model as Beliefs.

The values of these fields are specified as part of maintenance task handling, as described in Section 9.5 and Section 9.6, and can be seen to generally follow the planning problem definition of classical STRIPS planning.

The resultant p_{pp}, if found, is a plan (linear sequence of *n* activities; $p_{pp} = \{a_1, \ldots, a_n\}$) that establishes the states required by pp_{goal}, when executed in pp_{init}, and where each

$a \in p_{pp}$ corresponds to some $c \in pp_{operators}$ (i.e. the agent is capable of executing every activity, either locally or through delegation).

CAMP-BDI is intended as planner-agnostic – i.e. we wish to treat the actual planning implementation as an effective 'black box' component. This allows flexibility to adopt specific planners for agent implementation based upon computational considerations; i.e. potentially ranging from classical or HTN runtime planners to the selection from libraries of preformed plan recipes. This will also entail implementation-specific requirements for translating *pp* to the appropriate problem specification (input) for that planner ($input_{planner}$), and converting the resultant output ($output_{planner}$) into a linear sequence of activities to return as *p*; Figure 9.1 suggests a design where the *Problem Converter* and *Plan Converter* components perform these respective operations.

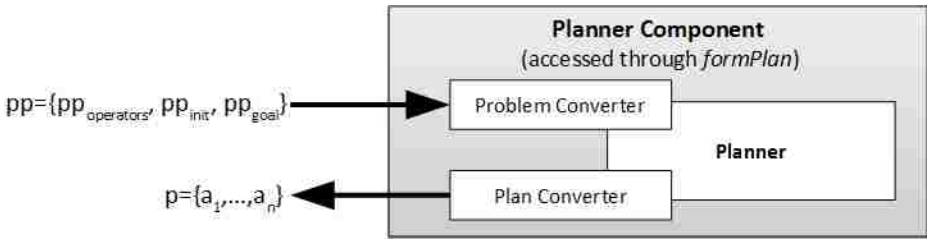


Figure 9.1: Abstracted view of *formPlan* design.

The issue of conversion is likely to be implementation specific, and dependent upon the semantics of both the CAMP-BDI agent (including that of capabilities, and the Belief representation model used to specify the initial state and goal) and selected planner. We can generalize the requirements of the Planner component as to take an input problem specification which can be formed from the contents of *pp*, and produce an output plan (if found) which can be linearized (or scheduled) into an activity sequence; i.e. a planner can be employed *iff* some process exists for conversion such that $pp \rightarrow input_{planner}$ and $output_{planner} \rightarrow p$.

As we leave the converter components as abstract due to their implementation specificity, this obviously allows for an extremely broad range of planning implementations to be employed; however, more restrictive assumptions can be formed with consideration of the most likely *practical* implementations. Our capability model, and the information within *pp* can be seen to closely correspond to classical planning rep-

resentations such as STRIPS or PDDL (which also informed our initial specification of the Capability model in Section 8.2.2, with a natural mapping similarly existing between classical plans and the required format of p.

HTN methods may also potentially be employed. Where the planning domain $D = (O, M)$, the set of operators O can be formed from the primitive capabilities within $pp_{operators}$, and the set of task decomposition methods M can be formed with each $m \in M$ corresponding to an individual plan (with selection precondition) represented by a composite capability[3]. This will also require the declarative goal pp_{goal} to be converted to an equivalent achievement task (e.g. $achieve[pp_{goal}]$); the resultant HTN will also require traversal to form a sequential set of leaf nodes, which can then serve to define the activities in p.

In Chapter 6, we formed the assumption that conformant, (PO)MDP or conditional/contingent approaches would be infeasible for intention formation due to intractability reasons. We also assume these approaches will be unsuitable for maintenance planing under the same justification, and will not directly consider their applicability within *formPlan* – particularly as capability confidence estimation does not require an *exact* value (as might be needed for such approaches, such as to define the transition functions required for MDP policy formation), but an indicative one.

Finally, determinization or pseudo-probabilistic approaches may be used to incorporate confidence information alongside use of classical planning methods. This would likely entail additional requirements upon the Capability model to provide access to the information supporting confidence estimation – for example, to define deterministic operators with preconditions constrained to ensure (maintenance policy defined) confidence thresholds are met, or to associate confidence-based metrics with various preconditions to support a PAC-PLAN (Jiménez *et al.* [2006a]) style pseudo-probabilistic approach[4]. As we are concerned with specification and usage of capability information for maintenance reasoning – i.e. for performing introspection to anticipate threats and generate maintenance task objects – we will not define a general approach for providing such additional information from Capability objects, and instead simply note addi-

[3] i.e. meaning that if a composite capability maps to n plans, then n methods will be generated in M to correspond to that capability.

[4] These approaches were employed for our experimental evaluation, described in Chapter 11.1.2.1.

tional implementation time decisions regarding the capability model may be driven by the semantics of *formPlan*, and any requirements of the *Problem Converter* therein.

9.4.3 Acceptable Plan Criteria

Whilst generalizing the plan generation method allows greater flexibility in practical implementation, it also requires the agent to determine whether generated 'plans are acceptable under the confidence constraints guiding maintenance. We refer to the maintenance plan generated for an *mt* as p_{maint}; the decision whether to accept p_{maintt} is performed using *acceptPlan* (Algorithm 17), which compares estimated specific confidence of p_{maint} against the threshold (*Th*) field within the maintenance policy (ϕ_{mt}) mapped to $mt.a$. This ensures $plan_i$ is not modified unless p_{maint} offers a confidence improvement upon $mt.a$. If *mt* is of *effects_for_pre* type – at immediate, near certain, risk of preconditions failure – the confidence threshold is relaxed; we treat a low probability of success as preferable to *zero* probability.

Algorithm 17: The *acceptPlan* method

Data: p – a plan
B – the execution context of the first activity in p
mt_a – a maintenance task for some a, being addressed by p
Result: **boolean** – true if confidence is sufficient for ps insertion

$conf_p \leftarrow \text{confidence}(p, B_p)$;
if $conf_p > mt_a.\phi_a$ **then**
    return true;
else if $(conf_p > 0)$ & $(mt_a.type = effects\_for\_pre)$ **then**
    return true;
else
    return false;

Worst case complexity for *acceptPlan* derives directly from worst case confidence estimation, itself depending on the specific confidence estimation implementation (Section 8.2.6). We generalize this as requiring a full traversal of an AND-OR tree, giving $O(n)$ complexity where p_{maint} consists of n activities.

9.4.4 Plan Insertion

Before discussing handling of maintenance task types, we define the various approaches for *inserting* generated maintenance plans (p_{maint}) into some existing plan p. Each approach takes a maintained activity a, plan to be inserted p_{maint}, and plan (being maintained) p as arguments, and produces the modified p as an output. We use a hierarchical plan as an example (Figure 9.2). Consideration of hierarchies allows theses approaches to be applied for 'flat' (i.e. single-level hierarchy) or continual plans (i.e. partially decomposed hierarchies).

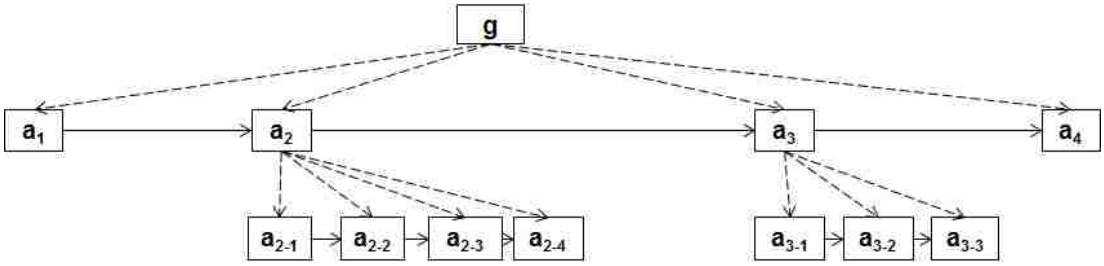


Figure 9.2: An abstracted example of a hierarchical plan.

Using *insertBefore*(a, p_{maint}, p) modifies the subplan containing a by prepending p_{maint} to a – such that a_{p1} follows a's predecessor, with a following a_{pn}, ensuring p executes immediately before a. If a is the first activity in that sub-plan, then a_{p1} instead becomes the first activity.

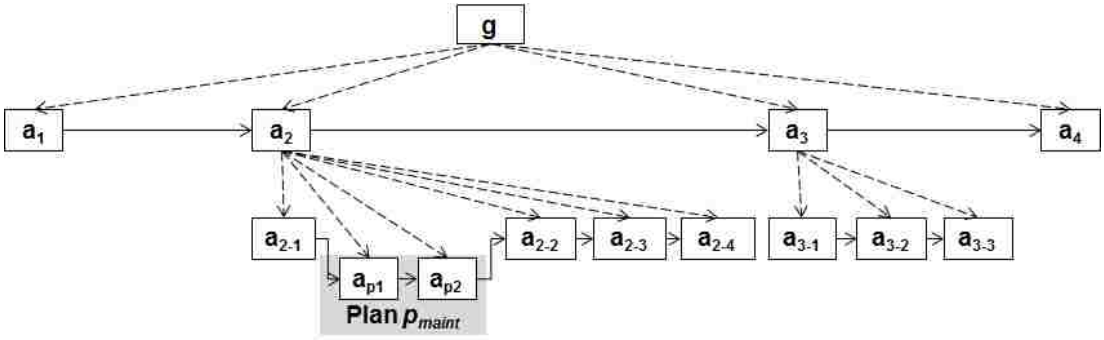


Figure 9.3: Insertion of $p_{maint} = \{a_{p1}, a_{p2}\}$ as the predecessor to a_{2-2}

Intended for effects maintenance, *replaceInPlan*(a, p_{maint}, p) substitutes p_{maint} for a (Figure 9.4): a is removed from p, as it is no longer required.

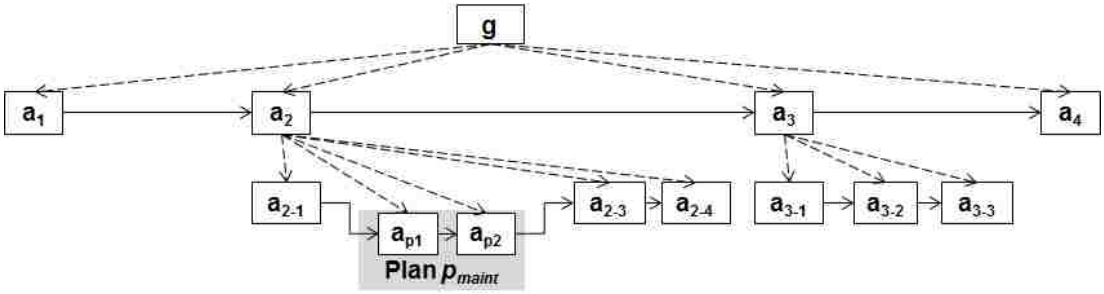


Figure 9.4: Insertion of p_{maint} in substitution for a_{2-2}

Using *replaceFromActivity*(a, p_{maint}, p) (Figure 9.5) expands insertion to cover to the end of the subplan from a inclusive. Consequently, a and *all* successors in the relevant sub-plan of p are removed and replaced with p_{maint}; a_{pn} becomes the last activity. Activities following the modified sub-plan at parent plan levels, or in subsequently executed sub-plans (i.e. decomposing other sub-goals), are not altered.

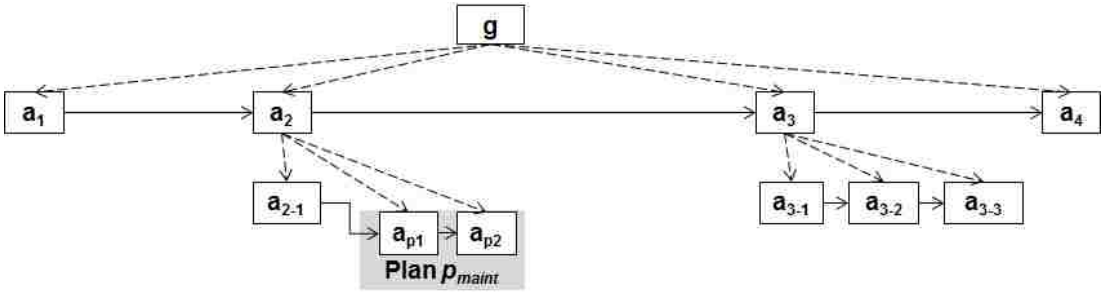


Figure 9.5: Replacement of a_{2-2} by p_{maint}, inclusive of following subplan activities

Finally, *insertNewRefinement*(a, p_{maint}, p) (Figure 9.6) requires a to correspond to a composite capability (i.e. represent a sub/goal to be achieved), and inserts p_{maint} as it's refinement – replacing any decompositional plan for a. If a corresponds to $goal_i$, this equates to complete replanning of. This is comparable to a re-refinement in HTN plan repair – i.e. as in *Unrefinement Planning* (Krogt and Weerdt [2005]).

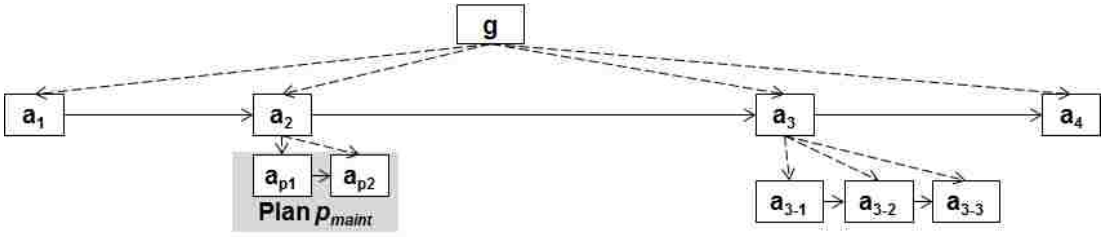


Figure 9.6: Insertion of p_{maint} to refine the composite activity a_2.

9.5 Preconditions Task Handling

Successful preconditions task handling inserts a maintenance plan that re-establishes (anticipated violated) preconditions for $mt.a$. The *neck point* (Drabble *et al.* [1997]) for inserting the maintenance plan is immediately preceding $mt.a$; if p is hierarchical, this will be within the relevant subplan. Algorithm 18 details preconditions handling, where the maintenance planning problem forms the goal from capability preconditions.

Algorithm 18: The *handlePreconditionsTask* function

Data: mt – a maintenance task
p – the plan containing $mt.a$
Result: **true** if a plan was found and inserted
$c_a \leftarrow$ getCapability($mt.a$);
$\phi_a \leftarrow$ getPolicy($mt.a$);
if $\phi_a.DC \subseteq mt.B_a$ **then**
  return false;
$pp_{mt} \leftarrow \{$formOperators($mt.B_a, \phi_a.Threshold$), $mt.B_a$, c_a.pre($mt.a$)$\}$;
$p_{mt} \leftarrow$ formPlan(pp_m);
if p_{mt} *found & acceptPlan(*p_{mt}, $mt.B_a$, $\phi_a.Threshold$*)* **then**
  insertBefore($mt.a$, p_{mt}, p);
  return true;
else
  return false;

As only one maintenance task is provided as an argument, and no iteration is performed, *handlePreconditionsTask* will terminate provided the plan formation, confidence evaluation and insertion methods do so. General complexity can be given as $O(n) = O(n_{accept}) + O(n_{formPlan}) + O(n_{insertBefore})$ – i.e. defined by sequential performance of plan formation, plan acceptance and plan insertion operations. The dominant factor is likely to be $O(n_{formPlan})$ in most cases, due to the inherent complexity of runtime planning; although this itself depends upon the implementation of *formPlan*. This helps justify decoupling planning from the core CAMP-BDI algorithms, as the implementation of *formPlan* can be altered if complexity becomes an issue.

9.6 Effects Task Handling

An *effects* (or *effects_for_pre*) maintenance task (*mt*) is handled by replacing $mt.a$ with a (sufficient confidence) plan achieving the same *effects*. As it will not always be possible or desirable to substitute *only* $mt.a$ activity, *handleEffectsTask* (Algorithm 19) uses iterative abstraction, similar in concept to HTN repair – with the intent being to minimise the scope of eventual modifications and restrict disruption to $plan_i$ (and associated obligation or dependency contracts).

Assuming a hierarchical plan, effects maintenance planning gradually increases the scope of planning, terminating when an acceptable plan is found *or* no further scope expansion is possible (total replanning has been attempted):

1 The algorithm first attempts to directly replace activity a_{mt} (where $a_{mt} = mt_a.a$, $a_{mt} \in p_{sub}$ and p_{sub} is a subplan within a hierarchical $plan_i$); the resultant p_{maint} achieves the same (capability defined) effects as a_{mt}. This is performed using *trySubstitute* (Algorithm 20); if a_{mt} corresponds to a composite capability p_{maint} will be inserted to refine a_{mt} – otherwise, p_{maint} replaces a_{mt}. If drop conditions (defined in $\phi_a.DC$, where ϕ_a is the maintenance policy associated with a_{mt}) are met, planning is treated as intractable and *trySubstitute* immediately returns.

   Direct substitution risks iteratively increasing the length of p_{sub} and (by extension) $plan_i$, as a_{mt} will be replaced with a plan of *at least* a single activity. This risks increasing complexity from preserving the remainder of a maintained p_{sub}, even if those preserved activities do not *directly* contribute to the goal.

   Existing dependencies suggest external capabilities may be necessary to achieve the goal of p_{sub}. Changes since plan formation may render it uncertain whether external capabilities are still available for contract formation, even if prior dependencies had been formed upon them; the potential obligants may have experienced situational changes to prevent acceptance of new dependency requests.

Algorithm 19: The *handleEffectsTask* function

Data: mt – a maintenance task

i – An intention where $mt.a \in plan_i$

Result: **true** if a plan was found and inserted into $plan_i$

$a_{mt} \leftarrow mt.a$;

$B_{mt} \leftarrow mt.B_a$;

$C_{a_{mt}} \leftarrow$ getCapability(a_{mt});

dependenciesFollow $\leftarrow$(true iff $\exists\ a_{succ} \in p_{sub}$ where $a_{succ} \in\ Dp$ & a_{succ} is to be executed after a_{mt});

if *(dependenciesFollow* $\|(C_{a_{mt}}.type$=*composite)* **then**

  substituted $\leftarrow$ trySubstitute(mt, $goal_i$);

  if *substituted* **then**

    return true;

if $plan_i$ *is hierarchical* **then**

  Plan $p_{sub} \leftarrow$ subplan of $plan_i$ containing a_{mt};

else

  $p_{sub} \leftarrow plan_i$;

if *(a_{mt} is not the first activity* $\in p_{sub}$*) &*

($\exists a_{pre} \in p_{sub}$ where a_{pre} precedes a_{mt} & $a_{pre} \in Dp$) **then**

  inserted $\leftarrow$ tryReplaceInclusive(mt_a, p_{sub}, $goal_i$);

  if *inserted* **then**

    return true;

while $a_{mt} \neq goal_i$ **do**

  $a_{mt} \leftarrow$ goal activity performed by p_{sub};

  reRefined $\leftarrow$ recurseEffectsUp(a_{mt}, $plan_i$);

  if *reRefined* **then**

    return true;

return false;

Algorithm 20: The *trySubstitute* function

Data: mt – a maintenance task of effects type p – A plan containing $mt.a$

Result: **true** if a plan was found and substituted for $mt.a$ in p

$c_a \leftarrow$ getCapability($mt.a$);

$\phi_a \leftarrow$ getPolicy($mt.a$);

if $\phi_a.DC \subseteq mt.B_a$ **then**

  return false;

$pp_{mt} \leftarrow \{$formOperators($mt.B_a$, $\phi_a.Threshold$), $mt.B_a$, c_a.eff($mt.a$)$\}$;

$p_{maint} \leftarrow$ formPlan(pp_{mt});

if p_{maint} *found & acceptPlan*(p_{maint}, $mt.B_a$, $\phi_a.Threshold$) **then**

  if $c_a.type$=*composite* **then**

    insertNewRefinement($mt.a$, p_{maint}, p);

  else

    replaceInPlan($mt.a$, p_{maint}, p);

  return true;

else

  return false;

As a result, we reason it preferable to (attempt to) preserve existing dependencies, despite the potential for additional plan complexity. Substitution is therefore employed for primitive activity tasks *only* where some successor(s) of a_{mt} in p_{sub} are associated with an existing dependency contract, to avoid that external capability being required by, yet unavailable to, later maintenance planning. Composite tasks use re-refinement to account for where a task consolidates multiple sub-plan tasks *or* uncertainty exists over currently unrefined goals (i.e. for continual planning).

2 If a_{mt} is not substituted, and there are dependencies or goal contributing activities *preceding a* in p_{sub}, the scope increases to consider replacement of p_{sub} from a_{mt} inclusive (*tryReplaceInclusive*; Algorithm 21). If successful, a new suffix is set for a_{mt}'s predecessor by using *replaceFromActivity* to insert p_{maint} – equating to *partial* re-refinement of p_{sub}, where p_{maint} achieves the same goal/performs the same composite activity as p_{sub}. This step is skipped if a_{mt} is first in p_{sub},

as the following stage of scope expansion is equivalent. The utilized maintenance policy ϕ_a is that associated with the goal of p_{sub}; the planning problem is regarded as intractable (and planning skipped) if $\phi_a.DC$ holds in $mt.B_a$.

3 If the algorithm cannot restrict modifications to a subset of p_{sub}, it will attempt to re-*form* it – i.e. re-refine the goal (composite activity performed through p_{sub}). If an acceptable p_{maint} is found, *insertNewRefinement* replaces the existing refinement subplan p_{sub} with it (including any 'child' subplans of p_{sub}). If $plan_i$ is *flat*, $p_{sub} = plan_i$ – entailing complete replanning. For both this and the following case, the policy mapped to p_{sub}'s goal defines the drop conditions (through the *DC* field) used to avoid attempting intractable planning.

4 If p_{sub} cannot be reformed, the algorithm sets p_{sub} as the *parent* subplan – the algorithm uses recursion to abstract up the hierarchy, attempting to re-decompose successively more abstract goals until either an acceptable plan is found *or* an unsuccessful attempt made to re-decompose $goal_i$ (i.e. no further abstraction is possible).

Algorithm 21: The *tryReplaceInclusive* function

Data: mt – an effects maintenance task
p_{sub} – A subplan containing $mt.a$
$plan_i$ – An intended plan where p_{sub} represents a subplan part
Result: **true** if a plan was found and inserted into $plan_i$
$a_p \leftarrow$ goal achieved/activity performed by p_{sub};
$c_p \leftarrow$ getCapability(a_p);
$\phi_a \leftarrow$ getPolicy(a_p);
if $\phi_a.DC \subseteq mt.B_a$ **then**
    return false;
$pp_{mt} \leftarrow \{$formOperators($mt.B_a, \phi_a.Threshold$), $mt.B_a$, c_p.eff(a_p)$\}$;
$p_{mt} \leftarrow$ formPlan(pp_{mt});
if p_{mt} *found & acceptPlan(*p_{mt}, $mt.B_a$, $\phi_a.Threshold$*)* **then**
    replaceFromActivity($mt.a$, p_m, $plan_i$);
    return true;
else
    return false;

Effects maintenance attempts to minimize the scope of maintenance planning to

reduce distruption to $plan_i$. This is similar to HTN plan repair, and aims to mirror the associated stability benefits over replanning (Fox *et al.* [2006]) relevant to distributed planning (Komenda *et al.* [2012]). Where all scope levels are applicable, *handleEffectsTask* will abstract n levels 'upwards' in the plan hierarchy until planning is attempted for $goal_i$ – terminating when either a subfunction inserts a plan and returns true, or where *recurseEffectsTaskUp* (Algorithm 22) attempts to replan $goal_i$ and returns false. The latter assumes each recursion possible represents an abstractive step 'up' a plan hierarchy towards the root $goal_i$.

Algorithm 22: The *recurseEffectsTaskUp* function

Data: a – an activity, representing a sub-goal to be re-refined
$plan_i$ – the plan containing a
Result: **true** if a new refinement plan for a was inserted into $plan_i$
$c_a \leftarrow$ getCapability(a);
$\phi_a \leftarrow$ getPolicy(a);
$B_a \leftarrow$ estimated execution context for a_{mt};
if $\phi_a.DC \notin B_a$ **then**
  $p_{sub} \leftarrow$ (sub)plan containing a;
  $pp \leftarrow \{$formOperators($B_a, \phi_a.Threshold$), B_a, c_a.eff(a)$\}$;
  $p_{mt} \leftarrow$ formPlan(pp);
  if p_{mt} *found & acceptPlan(*p_{mt}, B_a, $\phi_a.Threshold$*)* **then**
    insertNewRefinement(a, p_{mt}, $plan_i$);
    return true;
if $a \neq goal_i$ **then**
  $a \leftarrow$ (parent)goal achieved/activity performed by p_{sub};
  return recurseEffectsTaskUp(a, $plan_i$);
else
  return false;

Each sub-function call (i.e. at each scope level) has O($c+p$) complexity, where c represents the complexity of confidence estimation for generated plans, and p the worst case planning complexity (these are expected to be dominant factors in the individual sub-function's complexity). For a plan with n hierarchical levels, worst case complexity for effects handling is $O((n+2)(c+p))$; n is the number of plan levels (hierarchical

depth), with two additional planning calls are made to (attempt to) substitute or replace inclusive the maintained activity. This can be simplified to $O(n)$ if a c and p are treated as constants (i.e. worst case cost).

The potential for multiple planning operations, particularly for 'deeper' (greater n) plans, is a potential concern in terms of computational cost; our algorithm trades this cost off against minimizing disruption to the overall $plan_i$ (and to any associated distributed intention). However, if domain specifics entail plan stability is not a concern, agent plans are particularly deep (high n), or inter-agent dependencies are limited (or have low associated communications costs), it is trivial to modify *handleEffectsTask* to omit earlier, more restricted scope operations and solely call *recurseEffects* at the $goal_i$ level (i.e. reducing to O($c + p$) complexity with a total replan).

9.7 Running Example

We use our definition of the CAMP-BDI maintenance algorithm to provide an example of maintenance behaviour within the previously described *CargoWorld*. In our example scenario, *Truck1* holds an intention (i) to deliver cargo (using geography from Figure 2.6). For illustration purposes, *Truck1* has cargo selection and delivery plans (composite capabilities) normally restricted to *LogisticsHQ*. Policies have been defined to give *unload* activities hold higher precedence than those using other capabilities, and to define a threshold for *move* which requires employed roads to be in a *dry* state. We describe our examples in the context of the progression of $plan_i$, and – for sake of simplicity – omit any hypothetical interleaving of *other* intentions with $plan_i$.

Figure 9.8 shows a hierarchical $plan_i$ to deliver an item of cargo to M, where *Move*(J,D) is next to execute. The plan defines that *Truck1* will move to D, load the specific cargo *Cargo1* there, move to M and finally unload *Cargo1* (i.e. achieve *deliverCargo*(M)).

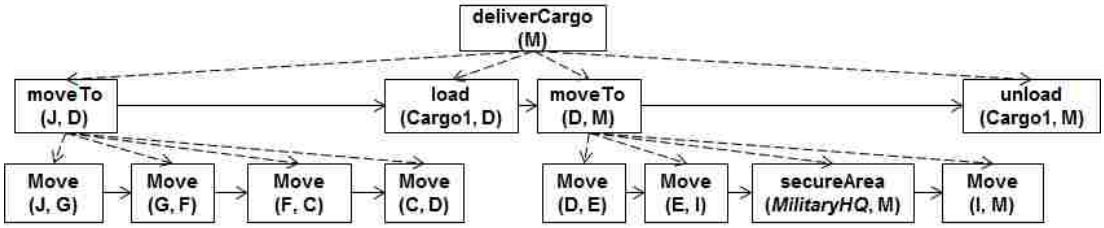


Figure 9.7: Initial $plan_i$ for *deliverCargo*(M).

9.7.1 Preconditions Maintenance Task handling

In this first example, a *dangerZone* has emerged at I. Before executing $Move(J, G)$, the *maintain* function is called for i ($goal_i$ =$deliverCargo(M)$). The following sequence of behaviour activities result;

1 *formAgenda* iterates through leaf tasks and identifies that preconditions of $Move(E,I)$ do not hold (Figure 9.8). As this activity is followed by a dependency (for $secureArea(MilitaryHQ,M)$ preconditions task mt_{pre} ($mt_{pre}.a$=$Move(E,I)$) is generated; this is the sole task within the returned agenda.

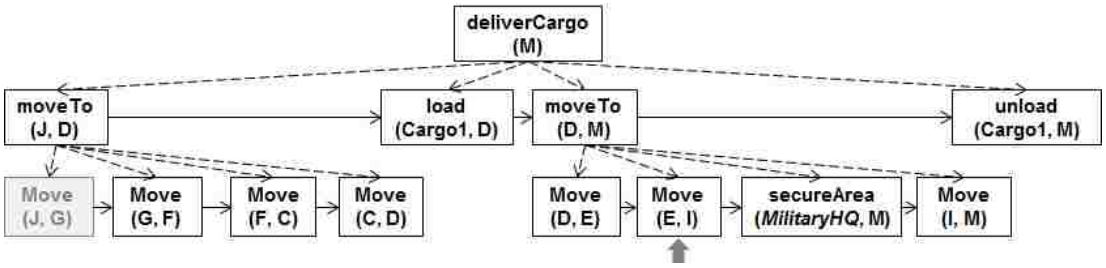


Figure 9.8: Initial state of $plan_i$, where the next activity to execute is $Move(J,G)$ and preconditions of $Move(E,I)$ are violated (threatened activities are indicated by gray arrows underneath).

2 *handlePreconditionsTask* is invoked for mt_{pre}. A plan $p_{maint_{pre}}$ is found, which restores the states required by that *Move* activities' preconditions through a single $secureArea(MilitaryHQ, I)$ activity.

3 Maintenance completes by inserting $p_{maint_{pre}}$ before the threatened $Move(E,I)$ (Figure 9.9); *handlePreconditionsTask* returns true, allowing *maintain* to exit.

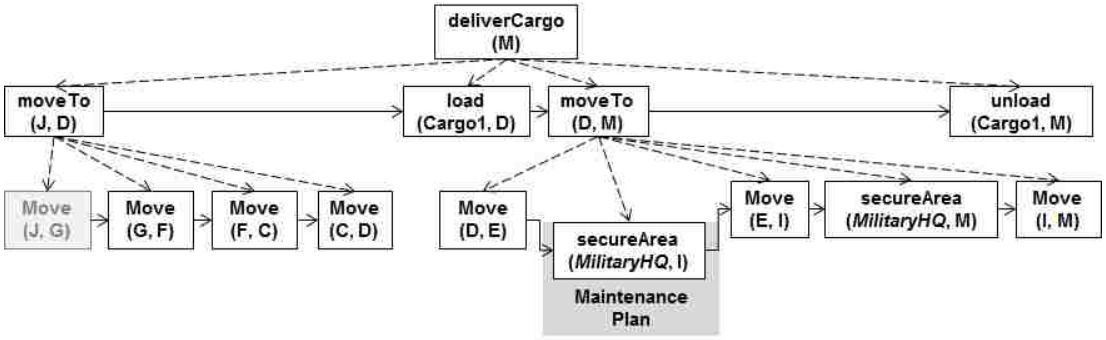


Figure 9.9: Insertion of a maintenance plan prior to the threatened $Move(E, I)$.

9.7.2 Effects Maintenance Task handling

During execution of $Move(J,G)$ (following the prior maintenance), junction C becomes dangerous. Upon the next reasoning cycle (i.e. selection of i), the next activity of $plan_i$ becomes $Move(G,F)$ with the *Truck*'s B set accordingly updated. The following behaviour arises from $maintain(i)$;

1 *formAgenda* identifies preconditions are violated for $Move(F,C)$ due to the dangerZone at C (Figure 9.10). As this activity is not followed by any dependencies within the subplan for $moveTo(J,D)$ and does not contribute any goal states, an *effects* task – mt_{eff-1} with $mt_{eff-1}.a$=$Move(J,G)$ – is added to the agenda (eventually returned containing mt_{eff-1} only).

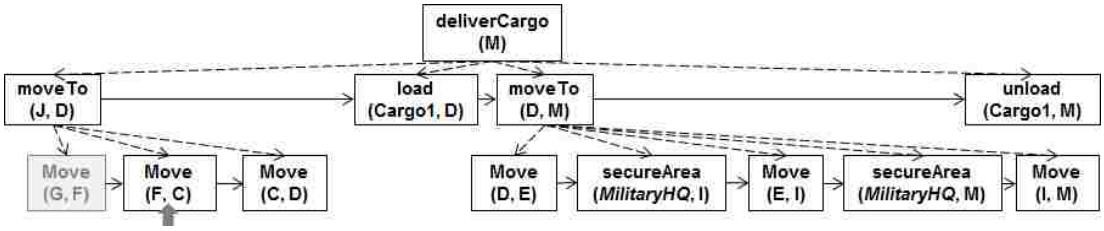


Figure 9.10: The $plan_i$ where the next activity is *move*(G, F), and *move*(F, C) is threatened.

2 *handleEffectsTask* is invoked for mt_{eff-1}. As $Move(F, C)$ is neither composite, nor followed or preceded by a dependency, *trySubstitute* and *tryReplaceInclusive* are skipped, with *recurseEffects* employed for the parent goal(s).

3 The maintenance planning goal is set as the parent goal for the subplan – $MoveTo(J, D)$ – with initial state as the execution context for the first activity in the relevant subplan (in this case, also the next activity to be executed in $plan_i$).

4 The resultant generated maintenance plan $p_{maint_{eff-1}}$ ($p_{maint_{eff-1}}$=$\{move(G,J), move(J,K), move(K,H), move(H,C), move(C,D)\}$), is inserted to re-refine $MoveTo(J,D)$ (Figure 9.11).

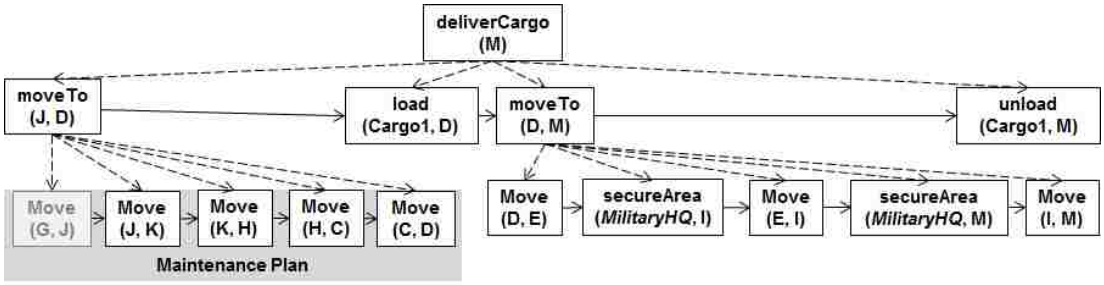


Figure 9.11: Insertion of a maintenance plan from *move*(F,C) inclusive, replacing the following activities in the subplan for *moveTo*(J,D).

9.7.3 Effects Maintenance Task consolidation and handling

We next consider subsequent execution of $plan_i$ (with $Move(J,K)$ now the next activity), where roads $D \rightarrow E$ and $E \rightarrow I$ both become slippery:

1 *formAgenda* estimates the confidence for $Move(D,E)$ and $Move(E,I)$ activities. Due to slippery road conditions, both have confidence below the *Th* field defined in their associated policies. Effects type tasks are added into the agenda; mt_{eff-2a} ($mt_{eff-2a}.a$= $Move(D,E)$) and mt_{eff-2b} ($mt_{eff-2b}.a$= $Move(E,I)$).

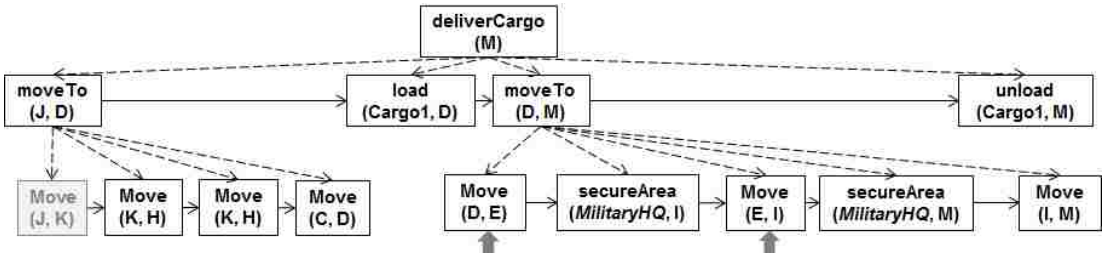


Figure 9.12: Multiple threatened activities in $plan_i$; the current activity is *Move*(J, K).

2 The *consolidate* function, called when *formAgenda* returns for the subplan for *moveTo*(D,M) consolidates the agenda $\{mt_{eff-2a}, mt_{eff-2a}\}$ into a single effects task $mt_{eff-2-consolidated}$ with $mt_{eff-2-consolidated}.a$=*moveTo*$(D,M)$.

3 *handleEffectsTask* is called for $mt_{eff-2-consolidated}$; as this task concerns a composite activity, *trySubstitute* is employed.

4 The new plan for *moveTo*(D,M), *mt* (where *mt*=$\{Move(D,C), Move(C,H), Move(H,I), Move(I,M)\}$) is inserted as a (re)refinement (Figure 9.13). This also removes the dependency for *secureArea* which lay within the prior (now re-refined) subplan – simplifying the refinement of *moveTo*(D,M) but with some (limited) disruption to obligant(s) for now-unecessary dependencies.

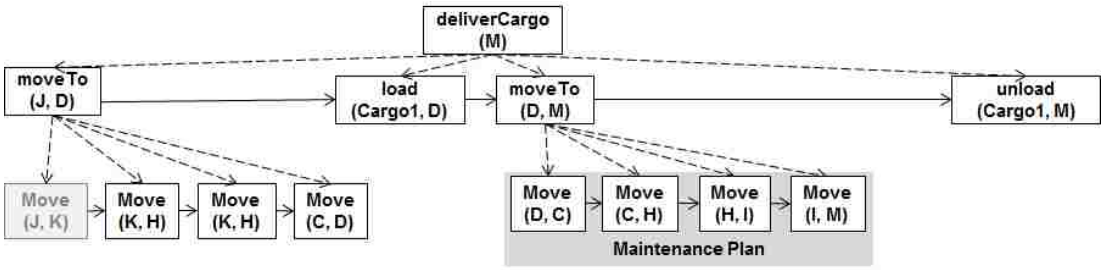


Figure 9.13: Re-refinement of *moveTo*(D,M) by handling a consolidated task.

9.7.4 Iterative Scope expansion in Maintenance

Our final example shows wider scope modification of $plan_i$, where multiple precondition tasks arise from exogenous change destroying *Cargo1*. In this case, execution has progressed to *Move*(C,D):

1 The *formAgenda* item identifies that preconditions of the *load* and *unload* activities in $plan_i$ have been violated, as *Cargo1* no longer exists (Figure 9.14).

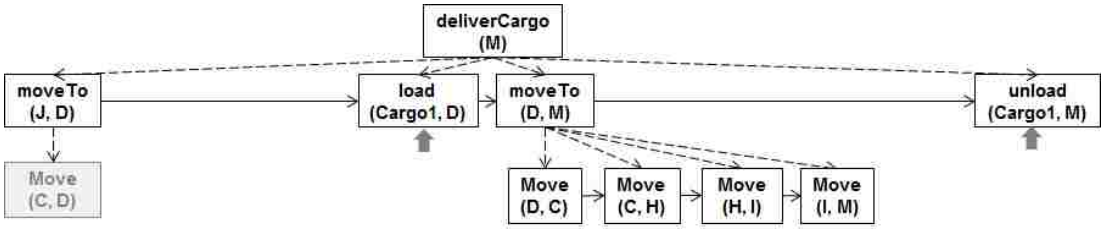


Figure 9.14: $plan_i$ where destruction of *Cargo1* threatens *load* and *unload* activities required for the *deliverCargo*(M) goal.

2 As *load* does not directly contribute a goal state and is to execute next, a corresponding *effects_for_pre* task is added to the agenda; $mt_{effects\_for\_pre}$.*a*=*load*(*Cargo1*,*D*).

3 As *unload* contributes a goal state (by depositing a cargo object at M), a corresponding *preconditions* task is also added (mt_{pre}.*a*=*unload*(*Cargo1*,*M*)); as the relevant maintenance policies give *unload* tasks higher priority, mt_{pre} take precedence in the agenda[5].

4 The resultant ordered agenda $\{$*unload*(*Cargo1*,*M*), *load*(*Cargo1*,*D*)$\}$ is returned by *formAgenda* for $plan_i$

5 *handleTask* attempts to address the *unload* preconditions task; *handlePreconditionsTask* fails as the absence of *Cargo1* renders it impossible, and returns false.

[5]For sake of this example, these tasks will *not* be consolidated to a single *effects_for_pre* task for *deliveryCargo*(M).

5 *handleTask* responds to this failure by generating an *effects* task for *unload*(*Cargo1*), and subsequently calls *handleEffectsTask* for the resultant mt_{eff} (mt_{eff}.*a*=*unload*(*Cargo1*)).

6 As *unload*(*Cargo1*) is neither preceded nor followed by a dependency, neither *trySubstitute* or *tryReplaceInclusive* are attempted. Instead *recurseEffects* attempts to re-refine $goal_i$ (i.e. the goal met by the subplan containing $mt_{eff}.a$).

7 A maintenance plan is found and inserted to re-refine $goal_i$, resulting in full reformation of $plan_i$ (Figure 9.15).

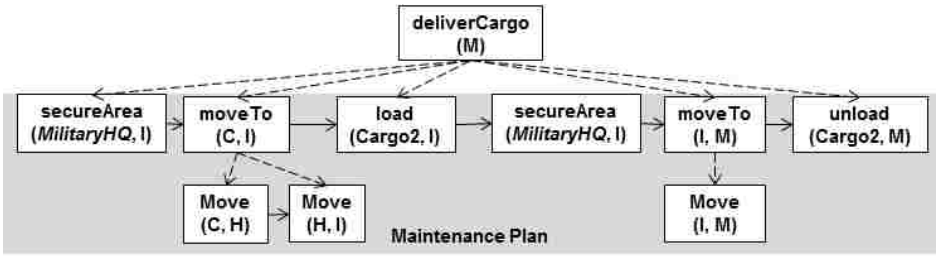


Figure 9.15: Re-refinement of *deliverCargo*(*M*) to handle the consolidated task.

9.8 Summary

This chapter contributes algorithms for the proactive modification of intended plans as part of a modified BDI reasoning cycle – referred to as *maintenance* – and which define the key behaviour of CAMP-BDI agents. The following chapter expands upon individual agent behaviour to use structured messaging, combined with contract information, for decentralized maintenance of *distributed* intended plans.

We detailed a two step process; a diagnostic stage which forms an agenda of *maintenance tasks* (characterising threatened activities in a plan), followed by handling though (attempted) appropriate proactive plan modificationn. We categorize maintenance tasks as *preconditions* or *effects* type (including an *effects_for_pre* subtype); this guides initial goal specification for, and later insertion of, maintenance plans to address identified threats. Although we primarily refer to hierarchical plans, our algorithms equally apply to 'flat' structures or partially formed hierarchies, with composite capabilities enabling reasoning over unrefined subgoals/abstract activities in the latter.

Our algorithms utilize the supporting architecture presented in Chapter 8:

- **Capabilities** allow introspective reasoning about intended plans, including quantitative evaluation of intended activities.

- **Contracts**, detailed in Chapter 10, provide information regarding delegated activity to allow the same introspective reasoning as for internal capabilities – whilst offsetting information requirements to the obligant.
- **Policies** allow definition of generic algorithms, by externalizing definition of maintenance trigger, drop and threshold conditions; allowing domain specific definition and run-time modification.

We focus upon maintenance following the intention selection stage of BDI reasoning, and assume agent BDI behaviour is *goal* orientated – selection of a given i is driven by the desirability of the associated $goal_i$ – and that maintenance modifications to the $plan_i$ cannot render i *less* selectable. This views *goal-driven* behaviour as a critical element of agent rationality, as expressed by Kinny *et al.* [1992] and determined by Wooldridge [2002] as a key requirement for *intelligent* agency.

Our maintenance algorithm adopts an approach similar to HTN plan repair, to minimize disruption to the intended plan and associated inter-agent dependency relationships. This does risk greater planning costs, particularly when planning scope is iterated 'up' a hierarchical plan. The consolidation (aggregation) of maintenance tasks mitigates against multiple planning steps where multiple tasks would otherwise be handled over multiple reasoning cycles. Effects maintenance uses conditional cases to relax the planning problem where viable, to avoid more constrained scope maintenance planning. This attempts to mitigate against the worst case complexity of *maintain*, where $n+3$ planning calls are required for a n level hierarchical plan; the additional three calls accounting for (failed) preconditions maintenance planning, plus direct substitution (*replaceInPlan*) and inclusive substitution (*replaceFromActivity*) effects maintenance steps.

In general, it is *expected* that pre-emptive approaches face additional costs; as any proactive system has to trade-off unnecessary mitigation behaviour through over-sensitivity (false positive failure predictions) against risking false positive predicted success. We also hypothesized that – in certain domains – the impact and cost of *failure* in terms of hindering reactive recovery may itself justify an additional *preventative* cost. Our evaluation, therefore, must consider both planning costs of our approach and the value of preventing failure and associated debilitative consequences. It is necessary to consider the value of *goals* themselves – the negative impact of goal failure may also justify additional expenditure, particularly where said failure risks damage to physical

resources or human lives.

We also provide policies to aid in *proportioning* this additional cost – if a particular goal has more severe failure consequences, its associated policy (matched to the relevant capability) can specify a lower threshold or additional trigger conditions to increase sensitivity of maintenance. Conversely, goals with lower failure costs can be allocated higher confidence thresholds or explicit drop conditions to reduce maintenance frequency – assuming their failure is permissible or that alternative reactive handling is available.

Chapter 10

Distributed Maintenance

This chapter extends individual agent maintenance behaviour to the distributed case, reflecting the importance of distributed planning in MASs. Obligation maintenance processes are defined – specifically, conditions for obligants to invoke their *maintain* method, and the behaviour required where a maintained intention is for a delegated activity. We also describe how dependants respond to notifications of obligant maintenance and conditions for adopting maintenance locally.

10.1 Introduction

The use of a MAS approach is often motivated by the distribution of knowledge and capabilities in a given domain rendering centralized approaches infeasible (Sycara [1998]). For similar reasons, we adopt a decentralized approach towards distributed maintenance by extending individual agent maintenance. Use of structured local behaviour for designing decentralized, distributed behaviour is well-established – Partial Global Planning (Durfee and Lesser [1991]), for example, is founded on the principle of "*co-ordination arising through local reasoning*".

10.1.1 Approach

Our distributed maintenance design aims to replicate *local* maintenance of hierarchical plans within the context of a distributed plan being executed by a decompositional agent team. Achievement of a delegated activity through an obligant's (intended) plan (i.e. achievement of a goal/subgoal activity through a plan/subplan) is here analogous to refinement of an HTN task. Mapping between HTN concepts and distributed agent

activity has been recognised by prior research; the planner component of RETSINA agents (Paolucci *et al.* [2000]) utilizes an HTN approach following a rationale of similarities between task decomposition and sub-task delegation to others, as does work on multiagent planning by Obst [2006]. de Silva and Padgham [2004] note HTN planners and BDI agents employ similar decompositional tree structures for (respectively) task refinement and goal-plan identification and observed, where both allow backtracking – in BDI after activity failure (including unexpected post-effect states) and in HTN when a pursued *solution* fails. Vincent *et al.* [2000] also observe that the TÆMS (Task Analysis, Environment Modelling, Simulation) language used to model (multi)agent activity extends HTN approaches, and shares similar task decomposition notions.

We conceptualized HTN plan repair as a process which determines if a (non-primitive) plan task should be re-refined, based upon the tasks in the current refinement. This entails agents adopt maintenance responsibility (with a delegated activity equivalent to a non-primitive task refined by the obligant's plan) when both;

- A delegated activity is at risk of failure, from either violation of preconditions or the anticipated level of quality[1] falling below a set (by contract maintenance policy) threshold.
- The obligant(s) for that activity have been unable to *restore* required confidence in their obligation to that dependant; i.e. meaning loss of mutual belief in that delegated activity being performed at the required level of quality.

10.1.2 Synchronization and Communication Requirements

Distributed maintenance requires agents to both share information regarding delegated activity *and* indicate whether they themselves attempted maintenance, in order to synchronize escalation of responsibility to (or adoption of responsibility by) a dependant. Distributed maintenance should start at the obligant level, with responsibility (and consequent maintenance activity) escalating to dependants only as far as is required to restore confidence. This limits disruption to the distributed plan to the smallest, most specific subset of the executing hierarchical agent team, and ensures obligants are able to use their local, more specific knowledge regarding to maintain their obligants *first*. Dependants and obligants must synchronize behaviour – but we also wish to reduce communication requirements when doing so. Rather than having (in effect) obligants

[1] Expressed, and subsequently referred to, as their estimated confidence.

directly inform their dependant when to attempt maintenance, the dependant should make that decision (to preserve autonomy); agents should similarly complete maintenance locally, inform any dependant of changes, and subsequently continue with any ongoing execution through their own autonomous reasoning.

It is possible an obligant may begin execution of an intention, only for the dependant to cancel that obligation following maintenance changes at it's own level. Whilst this risks obligants expending unnecessary effort in executing intentions subsequently aborted by dependant maintenance, it prevents forced waiting upon direct and indirect dependant maintenance – an unnecessary delay if no dependant maintenance changes are performed, or if such changes occur (temporally) later in the plan and would not impact the waiting obligant (and obligation). The autonomous, local adoption of maintenance responsibility instead avoids such lengthy synchronization. We assume that, if dependants do cancel a dependency following obligant maintenance, the 'wasted' expenditure executing an obligation between the obligant communicating their post maintenance update *and* resultant dependant cancellation would not be excessive.

10.1.3 Reasoning Cycle Methods

The previous chapter described the CAMP-BDI agent reasoning cycle, focusing upon the *maintain* function. Algorithm 23 repeats this reasoning cycle algorithm, highlighting parts associated with *distributed* maintenance:

- *extractObligationMaintainedMessages* handles post-maintenance *obligationMaintained* messages from obligants (*eventQueue* includes message receipt events).
- *formAndUpdateContracts* performs dependency contract formation for (to be) delegated activities within $plan_i$, and updates any existing associated dependency or obligation (for $goal_i$) contracts.
- *maintainDependencies* triggers maintenance in response to receipt of *obligationMaintained* messages – i.e. where obligants have maintained an obligation $plan_i$, with potential impact upon the dependant's $plan_i$.
- *maintainObligations* performs maintenance of suspended intentions associated with obligation contracts.

Algorithm 23: The CAMP-BDI reasoning cycle, with steps relevant to distributed maintenance highlighted as black

```
initializeState();
while agent is alive do
    msg_Ob ← extractObligationMaintainedMessages(eventQueue);
    B ← updateBeliefs(eventQueue, B);
    D ← optionGenerator(eventQueue, I, B);
    I ← deliberate(D, I, B);
    i ← updateIntentions(D, I, B);
    if i ≠ null & i not waiting on a dependency to complete then
        maintain(i, B);
        formAndUpdateContracts(i, B);
        execute();
    else if (Dps ≠ ∅) & ( msg_Ob ≠ ∅) then
        maintainDependencies( msg_Ob);
    else if Obs ≠ ∅ then
        maintainObligations(Obs);
    getNewExternalEvents();
    I ← dropSuccessfulAttitudes();
    I ← dropImpossibleAttitudes();
    I ← postIntentionStatus();
```

10.2 Information sources in Distributed Maintenance

We will now overview use of the CAMP-BDI supporting architecture (Chapter 8) within distributed maintenance.

10.2.1 External Capabilities

External capabilities are, from the perspective of maintenance, indistinguishable from internal capabilities in terms of their fields; capabilities are polymorphic, and effectively abstract the *sources* of their represented knowledge. Capabilities define fields which must be specified in a capability advertisement; by using this information to form external capabilities, recipients can use this information to reason over delegated

(or potentially delegated) activities without requiring underlying semantic knowledge of how that activity would be performed. Instead, only the advertiser requires semantic knowledge, as they hold sole responsibility for actually *calculating* the relevant (advertised) field values.

It will be typically more efficient in communications terms to perform advertisement updates (i.e. a 'push' approach) than require possible dependants to query for the latest capability information (i.e. 'pull'). Agents therefore will be required to update capability advertisements when changes to their B set impact confidence. Although not explicitly defined in our above reasoning cycle, this update behaviour can be incorporated into the *updateBeliefs* function – the exact semantics will depend on the specific implementation of capability advertisement and storage of capability information.

10.2.2 Dependency and Obligation Contracts

Teamwork approaches such as Joint Intentions (JI) (Levesque *et al.* [1990]) and Joint Responsibilities (an extension of JI) theory (Jennings [1992]) require mutual belief establishment between agents involved in a joint activity; our maintenance approach is influenced by the latter, which models separate commitments to goals and plans – to allow the latter to be revised if necessary to ensure belief in achieving the former. CAMP-BDI expresses these mutual belief requirements through contracts; agents (as an implicit or explicit condition of contract acceptance) will update mutual beliefs through communicating contract *updates* when necessary. Contracts both guard against contention and provide information regarding how a delegated activity is to be performed (i.e. through the causal links – CL – and external capability – EC – fields). This information is used in maintenance of both obligations and dependencies. CL provides obligants with the combined effects of activities (yet to be) executed by their dependant prior to the delegated activity; allowing estimation of the future execution context when maintaining that obligation.

Contracts allow for provision of more specific external capability information than generalized advertisement – and offering dependants more accurate knowledge for use when maintaining their dependant intention. This offsets semantic knowledge requirements to the obligant providing and updating the EC field (following exogenous changes and/or maintenance changes). Dependants could, alternatively, directly query

(poll) obligants to discover whether changes have occurred. However, this would incur additional communications costs from being performed in anticipation of *possible* change, rather than driven by definite changes known to the obligant. This would be exacerbated for composite capabilities, where a specific confidence query could lead to numerous reciprocal queries for each (potentially) delegated activity in the represented plans. Finally, this could compromise agent autonomy if requiring the handling of these queries be prioritized, to avoid maintenance across the distributed team being delayed by response waits.

Contract formation and update requirements entail additional messaging costs over ad-hoc (contract-less) activity performance requests. We argue this cost (particularly for updates) is justified through the provision of more accurate external capability information for maintenance, and consequently greater robustness. We also assume contracts are required to guard against contention, making contract *formation* (if not updating) a necessary expense regardless of robustness approach.

10.2.3 Maintenance Policies

Policies define conditions for both generation of maintenance tasks, and for the acceptance (insertion) of generated maintenance plans. Specific maintenance policies are held within contracts to ensure obligants and dependants share maintenance conditions for the delegated activity. This prevents maintenance being triggered at dependant but not obligant levels; ensuring the latter is not denied the opportunity to modify it's local $plan_i$ (if successful, with consequently reduced disruption to the overall distributed plan compared to maintenance by a dependant), due to being pre-empted by maintenance by the former.

Specification of contract maintenance policies is combined with post-maintenance messaging by obligants (Section 10.2.4.2) to synchronize adoption of maintenance responsibility up the distributed team hierarchy. An important aspect of our distributed behaviour is that obligants do not 'order' dependants to perform maintenance, but instead send updated information that allows the dependant to decide whether to maintain the intention containing the relevant dependency. Contract-set maintenance policies ensure consistency of *conditions* for forming and handling maintenance tasks; structured messaging drives when agents *employ* that policy as part of invoking *maintain*.

10.2.4 Forming and Updating Contracts

Agents are responsible for forming and updating contracts. This section describes that process, viewed as a necessary ancillary process to provide information for both maintenance and execution in general. Although we consider specific contract formation protocols as implementation specific, there is a clear relationship with the CAMP-BDI specific (in terms of requirements) update process – requiring the former be defined in detail sufficient to illustrate the latter. This section provides a highly abstracted view of the contract formation process, illustrating supporting assumptions regarding the *minimal* agent behaviour performed when a contract is formed between a dependant and obligant(s). Further activities, such as negotiation or obligant selection, lie within the abstracted elements of these descriptions and would be implementation specific.

10.2.4.1 Obligant selection and task allocation

In distributed plans, the allocation of tasks to agents (obligants) will directly impact the robustness and stability of that plan; an inappropriate obligant selection will increase the failure risk of delegated activities, and upon the distributed plan as a whole. Task allocation may be performed through direct assignment (i.e. where the dependant specifies the obligants to perform a delegated activity), or through more 'social' approaches such as combinatorial auctions[2].

Our reasoning cycle performs maintenance (i.e. calls *maintain*) after contract formation – the latter assumed to implicitly include any necessary obligant selection processes. This ordering allows maintenance to use the most accurate, up-to-date information regarding obligations and obligants.

However, we do not assume every delegated activity in the $plan_i$ considered by *maintain* will necessarily have an associated obligation contract, or even obligant(s) specified. Instead, our design considers three possible states of knowledge regarding a (to be) delegated activity in the design of the *getCapability* method (Section 8.2.4, Algorithm 2):

[2]A combinatorial auction sees agents bidding upon *combinations* of items, assigning some corresponding 'value' – for example, to perform some set of activities with some given utility, quality or cost (Hunsberger and Grosz [2000], Conitzer [2010]). The auction results in assignment of activities to agents, accepting bids such that all activities are allocated – with the aim being to ensure the best total cost (e.g. highest overall utility, or lowest cost).

- An obligant is specified and contract exists; the contract *EC* field is returned
- An obligant is specified but no contract exists; the external capability is taken from $C_{external}$ (the subset of agent capability knowledge representing received advertisements)
- No obligant is specified; the best option (highest general confidence) external capability within $C_{external}$ is used

We do not assume a specific mechanism for obligation selection, and assume the semantic details of whichever method *is* employed will be subsumed within implementation-specific contract formation protocols (for social methods such as auctions) or (for direct assignment) the planner implementation. A further simplifying assumption is that capability advertisements are updated to reflect whether the advertiser can accept obligations; i.e. the presence of a capability within $C_{external}$ indicates a dependency *can* be formed and upon which agents.

It is also assumed any non-zero confidence capability within $C_{external}$ can be included within the operator specification formed for maintenance planning, and that a willing obligant will exists if the relevant activity is used within a maintenance plan (as contract formation for a modified $plan_i$ occurs immediately after *maintain* completes). Where no capability in $C_{external}$ exists corresponding to a delegated $plan_i$ activity, that activity is regarded as having zero confidence – causing generation of a representative effects maintenance task. Where a delegated activity is due to execute next in the current $plan_i$, lacks a dependency contract *and* contract formation has previously failed (i.e. indicating a dependency cannot be formed, as opposed to dependency formation not yet having been attempted for an activity recently added by maintenance), similar logic is applied with a new maintenance task being generated for that activity – with subsequent maintenance planning employing the $C_{external}$ set, whose members reflect only obligations that *can* be formed in the current situation.

We assume the obligant selection processes are implementation specific, and leave these abstract for simplification. The following sections describe required minimal *contract formation* behaviour, illustrating assumptions and requirements regarding provision of contractual knowledge for maintenance.

10.2.4.2 Contract Formation Protocol

Our basic assumption of Contract formation is represented by a relatively simple protocol, starting with proposal of a contract by the (prospective) dependant. This initial contract defines an activity, causal link effects, and the policy associated with the dependant-activity pair. Figure 10.1 depicts the messaging sequence for successful contract formation; a proposed dependency contract must be accepted by all prospective obligants before it can be finally confirmed by the dependant. If at least one obligant rejects a contract, then the dependant will cancel the contract (Figure 10.2).

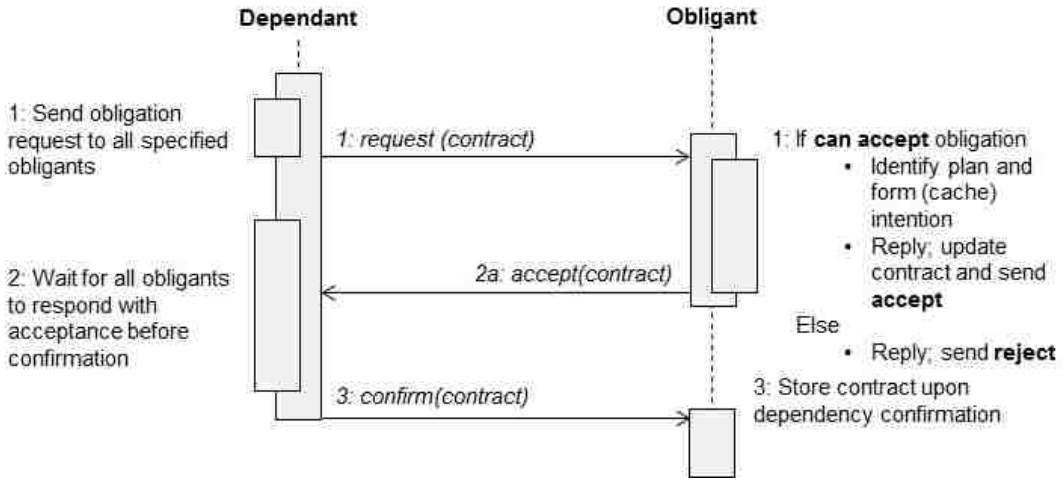


Figure 10.1: Sequence diagram of example messaging during contract formation.

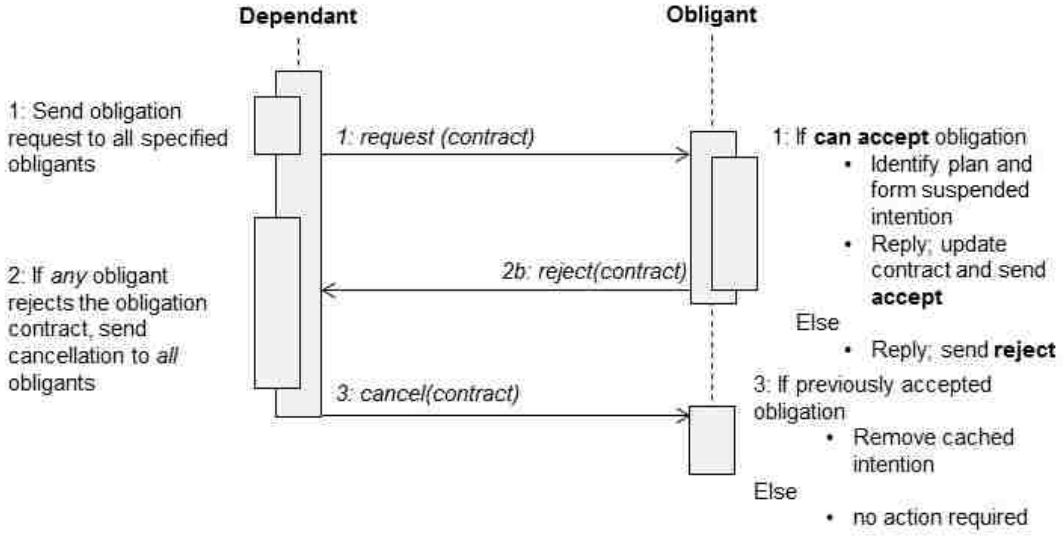


Figure 10.2: Sequence diagram of example messaging during contract formation, where the potential obligant *rejects* the obligation request.

How individual agents determine whether to accept (or potentially negotiate) a contract is implementation specific; we are solely concerned with the *outcome* of dependency formation, and leave specific contract formation semantics abstract. We assume acceptance includes specification of the contract's maintenance policy, and will require the obligant(s) hold the capabilities required to perform the delegated activity with acceptable confidence. We also assume obligants pre-emptively form intentions (*suspended* intentions) – representing a $plan_i$ identified in advance and held in memory ready to be executed upon dependant request.

Pre-emptive intention formation allows obligations to be maintained in advance of execution as part of ensuring mutual beliefs, although it is not *necessary* for CAMP-BDI. Obligants can instead use capability knowledge to reason about accepted obligations (matching activities to their internal capabilities) and perform contract updates; this would also be less memory intensive than pre-emptively storing intentions. However, pre-emptive formation allows obligants to identify the semantic detail of how they will (intend to) perform that specific delegated activity. Absence of this detailed information may risk unnecessary escalation of maintenance responsibility – with consequently greater distributed plan disruption – to dependants in circumstances where a suspended intention, if formed, could have been successfully modified.

Agents generate an effects maintenance task for any activity, due to execute, where contract formation previously failed. The resultant effects maintenance planning will use the most recent external capability knowledge, effectively forcing reconsideration of whether that (to be) delegated activity is desirable. Effects maintenance can also be triggered by changes in the contract's external capability field. For example, a potential obligant may set (and advertise) capability confidence to zero following some change in circumstance. This similarly causes a potential dependant (i.e. which requires use of that external capability, but which has not yet formed a contract with the advertiser) to perform effects maintenance and reconsider the intended use of that external capability, due to the potential obligant offering insufficient confidence.

10.2.4.3 Forming and Updating Contracts For Intentions

The *formAndUpdateContract* function (Algorithm 24) executes after *maintain* is called for some i, and both (attempts to) forms any dependency contracts required for activi-

ties within the maintained $plan_i$, and update existing associated contacts. In the case of dependency contracts for activities within $plan_i$, the *CL* field may be updated to reflect where preceding activities – and preceding effects – have been modified; this can also cover where maintenance did not modify $plan_i$ plan, but receipt of *obligant* updates changed the *EC* field (specifically, the effects) for relevant predecessor dependencies. Similarly, if $goal_i$ corresponds to an obligation contract, the *EC* field may be updated – informing the dependant if preconditions have changed, effects have been modified, or confidence differs. Again, this will include and propagate any *EC* updates received for the obligation $plan_i$'s dependencies.

Algorithm 24: The *formAndUpdateContracts* function

Data: i – An intention which may be an obligation and/or have associated dependencies; $i = \{goal_i, plan_i\}$

```
/* Update dependencies upon others; also determines and returns
   plan_i preconditions and effects                              */
```

$pre, eff \leftarrow$ updateDependencyContracts($goal_i, plan_i, \emptyset, \emptyset$);

```
/* If there is an obligation contract for goal_i and changes have
   occurred, updates and sends to the dependant                  */
```

if $\exists contract \in Obs$ *where* $contract.\alpha = goal_i$ **then**

- $B_i \leftarrow$ (Copy of) B;
- Update B_i with $contract.CL$;
- $conf_{obl} \leftarrow conf(plan_i, B_i)$;
- **if** $(conf_{obl} \neq contract.EC.conf(goal_i, B_i)) \parallel (pre \neq contract.EC.pre(goal_i)) \parallel (eff \neq contract.EC.eff(goal_i))$ **then**
  - $contract.EC.pre(goal_i) \leftarrow pre$;
  - $contract.EC.eff(goal_i) \leftarrow eff$;
  - $contract.EC.conf(goal_i) \leftarrow conf_{obl}$;
  - Send updated *contract* to the Dependant of $goal_i$ using an *obligationMaintained* message;

The *updateDependencyContracts* function (Algorithm 25) performs both dependency contract formation and updates for activities within the given plan. Hierarchical structures are supported through recursion (similarly to maintenance agenda formation). The algorithm will iterate through all leaf activities in the $plan_i$, performing con-

tract formation or update steps, and finally returning cumulative sets of preconditions and effects. The returned effects set is formed as the accumulated (in execution order) effects of all leaf activities in $plan_i$, representing the overall state transition resulting from executing $plan_i$.

Algorithm 25: The *updateDependencyContracts* function

Data: g – the goal being met (composite activity being performed) by p
p – a plan or subplan of n activities (p=$\{a_1,\ldots,a_n\}$), where $n \geq 1$
pre – cumulative preconditions of activities prior to p
eff – effects of activities executed prior to p

Result: *pre* – *pre* updated to account for preconditions of each $a \in p$
eff – *eff* updated with the effects of each $a \in p$

for *Each* $a_i \in p$, *from* $i = 1$ *to* $i = n$ **do**
  $C_a \leftarrow$ getCapability(a_i);
  if a_i *is a subgoal with subplan* **then**
    $p_{a_i} \leftarrow$ subplan for a_i;
    $pre, eff \leftarrow$ updateDependencyContracts(a_i, p_{a_i}, pre, eff);
  else
    /* Set pre to include preconditions of a_i, excluding
       those states established by prior activities in p */
    $pre \leftarrow (pre \cup C_a.pre(a_i)) \setminus C_a.eff(a_i)$;
    $eff \leftarrow eff \cup C_a.eff(a_i)$;
    $conf_a \leftarrow C_a$.conf(a_i);
    performContractUpdates(a_i, eff);
  $i \leftarrow i+1$;
return pre, eff;

The preconditions set is defined as the goal preconditions, plus any *non*-holding preconditions of $plan_i$ activities – i.e. defining states the obligant requires the dependant to establish. These preconditions should hold as a condition of contract acceptance; however, as conditions change, it is possible a scenario will emerge where the obligant cannot restore required preconditions through maintenance. Updating the *EC* preconditions field allows these to be conveyed to the dependant – giving that agent the opportunity to adopt responsibility and provide states required by the obligant using it's *own* capabilities.

The updated preconditions set is formed by iterating through all leaf activities in $plan_i$, adding their preconditions to the cumulative returned set *if* not established by the accumulated effects of preceding activities are removed. This ensures this cumulative preconditions set only contains states required to be present in the starting execution context for $plan_i$ – accounting for the preconditions of *all* $a_1, \ldots, a_n \in plan_i$ to execute – and omits those established by internal casual links (where preconditions of a_i are achieved by the effects of at least one a_x, where $x < i$).

After dependency contract updates are performed for an $plan_i$, the agent identifies whether the associated $goal_i$ corresponds to an obligation contract. If so, and where the confidence, preconditions or effects derived from $plan_i$ differ from that contract's *EC* field, the obligation contract is updated and transmitted to the dependant using an *obligationMaintained* message. This signifies the contract update follows a *maintain* operation, and that the obligant has already performed (or attempted to) any modifications required to restore confidence, or ensure preconditions, for that obligation.

The *performContractUpdates* function (Algorithm 26) defines conditions for determining whether contract formation or updating is required for a given activity a. The cumulative effects (*eff*) of prior activities in $plan_i$ determine the *CL* field of the associated contract, with updates performed where *CL* has changed (i.e. due to changes in preceding plan activities from maintenance modification or propagated changes in the *EC* fields of preceding activities' dependency contracts).

The policy field is immutable once a contract has been agreed between agents, as acceptance of an obligation is – at least partly – likely predicated on the exact policy agreed between the obligant and dependant(s). The contract formation process may involve negotiation between the dependent and obligants to define maintenance trigger conditions and constraints – such that modifying a maintenance policy would require a new contract be agreed, entailing a different protocol than for the (comparatively simple) update of information.

We only describe an abstract protocol, to illustrate our assumptions towards information held regarding dependency and/or obligation relationships. Contract formation and update processes entail additional communication cost, compared to both

Algorithm 26: The *performContractUpdates* function, a subfunction of *formAndUpdateContracts*

Data: a – An activity, corresponding to an external capability
eff – Cumulative effects of prior activities in the $plan_i$ containing a

if C_a *is external type* **then**
  if $\exists contract \in Dependencies$ *where* $contract.\alpha = a$ **then**
    `// Check for causal link updates`
    if $eff \neq contract.CL$ **then**
      $contract_{update} \leftarrow$ (Copy of) $contract$;
      $contract_{update}.EC \leftarrow eff$;
      Send update message for $contract_{update}$ to obligants;
  else
    $contract_{new} \leftarrow$ new Contract();
    $contract_{new}.\alpha \leftarrow a$;
    $contract_{new}.CL \leftarrow eff$;
    $contract_{new}.\phi \leftarrow$ getPolicy(a);
    `// External Capability will depend upon Obligants`
    Send $contract_{new}$ formation request to (prospective) obligants;

approaches that do not form contracts (but risk capable agents being unavailable when required), or reactive approaches (which do not require information to reason regarding *potential* failure). In the worst case, a $plan_i$ of n delegated activities (for an obligated $goal_i$) will send $n+1$ messages in each reasoning cycle (i.e. $O(n)$ cost for the *sending* agent) – representing update of dependency contracts for n activities, plus for an obligation contract corresponding to $goal_i$. This presents potential cost issues, particularly in domains with high degrees of decomposition and delegation between agents (increasing the size of n), or which are subject to frequent exogenous change and maintenance (requiring frequent contract updates), suggesting further investigation into communication optimization is desirable.

10.3 Maintaining Obligations

Obligation maintenance occurs where the i passed into *maintain* corresponds to an Obligation contract (i.e. $goal_i$ represents an activity delegated to the agent), and is

performed when either;

- The currently selected i has a $goal_i$ corresponding to an accepted obligation contract (activity delegated to the agent)
- The agent is idle (has no intentions selected for execution), but has *suspended* intentions for meeting current obligations

Suspended intentions can be pre-emptively maintained before execution – meaning contract updates sent to the dependant will reflect the obligant's ability to restore confidence through maintenance. If idle and beliefs have changed, the obligant iterates through and maintains suspended obligation intentions (Algorithm 27). The execution context for each intention is estimated using *addEstablishedStates* (Algorithm 28), which inserts casual links defined by the relevant obligation contract into the current B. This is not required if maintaining *selected* obligation intentions, as the current B defines the execution context.

Algorithm 27: The *maintainObligations* function

```
Data: Obs – The set of all obligation contracts held by the agent
// Assumes B_old is initialized to ∅ at agent startup
if B ≠ B_old then
    for Each contract_ob ∈ Obs do
        i_ob ← suspended intention where goal_i=contract_ob.α;
        B_ob ← B copied and updated with contract_ob.CL;
        changed_ob ←maintain(i_ob);
        if changed_ob then
            formAndUpdateContracts(i_ob, B_ob);
B_old → copy of B;
```

Algorithm 28: The *addEstablishedStates* function

Data: a – goal met / activity performed by a plan
B – a set of agent beliefs
Result: B updated with applicable causal link effects
if $\exists contract \in Dependencies$ *where* $contract.\alpha{=}a$ **then**
  $\Delta B \leftarrow B \cup contract.CL^{+}$;
  $\Delta B \leftarrow \Delta B \setminus contract.CL^{-}$;
  return ΔB;
else
  return B;

This assumes agents identify plans when deciding whether to accept an obligation request, as part of identifying whether the agent *can* perform that activity with sufficient confidence. Forming a $plan_i$ in advance enables more accurate communication of confidence, and allows maintenance to pre-emptively address the impact of exogenous change. This second case of obligation maintenance provides mutual belief maintenance; the obligant acts both to re-calculate confidence *and* identify whether it can respond to negative changes through maintenance changes to the suspended $plan_i$.

10.3.1 Obligation Maintenance Cost

Iteration through suspended intentions incurs additional cost from planning calls during maintenance and communication of contract updates (Section 10.2.4). Given worst case $O(n_a+1)$ messages for intention contract updates (where n_a is the number of $plan_i$ activities, and $goal_i$ is an obligation), for n_i cached $plan_i$s, worse case messaging complexity is $O(n_i n_a + n_i)$. Similarly, if we assume *every* obligation requires maintenance planning, this leads to $O(n_i)$ planning calls (and thus computational cost). This cost serves as our justification for reserving suspended obligation maintenance occur only when an agent is idle – preventing execution delays from maintenance reasoning where an intention *is* selected.

We can also suggest further optimisation approaches for domains where suspended obligation maintenance poses excessive cost, particularly if agents may hold large numbers of obligations in advance of their execution. Dependants may send addi-

tional information to allow obligants to be selective regarding such maintenance. For example, if an obligant has multiple obligations to the same dependant (including indirectly) and is given the relative ordering of their precedence within the dependant $plan_i$, obligation maintenance could be restricted to only those due to execute next.

It is not mandatory for CAMP-BDI agents to pre-emptively form intended plans for obligations; agents can instead use internal composite capability knowledge to update confidence (i.e. for the contract *EC* field) upon belief changes, rather than maintaining suspended $plan_i$s. We have suggested suspended intention maintenance to allow obligants to perform local mitigation, avoiding dependants having to perform maintenance if the obligant could resolve threats locally. It may be that a fixed capability approach leads to scenarios where no viable external capability (for re-delegation) exists, yet which could have been handled through suspended obligation maintenance by (more semantically aware) obligant.

Regardless of whether for an executing or suspended intention, the same post-maintenance behaviour applies, with an *obligationMaintained* message being sent (Section 10.3). This may lead to updates being sent to (sub)obligants (i.e. where a maintained obligation $plan_i$ has dependencies) to update *CL* fields. These do *not* trigger obligation maintenance by that (sub)obligant; adoption of responsibility for performing maintenance always progresses upwards in the decomposition hierarchy, to avoid lengthy iterative maintenance where obligation maintenance triggered by contract change subsequently triggers dependant maintenance – leading to further contract changes and corresponding maintenance of the earlier obligation, and soforth. This again has conceptual similarity to re-refinement HTN plan repair; both attempt to re-refine composite activities (including goals decomposed through delegation) increasingly 'up' a local or distributed plan hierarchy until successful.

10.3.2 Maintaining Joint Obligations

Joint activities occur where an activity must be performed simultaneously by multiple agents (requiring co-ordination); *obligations* for multiple agents to perform a joint activity result in a *joint obligation*. We denote joint activities through inclusion of multiple agent identifiers within their variables – i.e. *moveBox*(*robot1*, *robot2*, *box*, *a*,

B), where *robot1* and *robot2* are joint actors[3].

Micalizio and Torasso [2008b] suggest joint action be seen as simultaneous execution of a subset of simple actions – i.e. $a^{i,j}$ defining a joint activity where a^i and a^j execute simultaneously and actively co-operate to achieve the same effect. They suggest a^i and a^j can be considered individual agent obligations (i.e. upon $agent_i$ and $agent_j$), with separate preconditions and where agents take individual responsibility. Within CAMP-BDI these individual activities can be seen to reflect individual contributions of separate obligants to the same contract, for the same activity and to the same dependent, and requiring co-ordination as part of execution.

Joint Obligations represent a special maintenance case, due to the co-dependencies between joint obligants. CAMP-BDI makes a number of abstracting assumptions regarding joint activity performance (which may be further examined by future work):

- Synchronization occurs as an inherent part of scheduling and execution.
- That if synchronizing communication *is* required to be represented within $plan_i$s, it can be modelled in capability terms, where preconditions and effects represent the requirements and purpose of that communication (perhaps using a similar approach to knowledge-based modelling of operators within PKS, by Petrick and Bacchus [2004]).
- The planning process invoked by agents – both to initially form a $plan_i$ and for maintenance planning – accounts for multiagent planning (I.e. if an agent can form a $plan_i$ accounting for joint activity, it can form a *maintenance* plan doing the same).

Joint Obligations result in each obligant holding an independent $plan_i$ where the joint obligants co-ordinate their individual execution to perform the delegated joint activity. Each obligant must be capable of the contract activity, with capability knowledge stating where multiple obligants are required (such as through the signature, as described earlier). These activities are assumed *inherently* 'joint' – such that obligant maintenance cannot modify a local $plan_i$ for a joint obligation in such a way that the joint obligant set is changed.

The *EC* field of a contract for a joint obligation merges the individual capability

[3] Agents do not *have* to define actors within activity signatures; we use this convention to easily state obligants for a delegated activity.

information of each obligant, sent in their individual contract request replies, using the *mergeCapabilities* function (Section 8.2.4, Algorithm 4). The resultant capability defines confidence as the *minimum* confidence of individual obligants; we do not use an averaged value as this risks 'hiding' any risks (weak points) stemming from a single, but low confidence, obligant.

A dependant may also be itself a joint obligant; i.e. if *Truck1* holds a plan containing an activity such as formConvoy(*Truck1*, *Apc1*, *Apc2*, *A*, *B*). In this case, that agent adopts both the dependant and obligant role in maintenance. As a dependant, the agent attempts to form a new $plan_i$, allowing it to still perform that joint activity. In the obligant role, the dependant can replace the joint activity through effects maintenance if confidence cannot be restored – conceptually equivalent to re-refining at a more abstract plan level.

10.4 Maintaining Plans containing Dependencies

We use 'dependency maintenance' to refer to maintenance of an intention *i* where $plan_i$ contains delegated activities; representing adoption of *maintenance responsibility* by the dependant for it's part of a distributed plan. Dependency maintenance occurs under two conditions, neither of which require modification of our previously defined *maintain* algorithm; however, the latter does introduce an additional invocation condition:

- Within the CAMP-BDI reasoning cycle, where a maintained *i*'s $plan_i$ contains dependencies.
- Following obligants' maintenance, if the *EC* field of the dependency contract has changed.

The *maintainDependencies* (Algorithm 29) function is used where dependency maintenance is performed following receipt of *obligationMaintained* messages. These messages provide updated contract information (indicating a change in how the obligant(s) intend to perform an obligation), and signify the obligant has already attempted maintenance – including performing any possible local plan modifications in response to generated maintenance tasks. This ensures obligants will *not* modify obligation $plan_i$s in parallel with their dependant's maintenance, yet allows maintenance policies (i.e. the maintenance task generation and maintenance plan acceptance conditions) to

be shared within contracts.

The function first updates dependency contracts, before maintaining impacted intentions. The first step ensures *maintain* calls use the most up-to-date dependency contract information. The latter step avoids the same $plan_i$ being maintained multiple times due to different dependency contract updates (i.e. for multiple *obligationMaintained* messages, corresponding to different dependencies in the same $plan_i$). It can also be considered similar to the *consolidation* function applied in *formAgenda* (Section 9.3.1), as multiple plan threats (the multiple *obligationMaintained* messages) are aggregated into one event and handled through a single operation.

Algorithm 29: The maintainDependencies function

Data: *eventQueue* – all events which occurred between the previous and current reasoning cycle

$maintainSet \leftarrow \emptyset$;
for *each obligationMaintained message in eventQueue* **do**
  $i \leftarrow$ intention where $contract_{new}.\alpha \in i_{plan}$;
  if $\neg\exists\ i \in maintainSet$ **then**
    $maintainSet \leftarrow maintainSet \cup i$;
for *each* $i \in maintainSet$ **do**
  $B_i \leftarrow$ copy of current B;
  if *(*$\exists contract \in Obs$ *where* $contract.\alpha = goal_i$*) & (i is suspended) & (contract.EC has changed)* **then**
    Insert $contract.CL$ into B_i;
    maintain(i, B_i);
    formAndUpdateContracts(i);

In the case of joint activities – i.e. dependencies resulting in joint obligations – the dependant waits until it receives *obligationMaintained* messages from all obligants. The *extractObligationMaintained* function (Algorithm 30) removes *obligationMaintained* messages for a joint activity dependency from *eventQueue* and temporarily stores them. Once messages are received from *all* joint obligants, a single new *obligationMaintained* message is generated – aggregating the individual obligant updates into a merged contract – with a corresponding 'receipt' event inserted into *eventQueue*.

Algorithm 30: The *extractObligationMaintainedMessages* function

Data: *eventQueue* – A queue of events perceived by the agent, which can include those signifying receipt of *obligationMaintained* messages from obligants

Result: *eventQueue* – Updated to contain *obligationMaintained* messages representing the cumulative updates for all obligants, with messages received from individual joint obligants removed

messages ← every message receipt of type **for** *each msg* ∈ *messages* **do**
    contract ← contract sent in *msg*;
    α_{ob} ← *contract*.α;
    if α_{ob} *is a joint obligation* **then**
        eventQueue ← *eventQueue* \ *msg*;
        $received_{\alpha_{ob}}$ ← $received_{\alpha_{ob}} \cup msg$;
        if $\exists msg \in received_{\alpha_{ob}}$ *for every obligant of* α_{ob} **then**
            `// Form merged contract from all obligant updates`
            $contract_{merged}$ ← empty contract;
            for *each* $msg_{received} \in recieved_{\alpha_{ob}}$ **do**
                $contract_{received}$ ← content of $msg_{received}$;
                if $contract_{merged} isempty$ **then**
                    $contract_{merged} \leftarrow contract_{received}$;
                else
                    $contract_{merged}$ ← $contract_{received}$ merged with $contract_{merged}$;
            msg_{merged} ← new *obligationMaintained* message;
            Set msg_{merged} content as $contract_{merged}$;
            $msg_{receipt}$ ← receipt event for msg_{merged};
            eventQueue ← $eventQueue \cup msg_{receipt}$;
            $received_{\alpha_{ob}} \leftarrow \emptyset$;
`// Update contracts with messages retained in the` *eventQueue*
messages ← every message receipt of type *obligationMaintained*∈ *eventQueue*;
for *each msg* ∈ *messages* **do**
    $contract_{new}$ ← contract sent in *msg*;
    $contract_{old}$ ← contract ∈ *Dps* where $contract_{old}$.α=$contract_{new}$.α;
    Dps ← *Dps* \ $contract_{old}$;
    Dps ← *Dps* ∪ $contract_{new}$;
return *eventQueue*;

This avoids multiple iterative dependency maintenance operations, triggered by each obligant's *obligationMaintained* message. In a broader sense, this renders joint and single agent dependencies identical within the context of *maintain*; both are reduced into a single contract update event, communicated through a single *obligationMaintained* message receipt event. We assume implicit synchronization in the timing of obligation maintenance (i.e. that joint obligants perform maintenance in reasonably close temporal proximity, such that one obligants 'contribution' correlates with those of the other joint obligants) – such that the merged single contract can be considered representative of the current state of the joint obligation.

Receipt of an *obligationMaintained* message does not *force* dependant maintenance, as no synchronization is required between the sending obligant (as with obligation maintenance). Intentions may both contain dependencies and be associated with obligations; this may lead to 'chains' of maintenance activity where dependency maintenance (where that *i* is also an obligation) subsequently triggers maintenance by *it's* own dependant. A generalized example of such a process is given below (Figure 10.3):

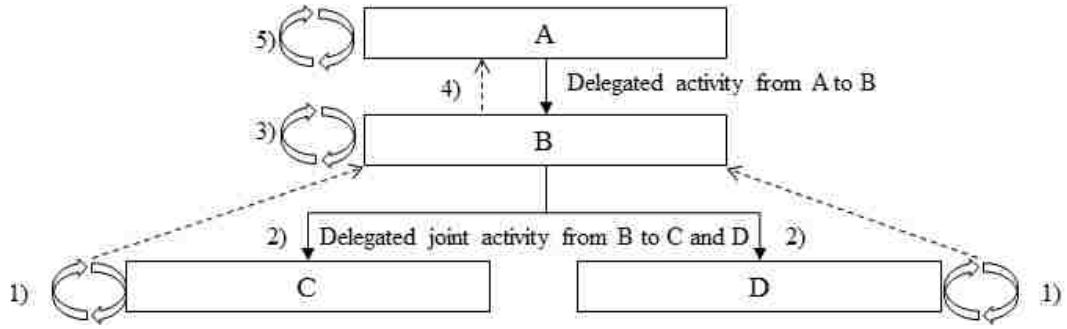


Figure 10.3: Example adoption of responsibility in a hierarchical team, where *B* is an obligant of *A*, and *C* and *D* are obligants for a joint activity in *B*'s plan.

1. Agents *C* and *D* call *maintain* within local reasoning cycle(s).

2. *C* and *D* individually perform post-maintenance messaging; each sends a *obligationMaintained* message to *B* that includes contracts updated to account for any maintenance changes.

3. *B* calls it's *maintain* method upon receipt of *obligationMaintained* messages from all obligants. The information in these messages is used by *B* to update

it's corresponding dependency contract.

4. *B* sends *A* post-maintenance messaging, again using *obligationMaintained* messages, with that updated contract also reflecting the updated information received from *C* and *D*.

5. *A* calls *maintain* upon receipt of *B*'s post-maintenance message; as *A* is not an obligation, no further messaging is required.

The Dp set is always updated with contracts from received *obligationMaintained* messages, regardless of whether dependency maintenance is triggered (although maintenance may remove that contract by modifying the dependant $plan_i$). Assuming execution of the dependant i will begin at some point, i will be maintained through the CAMP-BDI reasoning cycle and consequently utilize the received contract information (unless superseded by more recent updates).

Dependency maintenance may result in multiple maintenance operations, if no intention is selected and the *eventQueue* includes receipt of multiple *obligationMaintained* messages. This leads to worst case complexity of $O(n_{om}n_{mt})$ at the individual agent level, where n_{om} is the number of *obligationMaintained* messages (at worst equal to the size of Dp) and n_{mt} stems from our calculation of worst case maintenance complexity as $O(n)$. The overall cost will increase with adoption of responsibility by increasingly higher-level dependants; if the maximum 'depth' of dependencies (i.e. from the root to leaf activity in the distributed plan) is n_D agents, the total complexity across the *system* would be $O(n_{om}n_{mt}n_D)$. This is partially mitigated by CAMP-BDI agents only maintaining dependencies if they do not have an intention selected – although this does not reduce *worst case* complexity, it ensures agents only expend computational effort on message-triggered dependency maintenance if otherwise idle.

These costs could be reduced by optimisation to account for more specific differences between dependencies. For example, *extractObligationMaintainedMessage* can be modified to filter as well as aggregate messages – received contract updates without a significant (i.e. below maintenance policy threshold levels) confidence loss could be stored in the Dp set and removed from the *eventQueue* to defer maintenance. Alternatively, a private/public action (Brafman and Domshlak [2008]) approach could also be used to avoid update communication if contract changes only concern private atoms,

or update frequency could be determined to decrease with (if known) greater temporal *distance* before execution of that obligation.

10.5 Example Distributed Maintenance Behaviour

We now present a detailed example of distributed maintenance in the Cargoworld domain, using the geography given in Figure 10.4. The *LogisticsHQ* agent, through a combination of dependencies upon*Truck1* and *Helicopter*, intends to transport cargo from location F to M. This results in the distributed plan shown in Figure 10.5, where various dependecy contracts are formed with *Truck1* and *Helicopter* by *LogisticsHQ*.

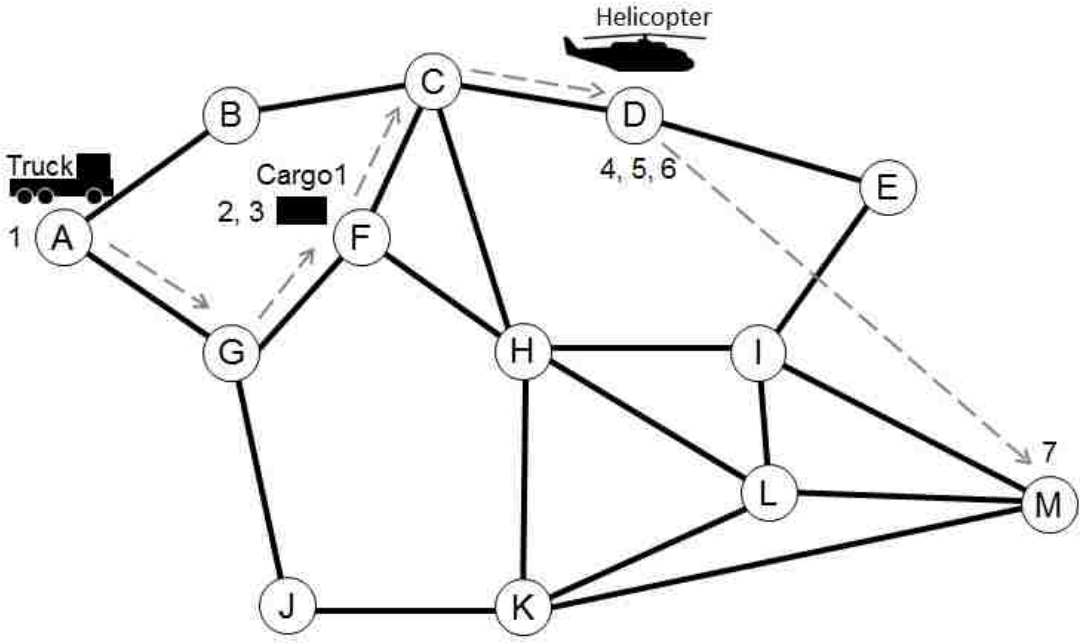


Figure 10.4: Example geography, showing sequence of agent activities – *Truck1* will travel to F(1), load *Cargo1*(2), before travelling to D(3), and unloading *Cargo1*(4). The *Helicopter*, present at D, will load *Cargo1*(5) before flying to M(6) and unloading(7).

Figure 10.6 illustrates which agents perform obligation maintenance and, as a result, send *obligationMaintained* messages. Here, *Truck1* is currently performing it's obligation to *moveTo*(A, F). *LogisticsHQ* has suspended its $plan_i$ for *deliverCargo*(M) until that executing dependency completes. As part of execution *Truck1* will maintain the intention for *moveTo*(A, F), including sending an *obligationMaintained* message to the dependent *LogisticsHQ*.

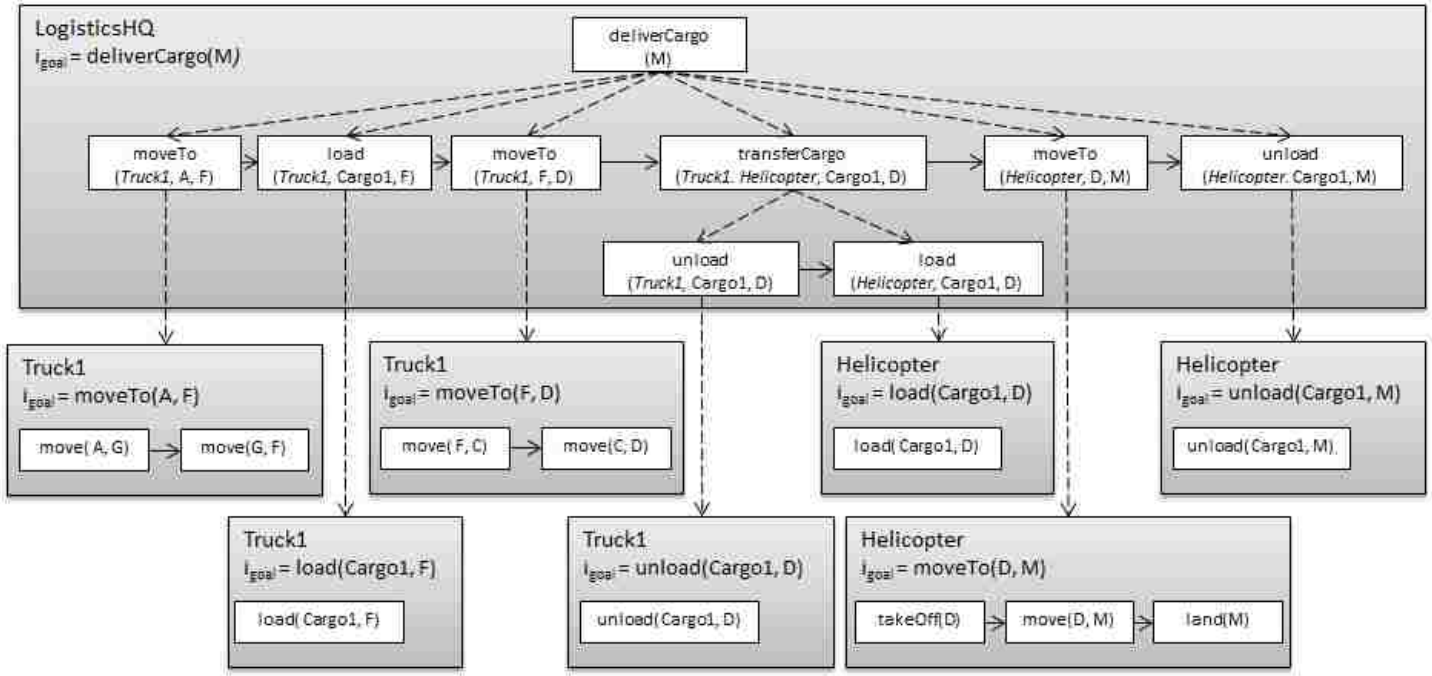


Figure 10.5: Example distributed plan, showing *LogisticsHQ* holding multiple dependency contracts with *Truck1* and *Helicopter*. Each grey box corresponds to an agent's $plan_i$, where solid lines indicate relative activity order. Dashed lines depict task decomposition; either to internal subplans (within the grey box), or through delegation (and achievement by obligation $plan_i$s).

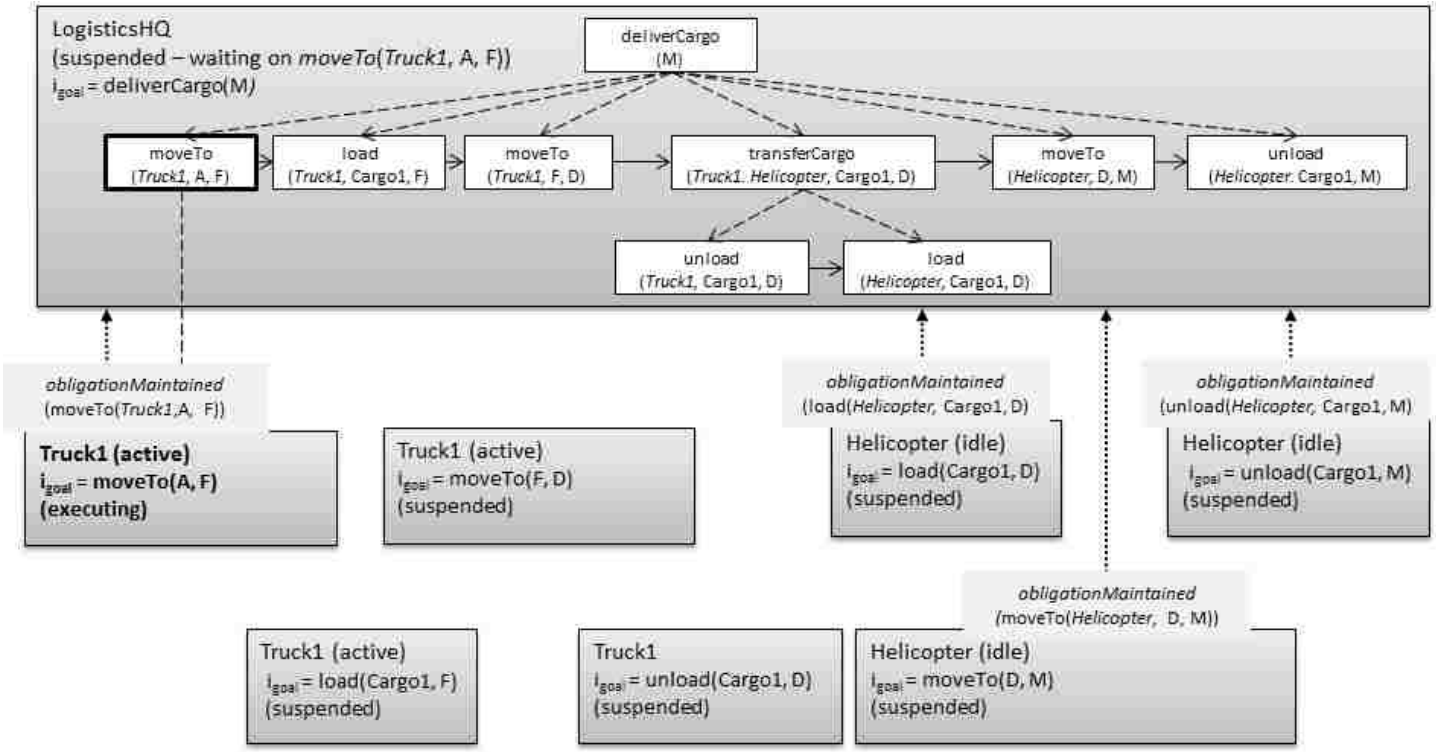


Figure 10.6: Example of *obligationMaintained* messaging during distributed maintenance. *Truck1* is currently executing its obligation to *moveTo*(A, F) – indicated by bold text – with *LogisticsHQ* suspending it's own dependant $plan_i$ (with the executing dependency indicated by thicker black border) until that obligation completes.

As *Truck1* has an intention selected, it will *not* maintain the suspended $plan_i$s for the obligations to *load*(*Cargo1*, *F*) and *moveTo*(*D*, *F*). The obligant *Helicopter* is currently idle (no selected intention), and *will* perform obligation maintenance of the suspended intentions for *load*, *moveTo* and *unload*. *LogisticsHQ* continues to receive and handle belief updates, but reduces candidate desire and intention candidate sets to empty due to waiting for completion of *moveTo*(*Truck1*, *A*, *F*) – no other intention will be selected for execution, but dependency maintenance can still be triggered by *obligationMaintained* messages.

We suggest a scenario where *Helicopter* detects location *M* is no longer secure – violating preconditions for *moveTo* and *unload* activities at *M*. *Helicopter* fails to successfully modify either obligation's $plan_i$ in mitigation; an *obligationMaintained* message is sent to the dependant *LogisticsHQ* for each. These messages inform *LogisticsHQ* of the following;

1. *Helicopter* has attempted and been unable to counter threats within it's plan for these obligations; the updated contract *EC*'s preconditions are updated to include the violated ¬ *dangerzone* (*M*) condition.

2. *LogisticsHQ* can assume that, having attempted maintenance already, *Helicopter* is unlikely to modify it's $plan_i$s for *moveTo*(*D*, *M*) or *unload*(*Cargo1*, *M*); *LogisticsHQ* therefore can maintain it's dependant *i*.

LogisticsHQ performs message triggered dependency maintenance of the *deliverCargo* intention. Maintenance tasks mt_{pre1} ($mt_{pre1}.a$ =*moveTo*(*Helicopter*, *D*, *M*)) and mt_{pre2} ($mt_{pre2}.a$ =*unload(Helicopter, Cargo1, M)*) are generated; both are preconditions tasks, as the former precedes an existing dependency contract, and the latter achieves a state required for the $goal_i$.

LogisticsHQ successfully handles mt_{pre1}, by inserting a new activity to *secure M* – a corresponding dependency is formed upon *MilitaryHQ*, which in turn forms associated (sub)dependencies upon *APC1* (Figure 10.7) to perform the necessary activities within the world. As *LogisticsHQ* has successfully handled a task, *maintain* exits with the agent performing contract updates to reflect dependant $plan_i$ changes. The *CL* field for *Helicopter*'s obligations to *moveTo*(*D*,*M*) and *unload*(*Cargo1*, *M*) is updated to include the ¬*dangerZone*(*M*) effects achieved by the inserted *secure* activity.

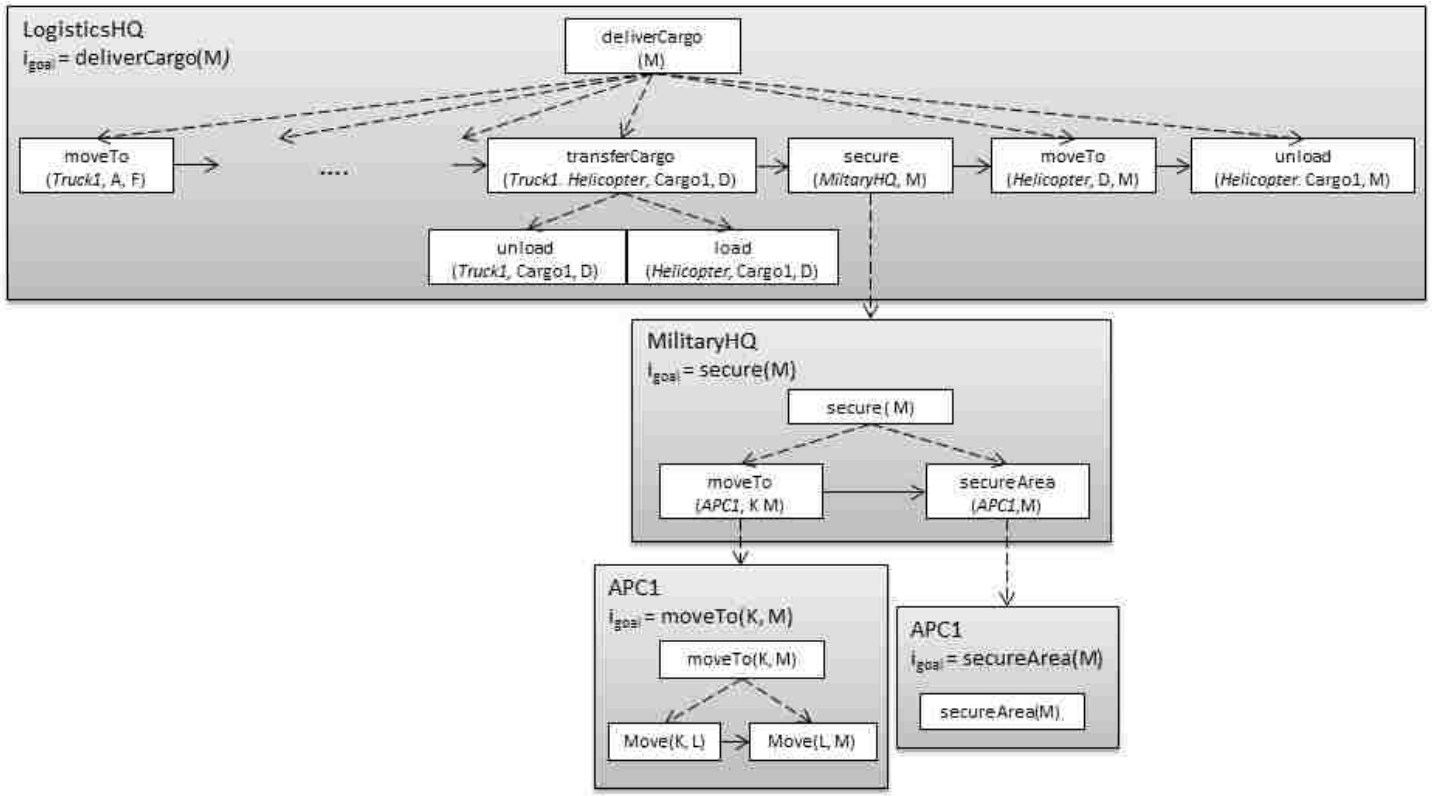


Figure 10.7: Example outcome of effects maintenance at dependant level, a *secure* activity is inserted, resulting in a new dependency contract being formed with *MilitaryHQ* (itself dependant upon *APC1*).

A coincidental side-effect of handling mt_{pre1} will be re-establishment of *unload*'s preconditions (i.e. whose anticipated violation led to generation of mt_{pre2}), which require that same state.

A further example of maintenance at the dependant level is given by Figure 10.8. In this scenario *windy* conditions reduce the confidence of *Helicopter* in *moveTo*; as this stems from weather conditions (i.e. that cannot be modified using any agents capabilities), *Helicopter* cannot find a maintenance plan to restore confidence. *LogisticsHQ* receives an *obligationMaintained* message, where the updated contract's *EC* field gives a lowered confidence value below the threshold (*Th*) set by the contract maintenance policy.

LogisticsHQ consequently generates an *effects* maintenance task $mt_{effects}$ ($mt_{effects}.a$ = *moveTo*(*Helicopter*,D,M)). When handling $mt_{effects}$, *LogisticsHQ* cannot find and substitute a higher confidence maintenance plan for the *moveTo* activity (which is preceded and followed by dependencies). Maintenance scope is increased until a maintenance plan *is* found. As a result, the dependent $plan_i$ is reformed to replace *Helicopter* with use of a new obligant – *Truck1* – to transport *Cargo1* to, and unload at, M. This entails cancellation of existing dependency contracts with *Helicopter*, and formation of dependency contracts upon *Truck1* for inserted maintenance plan activities.

Although these particular examples do not depict maintenance planning scope extending across the entire dependant $plan_i$, *maintain* may see an *executing* dependency removed – e.g. if a dependant messages an updated contract *EC* showing low confidence, continues post-maintenance execution of that obligation, but the dependant's local maintenance replaces that executing activity (dependency). Removing an executing dependency from a $plan_i$ results in the associated contract being cancelled; the obligant will drop the executing i as soon as possible following loss of (external) motivation for that $goal_i$. This does mean the obligant will have wasted time and effort on that partial execution, but the alternative is to force the obligant to wait on maintenance results from it's direct and indirect dependants before continuing execution – such synchronization would result in unnecessary execution delays where an obligation is *not* cancelled as a result of dependant maintenance.

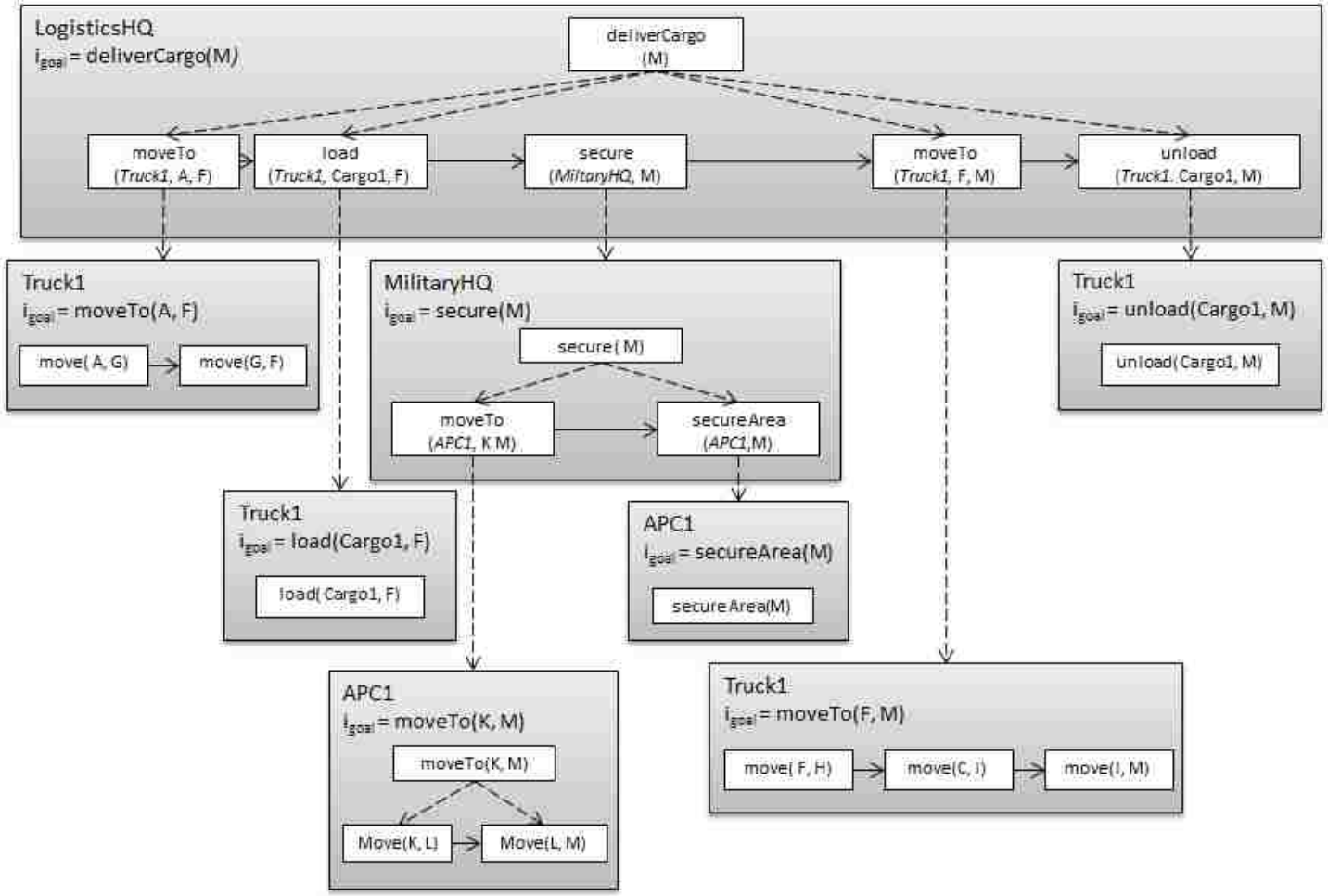


Figure 10.8: Example outcome of effects maintenance at dependant level; *LogisticsHQ* modifies it's $plan_i$ to use *Truck1* as a new (sole) obligant for delivering *Cargo1* to M

10.6 Summary

This chapter extended individual CAMP-BDI maintenance behaviour to cover distributed plans. We described adoption of individual responsibility by agents based upon a combination of structured messaging and knowledge sharing through contracts. A key element of our design is *decentralization* – as the semantic knowledge and situational awareness (of environment and agent's current mental state) requirements will likely render centralized approaches impractical for complex realistic environments.

We preserve agent autonomy and do not mandate dependants automatically respond to *obligationMaintained* messages with maintenance, but instead choose to adopt maintenance responsibility through local reasoning; future work may extend the adoption process to consider factors beyond current intention state. The *obligationMaintained* messages facilitate synchronization, both conveying contract changes and indicating the obligant's local maintenance has completed. This allows the dependdant – if it chooses to do so – to maintain the dependant $plan_i$ under an assumption that, if the shared maintenance policy (defined in the contract) indicates some threat, the obligant has already performed any maintenance changes possible through it's own capabilities.

The supporting architecture defined in Chapter 8 supports our distributed maintenance design. Contracts support maintenance by providing causal link information to estimate execution context when maintaining suspended intentions. Dependency maintenance is supported through the *EC* field – which encapsulates the obligant(s) local semantic knowledge, allowing dependants to reason about delegated activity as if locally performed (i.e. through internal capability). Our capability model provides a common representation, for communication between agents within contracts.

The final element of the supporting architecture are *maintenance policies*. We previously described use of maintenance policies to guide maintenance task generation within *maintain*. Contracts include a shared maintenance policy, to ensure both dependant and obligants form maintenance tasks using the same criteria, and allowing implicit synchronization when combined with the use of *obligationMaintained* messages. If a dependant attempts maintenance following receipt of *obligationMaintained* messages and consequently forms a maintenance task for the associated dependant

activity, the shared policy entails obligant(s) must have already have formed and attempted to handle equivalent local maintenance tasks under those policy conditions.

Structured *obligationMaintained* messaging, combined with contract defined maintenance policies, allows dependency maintenance to avoid modifying larger, more abstract, sections of a distributed plan due to falsely assuming obligants cannot maintain confidence at their local $plan_i$ level. This reduces distributed maintenance changes, minimizing disruption to the distributed intention and associated inter-agent dependency relationships, and also mirroring iterative expansion of planning scope in the local maintenance task handling algorithms.

We have primarily considered CAMP-BDI in comparison to reactive replanning (whilst not *excluding* reactivity; our design inserts itself as reasoning prior to execution, rather than after any execution failure). In the distributed case, reactive replanning can be a simple process where, upon failure, obligants attempt to replan $plan_i$ (where the post-failure state represents the initial state in the planning problem, and the goal is $goal_i$) before reporting failure to their dependant. If that obligant *cannot* form a new plan to satisfy their obligation, the dependant fails that delegated task activity and performs replanning at it's *own* level. Failure and response would escalate up a dependency hierarchy until either an agent successfully replans, or none can.

Our approach does entail greater complexity – and computational cost – than this type of reactivity. As with localized maintenance, agents in a dependency hierarchy must consider *possible* failure rather than post-hoc respond to *known* activity failure. In any realistic system, we must assume false positives will occur during maintenance task identification – causing unnecessary handling and planning operations, with increased computational cost upon the overall distributed team (particularly where maintenance responsibility is adopted at higher levels). Updating of contract information also entails messaging costs – while CAMP-BDI agents require information updates during even successful execution to facilitate maintenance, reactive approaches only require information in a *post-failure* state, as necessary for (attempted) recovery.

The CAMP-BDI design has been founded on a basis that, in at least some cases, it is worthwhile to incur these additional costs – including from false positives – to avoid greater and more significant costs associated with actual activity failure. Such costs are

likely to be activity and domain specific, making it difficult to generalize the trade-off between risking these and our proactive costs. It is also possible to partially mitigate these if a reactive approach is implemented in parallel, using maintenance policies. Confidence threshold values can increase or decrease the probability of maintenance task generation (and maintenance planning), by being raised or lowered based upon the severity of failure consequences for their associated activity-agent pair. Failure of activities with low associated costs could be set as solely handled reactively, in recognition of a relative ease of recovery. Depending on planning implementation, contract updates sent can be reduced to only occur where changes to *CL* or *EC* fields will impact other agents (i.e. correspond to *public* atoms).

The following chapter, details experimental evaluation of the CAMP-BDI design. We consider a variety of metrics against a reactive approach, including goal achievement, under a variety of perturbation and failure consequence configurations.

Chapter 11

Experimental Evaluation

Hanks *et al.* [1993b] describes the purpose of experimentation in AI as being to discover the relationship between a system S, environment E, and the behaviour B of S. This chapter evaluates our contribution, comparing robustness (effectiveness of B) for CAMP-BDI against alternative robustness approaches (various S) in a *Cargoworld* environment under various levels of perturbation (E).

We first describe implementation of the *Cargoworld* and our evaluated robustness approaches in Section 11.1. This is followed by our experimental design (Section 11.2), giving the key metrics measured, experimental protocol and environmental configurations. The following chapter presents our experimental results and compares CAMP-BDI against reactive and continuous replanning approaches.

11.1 Implementation

Our implementation extended *Jason* (Bordini and Hübner [2006]), a Java based BDI framework[1] to implement both the *Cargoworld* simulator providing our experimental environment, and the agent systems being evaluated[2]. Jason provided a stable, well-documented framework to support implementation of, and agent integration with, the *Cargoworld* simulator, and has been used for prior experiments involving integration of runtime planning within BDI agents, such as *Peleus* (Meneguzzi and Luck [2008]).

[1] `http://jason.sourceforge.net/wp/`

[2] Code for both is accessible at `https://gitlab.com/aldo_14/CAMP-BDI`

11.1.1 Implementation of the Cargoworld Simulator

Our experimentation used an implementation of the *Cargoworld* domain – the *Cargoworld simulator* – introduced in Section 2.4, and further specified in Chapter 7. Environmental geography was generated using a seeded, procedural algorithm; an initial geography was used for implementation and testing, with new geographies generated for the actual experimentation (Section 11.2.1). Probabilities for exogenous change and debilitative failure consequences were parametrized for explicit specification.

Activity in *Cargoworld* centred around delivery of cargo to requesting locations. Delivery requests were generated at a random location (selected using a seeded random number generator), and accompanied with generation of a new cargo object at the most distant (but accessible) junction from the request location. Only one cargo object was present at any one time – existing cargo objects (i.e. undelivered from prior failure) were destroyed upon generation of new requests. This minimized divergence in the initial state contexts under which goals were generated for the approaches being compared, provided explicit control over the difficulty of each goal request, and ensured the desired distance between cargo request location and the cargo object (to be) delivered.

Exogenous events, with frequency and quantity determined by configured probabilities, were generated either after activity completion and upon generation of a new cargo request (referred to as 'per step' for clarity). Activity execution duration was intentionally truncated for the purposes of practical experimentation; in a real-world scenario we would expect a typical move activity to take significantly longer than feasible for repeatable experimentation. This led to agent reasoning – especially planning – dominating over the time spent acting to a degree unlikely to hold for realistic activity durations. A 'stepwise' exogenous change approach was intended to avoid plan success solely due to the world changing to a more optimal state *whilst* agents were planning, rather than as a result of effective mitigation behaviour.

Particular states and state combinations reduce the risk of activity success, without prohibiting it. For example, vehicles could travel slippery tarmac roads, but would face consequently increased failure risk. Risk increasing states may have a combined impact; i.e. a Truck would face further increased risk if travelling a slippery tarmac

road while *also* damaged.

The *Cargoworld* domain provided a Transport type domain; this type of domain has been extensively used in existing work, including within International Planning Competition (IPC) domains (Section 2.2) frequently employed as planner benchmarks (Helmert [2008]). Our simulator implementation allowed specific control over perturbation, providing the ability to scale environmental difficulty. This provided a stochastic, non-episodic environment, where exogenous change can occur randomly (from an agent perspective) with effects lasting beyond the current sequence of MAS activity.

11.1.1.1 Exogenous Change Parameters

Figure 11.1 lists parameters controlling environmental perturbation, as used to control generation of exogenous change events. Parameter values were continuous, with their range 0:1 equating to 0 to 100% probability or proportion per step.

Road flooding state could only be modified by exogenous change, meaning agents could not modify these road conditions (i.e. to be dry). Maximum limits were consequently imposed (through *maxSlippery* and *maxFlooded*) to avoid scenarios where the road conditions rendered system goals impossible regardless of any failure mitigation approach (i.e. where no routes existed between locations), and to help control environmental difficulty.

Parameter	Purpose
windyChangeChance	Probability of changing to or from a '*windy*' state.
closeRoadChance	Probability of at least one or more (to a limit defined by **maxRoadClosuresPerStep**), randomly chosen roads becoming '*closed*'.
openRoadChance	Probability of opening a random, closed road.
addDzChance	Probability of one or more (to the maximum number defined by **maxDzPerStep**) junctions becoming dangerous – i.e with addition of a '*dangerZone*' state.
removeDzChance	Probability of a (randomly selected) existing '*dangerZone*' being removed.
maxRoadClosuresPerStep	Maximum number, as a proportion, of roads which can be set as '*closed*' in any single step.
maxDzPerStep	Maximum proportion of junctions which may have a corresponding '*dangerZone*' added in any single step.
chanceOfFlooding	Probability of one or more roads (limited by **maxFloodPerTurn**) increasing in flood state per step.
chanceOfDrying	Probability of at least one road (limited by **maxFloodPerTurn**) decreasing in flood state (drying) per step.
maxSlippery	Maximum proportion of roads which can be '*flooded*' at any one point in time.
maxFloodPerTurn	Maximum, as a proportion, roads which can be increased in flooding state per step.
maxDryPerTurn	Maximum, as a proportion, roads which can dry out per step.
maxFlooded	Maximum proportion of roads which may be '*slippery*' at any one time.
maxSlippery	Maximum proportion of roads which may be '*flooded*' at any one time.

Figure 11.1: Parameters controlling exogenous change in the *Cargoworld* simulator

11.1.1.2 Failure And Debilitative Consequence Parameters

The simulator modelled two types of activity failure; **Preconditions** failures where preconditions did not hold at execution, and **Non-deterministic** failures occurring *during* execution due to that activity being performed in a state with increased risk of failure – i.e. a *Truck* moving along a *slippery* tarmac road, or doing so while damaged, would have increased probability for failing during execution of that activity.

Exogenous failure – i.e. solely due to some exogenous event *during* execution – was not modelled; this type was explicitly *not* handled by CAMP-BDI (which is concerned with 'predictable' failure types). Such failures could only be handled reactively; we can intuit our approach would have an inevitable disadvantage if such types dominated failure, although we assume any practical implementation would see CAMP-BDI being paired with complementary robustness methods (including reactive ones).

Failure risked debilitative consequences, with probabilities defined through the parameters in Table 11.2 (applied individually for each failed activity). Agent health was modelled in three states, of increasing severity – healthy (optimal), damaged (suboptimal performance, i.e. increased risk of failure for any activity) and mortal (unable to act). Agent damage could persist over multiple delivery requests, and also accumulate – i.e. failure by a damaged agent could cause mortal damage. If a damaged agent was idle for the duration of a cargo request, however, it would partially 'heal' (from '*mortal*' to '*damaged*', or '*damaged*' to '*healthy*').

Parameter	Purpose
cargoDestroyChance	If the agent is carrying cargo, the probability of that cargo being destroyed by failure.
cargoSpillChance	The probability of cargo (if carried) *spilling*; rendering a road '*toxic*' if the acting agent was on a road, or surrounding roads '*toxic*' if that agent was at a junction. Toxic roads are unusable for travel by any agent other than APC and Hazmat types – the latter can remove toxic states through successful *decontaminate* activities.
chanceOfPostFailureDamage	Probability of agent damage following a failure.
stuckChance	Probability of the agent becoming '*stuck*' (i.e. skidding off the road) and unable to move following failure of a movement activity. This reduces the effectiveness of 'brute force' solutions that repeat a specific movement (with a non-zero chance of failure) until it eventually succeeds.

Figure 11.2: Parameters controlling probabilities for potential failure consequences.

11.1.2 Implementation of Experimental Systems

Several different robustness approaches were compared against CAMP-BDI. We extended *Jason* to support contract formation (and general support for performing delegated activity) and runtime planning, to provide a common framework which was extended to implement both CAMP-BDI and our comparative approaches. *Jason* is an AgentSpeak (ASL) interpreter; agent behaviour is defined by predefined (unground) plans, triggering events (e.g. goal addition or belief changes) and selection preconditions (Figure 11.3); the same agent behaviour definitions (ASL files) were used in all approaches under evaluation, with robustness behaviour provided through the generalized underlying agent reasoning.

The LPG-td planner (Gerevini and Serina [2002]) was employed for runtime planning; this planner previously shown competitive performance in IPC domains (Hoffmann and Edelkamp [2005]), and offered a means to test alternate modalities of planning within CAMP-BDI (Section 11.1.2.1). Our algorithms were implemented as be-

ing planner-agnostic, solely assuming an accessible module existed which – given a planning problem – would attempt to form and return a plan (as described in Section 9.4.2).

```
@moveToAtJ
+!moveTo(AGENT, CLOC, CDES):
not busy(AGENT) & atJ(AGENT, CLOC) & not mortal(AGENT) &
location(CLOC) & location(CDES) & not dangerZone(CLOC) &
not dangerZone(CDES)
<- vehicle.ia.planRoute(CLOC, CDES); !doRoute(AGENT, CLOC, CDES).
```

Figure 11.3: Example AgentSpeak plan, labelled *moveToAtJ*, for travelling between two junctions. This plan is applicable for intention selection upon addition of the *moveTo* goal (indicated by `+!`), provided the context condition (in italics) holds. The body of the plan (following `<-`) consists of two steps; *planRoute* generates and inserts a new route plan (sequence of move activities) into the plan library, and a second which adds a *doRoute* goal to stimulate adoption of that route plan as a $plan_i$.

All experimental systems employed *LPG-td* (Gerevini and Serina [2002]) for runtime planning. This planner was selected due to both prior competitive performance in IPC domains (Hoffmann and Edelkamp [2005]) and support of PDDL metric extensions; the latter allowed for additional evaluation of *CAMP-BDI.Quality* (described subsequently), which performed pseudo-probabilistic planning using a metric minimization approach. As LPG-td is a generalized, domain independent, planner we also viewed it as unlikely to have any biasing features for or against our proactive approach.

Our implementations were fully decoupled from the planner to avoid any planner specific optimizations associated with more direct code integration. PDDL files for operator and problem specification formed to serve as input for invoking the LPG-td executable; any resultant solution (plan) files generated where then detected and – if so – loaded and parsed to form an ASL format plan for use within our agent code.

Restriction to a single – but domain independent – planner allowed our experimentation to concentrate upon investigating differences arising from variance in planning *context* (i.e. with proactive versus reactive invocation), rather than differences between planners invoked under the same context (i.e. examining a multiplicity of planners for

each approach). Additionally, inter-planner comparison would (arguably) have only suggested optimal planner selection for each approach within the *specific Cargoworld* environment, rather than provided further information to compare the benefits – or drawbacks – of CAMP-BDI against reactive approaches *in general*.

We do not believe an alternate runtime planner would significantly impact our results, due to both the planner-agnosticism of CAMP-BDI, the decoupling of experimental implementations from any possibility of planner-specific optimizations and the generality of LPG-td itself. A library of pre-formed recovery or preventative plans *could* instead be employed for general efficiency – in which case significant differences would likely emerge from the designer's ability to anticipate ante or post-failure robustness scenarios. This would represent the outcome of the system designers anticipation of scenarios, however, rather than the efficacy of the particular (proactive or reactive) robustness approach.

We did not compare the quality of the plans generated by each approach. Such metrics are primarily used to compare the relative performance of different planners, by allowing comparison of which provides the 'best' plan when given the same planning problem. However, our approach is posited as an alternative to reactive methods – meaning inherent and intentional differences in the planning problems generated which prevent like-for-like comparison between generated plans from these approaches. In this context, quality based metrics would serve more to examine a specific implementation, rather for considering more general properties of proactive versus reactive approaches – particularly as our CAMP-BDI design is designed as planner agnostics.

Measuring the generated plan quality would also be of reduced value due to our robustness concern; any advantage in plan quality held by a particular approach could be outweighed by the impact of further failure, as further $\texttt{plan}_\texttt{i}$ revisions (whether reactive or proactive) removing unexecuted parts of that generated plan. In this context, superior *individual* plan quality (i.e. for $plan_i$ modifications by the robustness process) could be outweighed by overall costs from activity failure and backtracking costs or potential $goal_i$ failure.

Instead, our gathered metrics (Section 11.2.2) examine activity cost per goal achieved – to consider the actual activities *executed* and factor in costs from unsuccessful goal

pursuit. We also evaluated planner invocations and planner execution time to assess whether CAMP-BDI risks excessive computational cost with pre-emptive planning (but with consideration of robustness benefits); albeit noting the latter will provide only a general indication, with values specific to LPG-td.

11.1.2.1 Systems under comparison

The following agent robustness approaches were implemented and evaluated:

- **CAMP-BDI**: Our proactive robustness approach, presented (i.e. implementing the supporting architecture and algorithms) in the preceding Chapters.
- **Replanning**: Following activity failure, agents form (taking the post failure state as the initial state specification) and insert a new $plan_i$ for the relevant $goal_i$. The activity (and corresponding intention) is only regarded as failed if no plan could be found.
- **Continual planning**: Agents attempt to reform $plan_i$ after every activity completes regardless of whether it succeeded or failed; failure conditions are the same as for *Replanning*.
- **None**: Agents had *no* failure mitigation strategy.

Activity execution (from the agent perspective) covers both that of a primitive activity within the environment, and of a delegated activity (seen as primitive from the point of view of the dependant, and composite by the obligant). *Continual Replanning* agents would only replan for a dependant $plan_i$ upon a notification of a delegated activity completing (successfully or otherwise) from an obligant; *Replanning* agents would similarly wait for an obligant to report failure before performing any local replanning.

As replanning is performed on a local $plan_i$ basis, this results in a distributed repair strategy; dependants only replan upon notification of failure from an obligant, limiting distributed plan changes to the replanning agent and any direct or indirect obligants. This bears similarities to *FF-Replan*'s approach (Yoon *et al.* [2007]); preconditions for activities are effectively a single-outcome determinization, having been selected on significance criteria (i.e. reflecting the qualification problem defined by McCarthy [1958]) – making non-deterministic failure similar to the divergence from expected outcome handled by FF-Replan performing replanning[3].

[3] *LPG-td* employs an algorithm similar to FF (Hoffmann [2001]), reinforcing this similarity.

At an agent team level, individual reactive replanning lead to distributed repair behaviour through cascading assumption of responsibility by dependants (where obligants failed in local replanning) – especially as the distributed plan can be viewed as hierarchical, with obligant plan formation equivalent to task decomposition (i.e. for obligation activities). The resulting MAS-level behaviour was seen as offering the most efficient reactive approach in planner invocation count and activity cost; our experimental observations of agent activity within *Cargoworld* led us to conclude local plan repair was unlikely to offer benefits over total replanning, as the likely physical activity failures (i.e. for movement or movement enabling activities) would typically impact multiple future $plan_i$ activities and eventually require repair approaches reconsider the entire plan regardless.

This meant the key benefit of local repair – namely plan stability – was unlikely to be observed, with instead potential for unnecessary extra planning operations as part of incrementally increasing planning scopes. Consequently, we omitted evaluation of individual agent plan repair, reasoning that reactive replanning would likely entail the same eventual outcome but without excess planning operations from attempting change minimization – and therefore would represent the optimal reactive approach within our experimental domain.

We implemented two modalities for forming operator specifications and performing planning for evaluating CAMP-BDI, using different modalities of *LPG-td*. In both implementations capability knowledge (i.e. signature, preconditions and effects) was employed to form operator specifications:

- **CAMP-BDI.*Speed***(*CAMP-BDI.Spd*): Operator preconditions were defined as those states offering confidence equal or greater to the *Th* field within that capabilities' associated policy (i.e. the policy effectively defined the significance values for forming preconditions). Planning was performed using the *LPG-td.speed* modality, which returned the first plan found. Figure 11.4 gives an example of a generated operator specification for a *Truck*'s *move* activity.
- **CAMP-BDI.*Quality***(*CAMP-BDI.Qty*): We adopted the pseudo-probabilistic planning approach employed by *PACPlan* (Jiménez *et al.* [2006a]); each operator was given a metric cost effect corresponding to the inverse log of the confidence

were those preconditions to hold (i.e. $cost = -log(conf)$), as in Figure 11.5. Multiple operators were formed for each capability, for different precondition combinations and associated confidences. Planning was performed using *LPG-td.quality*, which attempted to minimize the *cost* metric by forming increasingly lower-cost plans, until a maximum number of plans were found or a set time limit exceeded. These limits were set to 20 plans and 0.5 seconds execution time for experimentation; initial testing suggested these values were sufficient to ensure the optimal possible plan would be found.

```
(:action OP_1_move
        :parameters (?AGENT ?RID ?O ?J)
        :precondition
        (and (truck1 ?AGENT) (connection ?RID)
        (location ?J) (location ?O)
        (road ?RID ?O ?J) (healthy ?AGENT)
        (not (blocked ?O ?J)) (not (toxic ?O ?J))
         (not (dangerZone ?O)) (not (dangerZone ?J))
        (not (stuck ?AGENT)) (onR ?AGENT ?RID)
        (dry ?O ?J))
        :effect
        (and (atJ ?AGENT ?J) (not (onR ?AGENT ?RID))
        (not (atJ ?AGENT ?O)) )
)
```

Figure 11.4: Example PDDL operator from *CAMP-BDI.Speed* for a *move* activity, where agent damage or a slippery road state would reduce confidence below the defined threshold.

```
(:action OP_1_move
 :parameters (?AGENT ?RID ?O ?J)
 :precondition
  (and (truck3 ?AGENT) (connection ?RID)
  (location ?J) (location ?O) (road ?RID ?O ?J)
  (healthy ?AGENT) (not (blocked ?O ?J))
  (not (toxic ?O ?J)) (not (dangerZone ?O))
  (not (dangerZone ?J)) (not (stuck ?AGENT))
  (onR ?AGENT ?RID) (dry ?O ?J))
 :effect
  (and  (atJ ?AGENT ?J) (not (onR ?AGENT ?RID))
  (not (atJ ?AGENT ?O)) (increase (total-cost) 0.01) )
)

(:action OP_8_move
 :parameters (?AGENT ?RID ?O ?J)
 :precondition
  (and (truck3 ?AGENT) (connection ?RID)
  (location ?J) (location ?O) (road ?RID ?O ?J)
  (damaged ?AGENT) (not (blocked ?O ?J))
  (not (toxic ?O ?J)) (not (dangerZone ?O))
  (not (dangerZone ?J)) (not (stuck ?AGENT))
  (atJ ?AGENT ?O) (slippery ?O ?J) (tarmac ?O ?J))
 :effect
  (and  (atJ ?AGENT ?J) (not (onR ?AGENT ?RID))
  (not (atJ ?AGENT ?O)) (increase (total-cost) 7.1) )
)
```

Figure 11.5: Example PDDL operators for *Truck3*'s *move* activity, showing the confidence costs for different preconditions. *OP_1_move* offers 100% confidence where agents are healthy and the road is dry; *OP_8_move* offers reduced confidence (increased cost) where the agent is damaged and the road is slippery tarmac. Costs were multiplied from $-log(confidence)$ values due to precision issues observed with *LPG-td.quality*.

CAMP-BDI.*Speed* offered a guarantee that plans, if formed, would have sufficient confidence – but at the cost of tightly constraining preconditions. For CAMP-BDI.*Quality*, pseudo-probabilistic planning offered greater flexibility – agents could form *better* plans than the maintained activity, even if not all activities within the generated plans were above the required confidence level. This allowed iterative improvement; for example, preventing failure in immediate activities, with later maintenance addressing any later low-confidence activities introduced by the maintenance plan. In both systems, plan confidence estimation used a weighted average approach (Section 8.2.6.2); this had little impact on plan acceptance in the *Speed* case (due to already constrained preconditions), but enabled acceptance of plans in the *Quality* case (provided they improved upon the current $plan_i$). Primitive capability estimation was implemented using prior domain knowledge, and reflected our assumptions that such estimation was accurate.

Finally, the *None* system covered where agents possessed no proactive or reactive failure mitigation strategy; providing a worst-case goal achievement baseline, allowing comparison of relative difficulty for experimental configurations, and confirming the necessity of a robustness strategy. We note differences in agent behaviour would result in divergences in terms of environmental changes from agent activity; meaning our results will be somewhat approximated by this fact.

11.1.2.2 MAS design

We designed and implemented agents following the description presented in Section 7.2. The same *AgentSpeak* (ASL) files were employed in all cases; compared agent systems only varied in terms of robustness behaviour. Figure 11.6 gives agent types within the MAS and their individual capability sets. We did not specify a MAS organization (i.e. authority constraints between agents) – however, a *meta-organization* would emerge for each system goal through dependency formation, guided by the constraints upon capability advertisements between agents.

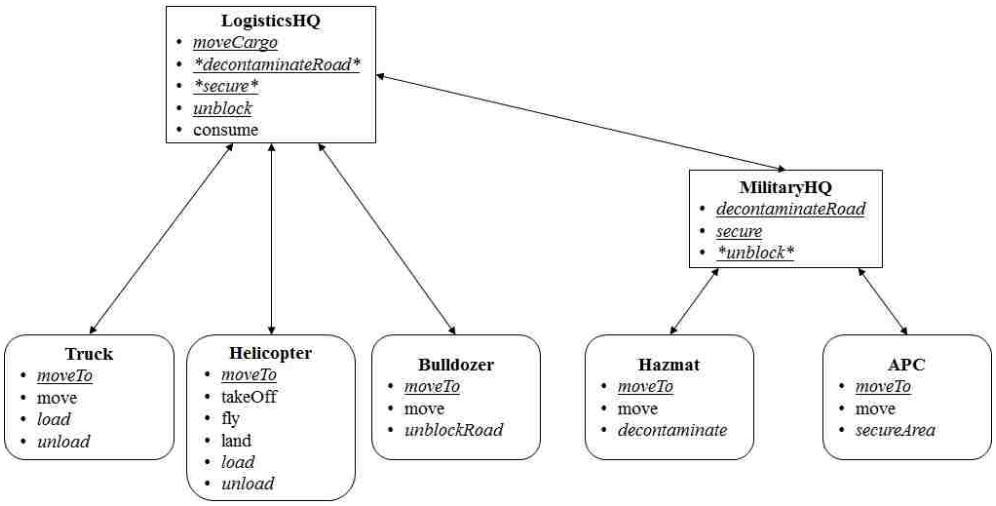


Figure 11.6: Diagram showing capabilities held by agents in the MAS; composite capabilities denoted by *italics*, whilst those advertised (i.e. forming other agent's external capabilities) are underlined. Figure 7.3 in Chapter 7 previously indicated the inter-agent visibility of advertisements. An asterisk (*) denotes capabilities where the agent performs a broker role; i.e. does not provide that capability internally, but provides functionality through use of it's own external capabilities. Physical agents are shown by rounded edges.

Although the simulated *Cargoworld* environment was arguably simple enough (to avoid details of *agent* behaviour being obfuscated by domain complexity) to simply use a single central agent, we designed our MAS to provide realistic meta-organizational hierarchies such as the *Strategic-Tactical-Operational* structure (Killion [2000]). The restriction of capability visibility reflects a typical motivation for a MAS approach, where distribution of knowledge and ability required agent co-operation and co-ordination to achieve system goals.

Three (types of) agent roles can be defined for an executing multiagent plan; root (those with dependencies only), middle (holding both dependencies and obligations) and leaf agents (holding only obligations). *LogisticsHQ* and *MilitaryHQ* represented logical agents – only able to achieve environmental change through delegation – and performed roles as both middle and broker agents by allowing formation of indirect dependencies between physical agents (which did not have peer-to-peer visibility for capability advertisements). For example, to perform a *moveTo* activity, a *Truck* agent

may for a dependency upon the *unblock* capability of *LogisticsHQ*; *LogisticsHQ* would consequently select and form a dependency upon a *Bulldozer* (holding the relevant physical capability) – meaning *LogisticsHQ* held root *and* middle agent roles (respectively) for *moveTo* and *unblock*.

Our experimental MAS was defined with two logical (one *LogisticsHQ* and one *MilitaryHQ*) and ten physical agent types (three *Truck*, one *Helicopter*, two *Bulldozer*, two *APC* and two *Hazmat* agents). The physical agent count was selected to provide logical agents with sufficient options to allow meaningful planning decisions, but without levels of redundancy that would effectively negate any agent loss from post-failure debilitation.

11.1.2.3 Maintenance Policies

Maintenance policies were implemented with default field values (Figure 11.3) and applied across all agents and capabilities. Policies associated with *Helicopter* flight capabilities inserted an additional '*windy*' state within *DC*, as this state could not be counteracted through maintenance. As our focus was on *application* within maintenance rather than communication of policies, we did not implement a dedicated policy service.

Field	**Value**
Threshold (*Th*)	0.95
TriggerConditions (*TC*)	$\emptyset$
Priority (*Pr*)	*Normal*/*Default*
DropConditions (*DC*)	{'*damaged*', '*mortal*', '*resting*'}

Table 11.3: Maintenance policy field values used during experimentation.

Maintenance policy values were defined using our knowledge of the *Cargoworld* (as were agent capabilities). *Th* was defined by estimating weighted averages for a sample four activity plan, and set such that a generated plan would be accepted if *at least* the first two activities had an acceptable confidence – i.e. where the agent was not '*damaged*', the road used (for road vehicle movement) was not '*slippery*' or conditions (if flying) were not '*windy*'. The *DC* represented those states that reduced

confidence, but which could not be counteracted by any agent capability – meaning no maintenance plan could exist to counteract them. *Th* values encompassed any states that would otherwise be present in *TC*, meaning *TC* could be left an empty set.

11.2 Experimental Design

The core hypothesis behind this work is that:
'*In realistic environments where failure risks debilitative consequences, a proactive approach of pre-emptive plan modification can improve robustness over a purely reactive approach*'.

In Section 4.1, we defined robustness and concluded that:
'*The efficacy of our approach is to be measured through goal achievement rate under perturbation; the latter defined as the rate of exogenous change.*'

This follows the definition of Hahn *et al.* [2003] of robustness as *'graceful degradation of performance under perturbation'*; which we interpret as meaning that, if our hypothesis holds, our approach should provide consistently higher goal achievement rates than reactive alternatives under increasing perturbation. This enabled comparison of proactive and reactive types of approaches, unlike more semantic dependant definitions such as activity success rate.

A number of secondary hypotheses were also formed within our design process, which can be evaluated as corollaries of the core hypothesis by examining whether the resultant CAMP-BDI design improved robustness:

- '*Agents can be embodied with capability knowledge to represent both those activities they can perform themselves, and those which they are dependent upon others to perform.*'
- *'The resultant capability model can be used to intelligently determine when plan failure is threatened, and to direct consequent mitigation behaviour.'*
- '*This local behaviour can be designed such that decentralized, distributed maintenance behaviour is achieved on an agent-team level through use of dependency contracts and appropriate co-ordination messaging.*'
- '*Policies – sets of behavioural constraints, applied to sets of agent-capability pairs – can be used to tailor agent maintenance behaviour during runtime, al-*

lowing a degree of adaptation to changing knowledge of the agent and environment.'

11.2.1 Experimental Geographies

Our experimental evaluation used two *Cargoworld* geographies, with different levels of complexity (Figure 11.7 and Figure 11.8). Geographies were created by the *Cargoworld* simulator through a procedural algorithm, using a specified seed value and set of constraints (maximum and minimum junction count, airport count, and the maximum number of road connections per junction). The generated road network was required to be fully interconnected, with every junction reachable from every other (excluding environmental states limiting road use). Initial world state was generated by simulating 500 exogenous change steps before the first cargo request was generated, and based upon the experimental parameters given in Section 11.3.

The primary factor influencing difficulty was the length of agent travel route - greater numbers of individual movement activities entailed greater potential exposure to the risks associated with both exogenous change or existing debilitated states. This route length was determined by both the number of junctions present (i.e. geography size) and their connectivity. Lower connectivity entailed less direct routes, increasing both activity length and the likelihood of backtracking being required when needing to form an alternate route from a given location; i.e. while a mesh-like geography offered near immediate options for an immediate change of route, limited connectivity increased the risk of the agent being partially down a linear or low-branching path at the point of replanning or repair.

We also used definition of *Airports*, present at only a limited number of junctions, to constrain the point-to-point movement possible for *Helicopters*. This avoided over-reliance upon these vehicle types, and prevented the impact of geographical complexity being wholly overridden by use of direct air travel. This also allowed alternate cargo transport approaches, including use of heterogeneous agent types where a *Helicopter* performed *partial* transport between intermediate locations (i.e. where *Truck* agents were employed to transport *Cargo* to and from locations inaccessible via *Helicopter*).

Cargo objects and requests were initially generated by our simulator at maximally

distant (but accessible) locations; this helped ensure differences in geography represented meaningful differences in difficulty. The alternative – use of wholly random generation – could potentially place cargo at an initial location immediately adjacent to the desired delivery location, negating our use of geographic size and complexity to define and gauge relative environmental difficulty.

Our first experimental geography – **World A** (Figure 11.7) – represented a relatively simple environment; routes (and consequent plans) between junctions were relatively short. World A also had greater interconnectivity (more connections between junctions), increasing options available for agent route planning.

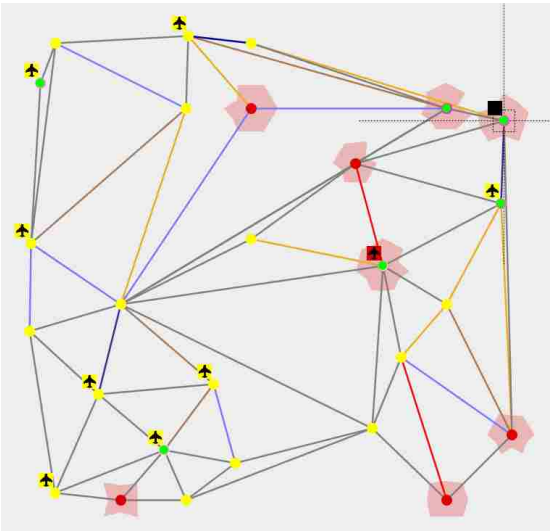


Figure 11.7: Screenshots showing geography of World **A**, with 27 junctions, 8 airports (used by *Helicopter* agents) and 63 roads.

World B (Figure 11.8) presented a more complex geography with increased junctions. Reduced interconnectivity compared to (the mesh-like) World A increased the risk of backtracking being required to mitigate failures (anticipated or actual) in route plans. Finally, a larger geography increased the risk of exogenous change impacting a given intention, by increasing the average route plan length and the average number of activities (potential exogenous change steps) required to achieve a goal.

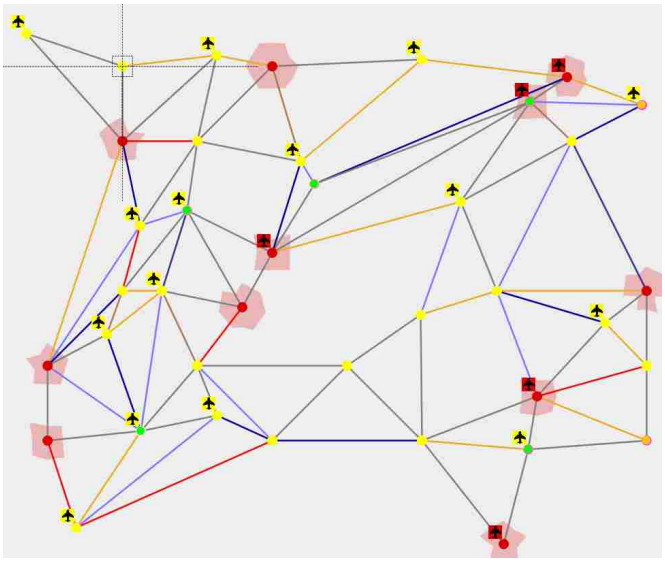


Figure 11.8: World **B** geography with 39 junctions, 16 airports, and 83 roads.

11.2.2 Key Metrics

Metrics were recorded and averaged from results of all experimental runs for each approach, in the relevant experimental configuration. Our core metric for comparing robustness was the *Delivery Success Rate* (Equation 11.1); this gave the percentage of cargo requests satisfied and, consequently, overall achievement of MAS system goals.

$$\textit{Delivery Success Rate} = \left(\frac{\textit{Total Deliveries Made}}{\textit{Total Delivery Requests}} \right) * 100 \tag{11.1}$$

The *Activity Success Rate* (Equation 11.2) gave the percentage of successful activities executed and was used to confirm the efficacy of *CAMP-BDI* in avoiding activity failure; this is expected to be lower for reactive systems, which respond *after* failure.

$$\textit{Activity Success Rate} = \left(\frac{\textit{Activities Succeeded}}{\textit{Activities Failed}} \right) * 100 \tag{11.2}$$

The primary aim of CAMP-BDI is to improve goal achievement – reflected in the *Delivery Success Rate* metric. However, additional metrics were gathered to analyze relative performance in terms of activity, planning and messaging costs.

A proactive method risks executing additional activities to address false positive anticipation of threats – but a reactive method may also risk additional costs to recover

from failure, such as backtracking or from enlisting alternate obligants. We measured *Average Delivery Cost* as the average number of activities executed per goal achieved, to factor in both the efficiency of goal achievement *and* costs where robustness behaviour was unable to prevent failure.

$$\textit{Average Delivery Cost} = \left(\frac{\textit{Total Activities Performed}}{\textit{Total Goals Achieved}} \right) \tag{11.3}$$

We did not directly compare the quality of generated plans, as the evaluated approaches performed planning under different contexts and with different (in the case of *CAMP-BDI* in particular) scopes of goal. This made direct comparison inappropriate, particularly as it would arguably serve to evaluate the performance of *LPG-td* rather than either proactive or reactive implementations. The average delivery cost can be considered as holding an indicative value through including the activities added through insertion of maintenance plans, or by replanning.

Several metrics were gathered to consider differences in planning cost. *Average Planning Operations Per Delivery* (Equation 11.4) was given as the total number of individual plan generation operations (successful or otherwise), divided by the total goals achieved.

$$\textit{Average Planning Operations Per Delivery} = \left(\frac{\textit{Total Planner Operations}}{\textit{Total Goals Achieved}} \right) \tag{11.4}$$

Average Planning Time Per Delivery (Equation 11.5) provided the average execution time in nanoseconds for each individual call to the *LPG-td* planner, used to indicate the approximate *complexity* of planning tasks generated by each approach. The values given by this metric were partially specific to the *LPG-td* implementation, but we suggest still held indicative value.

$$\textit{Average Planning Time} = \left(\frac{\textit{Accumulated Planning Time}}{\textit{Total Planner calls}} \right) \tag{11.5}$$

Our final metric (Equation 11.6) gave average messaging costs per goal. This was used to consider the additional costs of communicating contract updates in *CAMP-BDI*, and to compare relative stability between approaches by measuring the volume of messages concerned with forming, cancelling or modifying inter-agent dependencies.

$$Average\ Messages\ Per\ Delivery = \left(\frac{Total\ Message\ Count}{Total\ Goals\ Achieved}\right) \tag{11.6}$$

The *Total Message Count* was formed through recording and summation of the following message types:

- **dependencyCancel**: sent to cancel a dependency, whether as an outcome of plan failure or plan modification/replanning.
- **confirmContractWithObligant**: sent to confirm the formation of an obligation.
- **obligationMaintained**: sent by an obligant to inform the dependant it has completed maintenance, including any resultant contract changes.
- **dependencyMaintained**: sent by a dependant to obligants when maintenance impacts an/the obligation held by the latter – e.g. to update the contract's *CL* field following changes to preceding activities in the dependant $plan_i$.
- **updatedContract**: conveys changes in meta-knowledge regarding how dependencies will (are expected to) be performed.

Of these, the first two were common to all approaches; the latter three specific to CAMP-BDI. We selected these message types to evaluate how proactive behaviour incurred costs from additional contract updates (as extra information is required for threat anticipation), and whether there was a reduction in contract formation and cancellation messages for our repair-orientated strategy.

A separate metric *excluding* the **updatedContract** type was gathered for *CAMP-BDI* to evaluate stability (i.e. giving solely messaging for dependency contract formation, modification or cancellation). The *size* of messages sent was not recorded, as this was judged as implementation dependant, with optimization lying outside our investigative scope. We similarly considered optimization of *CAMP-BDI* **updatedContract** messaging cost as outside the scope of this thesis; our recorded results likely represent worst-case scenarios.

11.2.3 Experimental Protocol

Experiments were ran under different parameter configurations for both exogenous change and the likelihood of post-failure debilitation. All experiments were performed on a machine with an Intel i5-3750k processor, 16GB RAM and – to avoid any signif-

icant read/write latency – used a Solid State Drive (SSD). All code was implemented in Java, and ran with version 1.80.30 of the Java Virtual Machine.

To improve reproducibility, independent pre-defined seed values controlled exogenous event generation and the occurrence of post-failure debilitation; although the different robustness behaviours of each approach would still lead a degree of divergence in how the world state evolved with agent activity, even when using the same seed values. Each individual experimental run (i.e. the period of operation for a MAS employing the particular approach under evaluation) lasted for the generation of 100 delivery requests (i.e. for 100 – successful or failed – system goals). We judged that this limit was short enough to prevent excessive divergence (and less appropriate comparability) between approaches, but sufficient to allow gathering of data (allowing detection of longer-term instabilities or performance degradation from accumulated failure consequences).

For each parameter configuration tested, each approach (employed by a MAS following the design given in Section 11.1.2.2) was run for ten different seed values, and repeated six times for each seed. We used these repeats to account for microsecond level differences in agent reaction to belief updates or message receipt; due to the asynchronous nature of agent behaviour, such differences could still impact the outcome of experimental runs. We evaluated performance in two geographies ($n_{geo} = 2$), for combinations of perturbation ($n_{perturbation} = 3$) and debilitative risk ($n_{risk} = 4$); giving $n_{config} = n_{geo} \times n_{perturbation} \times n_{risk} = 24$ individual configurations. Each of four approaches ($n_{system} = 4$ was ran sixty times (six times for each of ten seed values; $n_{runs} = 60$) per environment configuration (giving a $n_{system} \times n_{runs} \times n_{config} = 5,760$ experiments). Finally, we ran *CAMP-BDI.Quality* in the highest perturbation configuration for World A and World B ($n_{quality} = n_{geo} \times n_{risk} \times n_{runs} = 480$) – performing an overall total of 6,240 individual experimental runs.

11.3 Experimental Parameters

We evaluated performance under scaled levels of both perturbation and risk of post-failure debilitation, controlled using the parameters defined in Figure 11.1 and Figure 11.2. This allowed assessment of performance under '*increasing levels of perturbation*' (Hahn *et al.* [2003]). Perturbation was scaled through $n_{exo} = \{1,2,3\}$. We

evaluated four levels of perturbation risk – $n_{risk} = \{0, 0.25, 0.5, 0.75\}$ for each value of n_{exo}. The initial values of parameters were set using initial experimentation with the *None* system; we selected a configuration identified as suitably difficult – to offer the possibility for improvement with robustness behaviour, but still allow scaling to a greater difficulty – we reduced variables to use a common denominator (i.e. providing the configuration for $n_{exo} = 1$).

Parameter	Value
windyChangeChance	$0.125n_{exo}$
closeRoadChance	$0.05n_{exo}$
openRoadChance	$0.1n_{exo}$
addDzChance	$0.1n_{exo}$
removeDzChance	$0.05n_{exo}$
maxRoadClosuresPerStep	$0.015n_{exo}$
maxDzPerStep	$0.03n_{exo}$
chanceOfFlooding	1
chanceOfDrying	0.7
maxFloodPerTurn	$0.06n_{exo}$
maxDryPerTurn	$0.06n_{exo}$
maxFlooded	$0.1n_{exo}$
maxSlippery	$0.1n_{exo}$
cargoDestroyChance	n_{risk}
cargoSpillChance	n_{risk}
chanceOfPostFailureDamage	n_{risk}
stuckChance	n_{risk}

Figure 11.9: Experimental parameters, where $n_{exo} = \{1, 2, 3\}$ and $n_{risk} = \{0, 0.25, 0.5, 0.75\}$ scale probabilities of exogenous change and debilitative consequence. Values for **chanceOfFlooding** and **chanceOfDrying** were fixed (favouring incremental addition of risk from slippery or flood states), with the flooding and drying related parameters effectively defining the rate of change.

11.4 Summary

This chapter described our experimental protocol, including configuration and use of the *Cargoworld* domain. The following chapter presents our experimental results for each evaluated maintenance approach, within World A and B geographies for progressively scaled $n_{exo} = \{1,2,3\}$ and $n_{risk} = \{0,0.25,0.5,0.75\}$[4].

[4] Appendix A gives example screenshots showing the state of each environment and n_{exo} configuration at the start of execution.

Chapter 12

Experimental Results

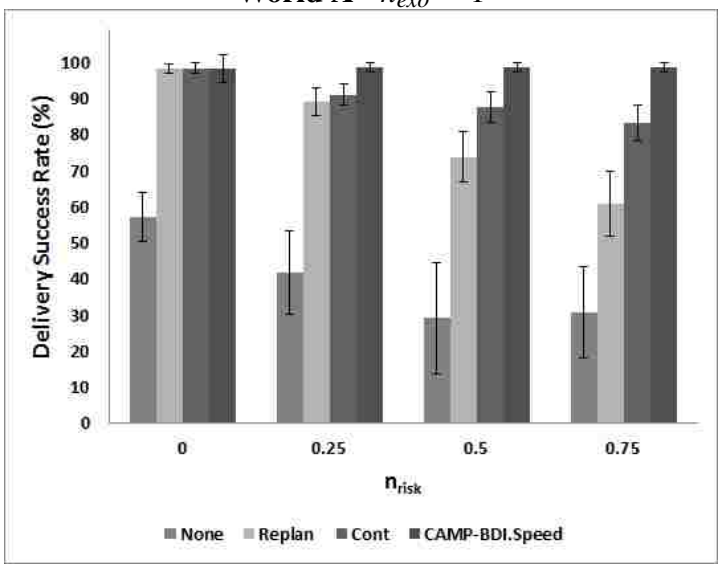
The following sections present our experimental results for World A and B over progressively scaled probabilities of perturbation and post-failure debilitation ($\mathbf{n_{exo}} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$ and $\mathbf{n_{risk}} = \{\mathbf{0}, \mathbf{0.25}, \mathbf{0.5}, \mathbf{0.75}\}$). *None*, *Replanning*, *Continual Replanning* and *CAMP-BDI.Speed* approaches were evaluated for all n_{exo}, with *CAMP-BDI.Quality* also evaluated in the highest difficulty perturbation configuration ($n_{exo} = 3$).

Our results show *CAMP-BDI* had a significant goal achievement advantage over *Continual Replanning* and *Replanning* results, especially as n_{exo} and n_{risk} increased. CAMP-BDI additionally had lower per-goal activity and planning costs, with lower messaging costs (for messages concerned with dependency formation or cancellation) indicating CAMP-BDI approaches offered greater plan stability.

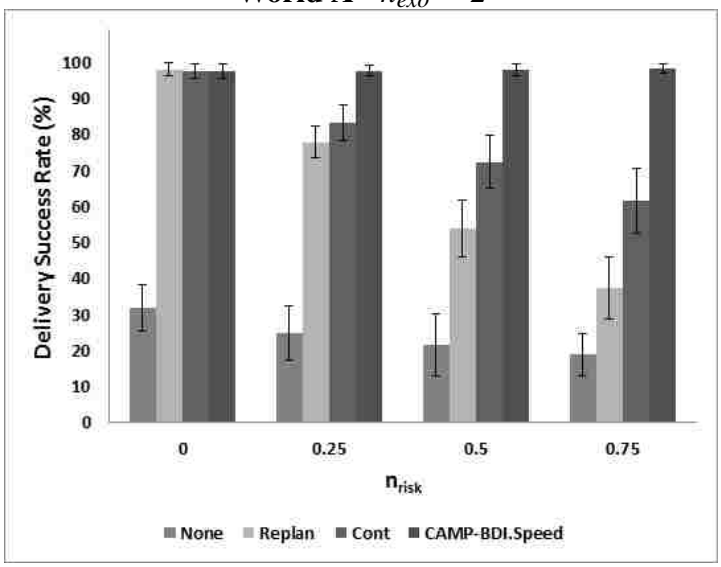
12.1 Delivery Success Rate

The average success rates of all approaches for all n_{exo} and n_{risk}, are given in Figure 12.1 for World A and Figure 12.2 for World B. Figure 12.3 and Figure 12.4 give differences between *CAMP-BDI.Speed* and other approaches for all n_{exo} and n_{risk}. As we observed reduced performance from *CAMP-BDI.Speed* in World B $n_{exo} = 3$, additional *CAMP-BDI.Quality* runs were performed to determine if the looser precondition constraints of *CAMP-BDI.Quality*'s pseudo-probabilistic planning approach offered benefits for maintenance in this configuration.

World A - $n_{exo} = 1$



World A - $n_{exo} = 2$



World A - $n_{exo} = 3$

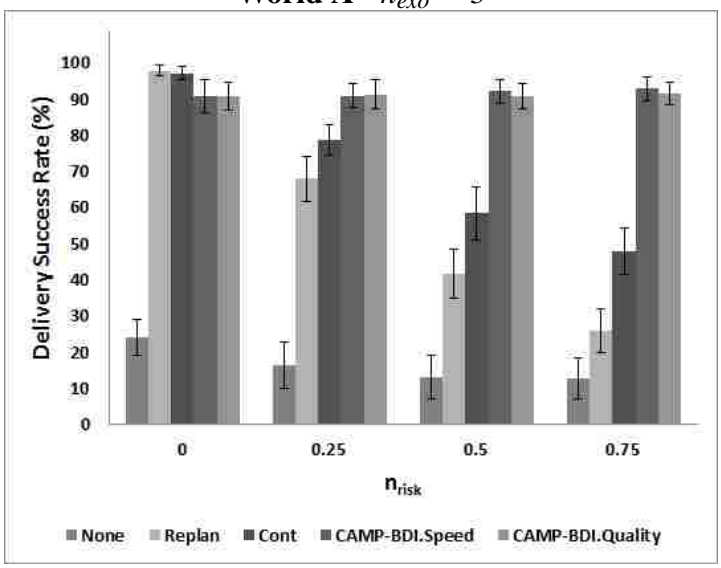
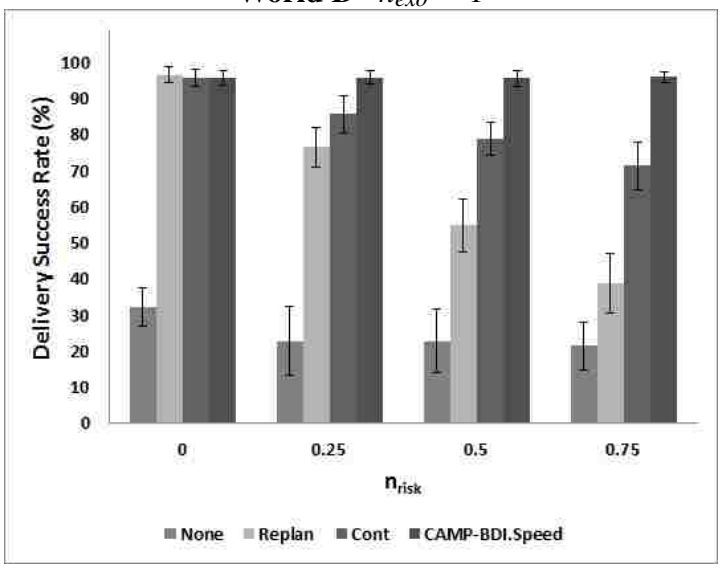
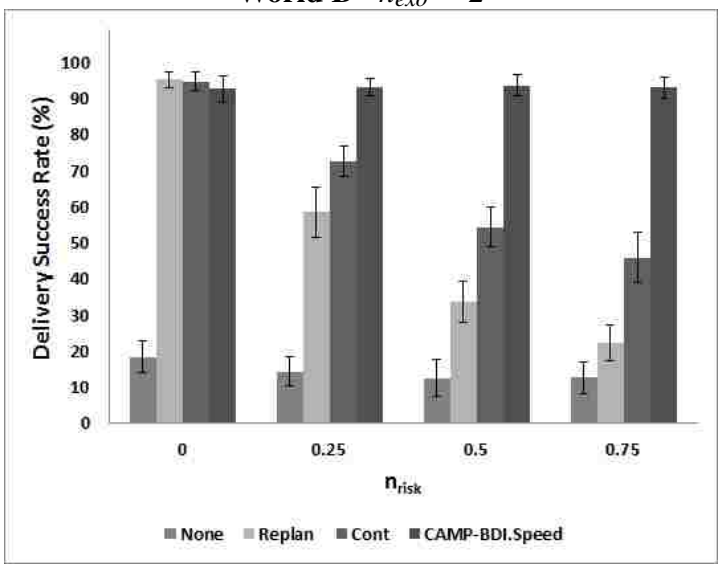


Figure 12.1: Delivery Success Rate in World A; exact numerical values for this and all other graphs within this chapter are given in Appendix B

World B - $n_{exo} = 1$



World B - $n_{exo} = 2$



World B - $n_{exo} = 3$

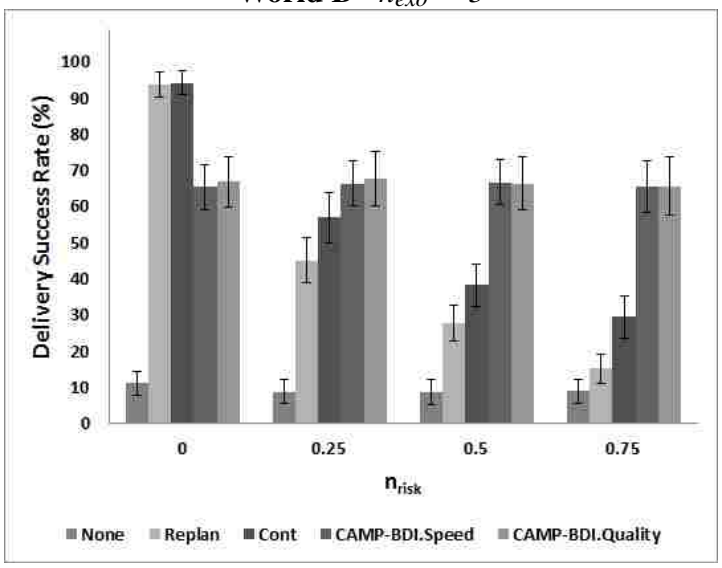


Figure 12.2: Delivery Success Rate in World B

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+49.983 ($1.55x10^{-50}$)	+56.967 ($5.65x10^{-43}$)	+59 ($2.46x10^{-40}$)	+67.8 ($4.01x10^{-45}$)
Replanning	+0.1 (0.863)	+9.55 ($3.01x10^{-27}$)	+26.233 ($6.61x10^{-35}$)	+37.783 ($3.5x10^{-39}$)
Continual Replanning	-0.117 (0.834)	+7.517 ($4.83x10^{-26}$)	+11.083 ($3.74x10^{-26}$)	+15.417 ($1.4x10^{-30}$)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+65.65 ($5.06x10^{-66}$)	+72.717 ($1.56x10^{-59}$)	+76.217 ($2.33x10^{-57}$)	+79.117 ($1.79x10^{-68}$)
Replanning	-0.45 (0.224)	+19.75 ($1.83x10^{-38}$)	+43.95 ($5.09x10^{-46}$)	+60.717 ($1.48x10^{-51}$)
Continual Replanning	+0.1 (0.773)	+14.433 ($3.17x10^{-29}$)	+25.65 ($3.69x10^{-33}$)	+36.567 ($2.06x10^{-38}$)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+66.967 ($1.7x10^{-60}$)	+74.55 ($3.25x10^{-61}$)	+79.017 ($9.34x10^{-64}$)	+80.317 ($1.33x10^{-66}$)
Replanning	-6.9 ($1.34x10^{-15}$)	+22.967 ($1.26x10^{-32}$)	+50.4 ($1.01x10^{-51}$)	+66.867 ($1.62x10^{-61}$)
Continual Replanning	-6.267 ($5.19x10^{-14}$)	+14.117 ($3.6x10^{-27}$)	+33.65 ($1.08x10^{-36}$)	+45 ($3.994x10^{-50}$)
CAMP-BDI Quality	+0.033 (0.965)	-0.433 (0.482)	+1.35 (0.036)	+1.233 (0.031)

Figure 12.3: Differences (p in brackets) in average delivery success rate between *CAMP-BDI.Speed* and other approaches in World A, showing that *CAMP-BDI.Speed* achieved more goals than *Continual Replanning* and *Replanning* where $n_{risk} \geq 0.25$, with that advantage increasing with n_{risk}. Each row corresponds to an approach compared to *CAMP-BDI.Speed*, with the columns giving the difference under increasing (left to right) n_{risk}. Positive values show that *CAMP-BDI.Speed* achieved more goals on average (had superior robustness).

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+63.25 ($1.93x10^{-63}$)	+73.017 ($1.39x10^{-53}$)	+72.8 ($2.94x10^{-54}$)	+74.4 ($8.1x10^{-64}$)
Replanning	-0.983 (0.009)	+19.267 ($1.06x10^{-34}$)	+40.733 (1.08^{-45})	+56.967 ($1.95x10^{-52}$)
Continual Replanning	-0.167 (0.707)	+10.183 ($9.61x10^{-21}$)	+18.65 ($5.91x10^{-35}$)	+24.583 ($3.76x10^{-36}$)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+74.233 ($1.23x10^{-68}$)	+78.583 ($3x10^{-74}$)	+80.95 ($8.51x10^{-69}$)	+80.317 ($5.84x10^{-71}$)
Replanning	-2.55 ($4.54x10^{-5}$)	+34.533 ($2.07x10^{-42}$)	+59.95 ($8.51x10^{-60}$)	+70.6 ($2.05x10^{-64}$)
Continual Replanning	-2.083 (0.001)	+20.45 ($1.92x10^{-42}$)	+39.15 ($4.05x10^{-49}$)	+47.033 ($7.56x10^{-50}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+54.3 ($5.85x10^{-56}$)	+57.583 ($4.16x10^{-57}$)	+58.05 ($2.23x10^{-58}$)	+56.65 ($1.3x10^{-52}$)
Replanning	-28.367 ($4.98x10^{-40}$)	+21.267 ($2.36x10^{-26}$)	+44.067 ($5.4x10^{-44}$)	+50.367 ($2.25x10^{-49}$)
Continual Replanning	-28.833 ($2.32x10^{-36}$)	+9.417 ($1.23x10^{-10}$)	+28.5 ($1.76x10^{-33}$)	+36.1 ($5.78x10^{-39}$)
CAMP-BDI Quality	-1.45 (0.161)	-1.333 (0.303)	+0.383 (0.766)	-0.15 (0.911)

Figure 12.4: Differences (p in brackets) in average delivery success rate between *CAMP-BDI.Speed* and other approaches in World B. Positive values indicate *CAMP-BDI.Speed* achieved more goals on average; our results show greater goal achievement for *CAMP-BDI* over replanning approaches for $n_{risk} \geq 0.25$, with that advantage growing with increasing n_{risk}.

In both geographies, *None* had universally poor performance, worsening with greater n_{exo}. The differences in goal achievement over increasing n_{risk} for *None* were not so obvious (or, indeed present), and not always significant (Figure 12.5)[1]; in several cases, particularly within World B, n_{risk} did *not* have a significant impact on goal success. This indicated the primary influence was perturbation rather than debilitation; plans for *None* would fail on the first activity failure, regardless of any resultant debilitation, and largely negating the possibility of cumulative debilitation seen in reactive approaches. These *None* results primarily define a worst-case minima for performance, but also show the necessity of failure mitigation behaviour in that environment.

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-6.667 ($5.91x10^{-5}$)	-12.567 ($2.56x10^{-6}$)	+1.767 (0.491)
$n_{exo} = 2$	-7 ($1.58x10^{-9}$)	-3.267 (0.017)	-2.667 (0.058)
$n_{exo} = 3$	+7.633 ($4.74x10^{-10}$)	-3.3 (0.007)	-0.767 (0.632)

World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+9.45 ($1.95x10^{-8}$)	-0.1 (0.952)	-1.217 (0.449)
$n_{exo} = 2$	-3.967 ($2.38x10^{-5}$)	-1.85 (0.275)	+0.067 (0.94)
$n_{exo} = 3$	-2.267 (0.001)	-0.1 (0.873)	+0.167 (0.818)

Figure 12.5: Differences (p in brackets) for Average Delivery Success Rate for increasing n_{risk} for *None* in World A and B; column names show the n_{risk} values whose results are being compared. Negative values show that *None* achieved *less* goals at the higher n_{risk} than at the lower.

In World A for $n_{exo} = 3$ and World B for $n_{exo} = 2$ and 3, where $n_{risk} = 0$, *Continual Replanning* and *Replanning* held small but significant advantages in goal achievement over *CAMP-BDI.Speed* (Figure 12.3). *Replanning*, but not *Continual Planning* also held a significant advantage for $n_{exo} = 1$ in World B. For World B, $n_{exo} = 3$, a much larger difference was observed between these reactive approaches and both *CAMP-BDI* modalities (Figure 12.4). The uniformly high success of reactive approaches at $n_{risk} = 0$ stems from the impossibility of debilitation from failure – which would allow repetition of the same failed low confidence activity, with no failure penalty, until eventual success. Where replanning approaches *did* fail to achieve a

[1]We treated $p < 0.05$ as indicating a significant difference.

goal, we attribute this to the specific world state preventing any successful delivery – most likely due to the specific combination of flooded roads (a scenario which could not be reverted using any agent capability) preventing an agent travelling to a required location and performing an activity critical to success (for example, if no *APC* was able to secure some location whose use, or transit through, was required for successful cargo retrieval and delivery). In contrast, *CAMP-BDI* agents would not attempt recovery for activity failure, and were dependent upon the ability to anticipate and form viable maintenance plans *before* a threatened activity's execution.

Our results also show *CAMP-BDI* offering superior robustness to reactive approaches, where failure risked debilitation (i.e. at $n_{risk} \geq 0.25$), increasing as perturbation increased (i.e. increasing n_{exo}). These results shown that, as the risk of debilitation increased – either with the increasing risk for individual failures being associated with debilitation, or with exogenous change increasing the overall risk for, and number of activity, failures – reactive approaches faced increasing scenarios where both debilitation rendered recovery impossible *and* where the lasting impact of said debilitation impacted the success of future activity.

Where reactive approaches did show superior performance at $n_{risk} = 0$, these differences were – whilst statistically significant – relatively small. The sole exception was in World B $n_{exo} = 3$, where differences between CAMP-BDI approaches and reactive ones were much larger – however, once debilitative risk was introduced, the consistent goal achievement of both *CAMP-BDI.Speed* and *CAMP-BDI.Quality* offered superior robustness over *Continual Replanning* or *Replanning* approaches.

Those cases where *CAMP-BDI* was disadvantaged at $n_{risk} = 0$ are likely to be due to the absence of *any* greater confidence plan to be found by maintenance (of either modality); as a purely pre-emptive system, *CAMP-BDI* would see overall goal failure upon any failed activity (including for low-confidence cases), whilst reactive approaches could – as noted prior, and due to not being bound by debilitation consequences – continually form plans that served to repeat the failed low confidence activity until success.

With $n_{risk} \geq 0.25$, goal achievement for both *Replanning* and *Continual Replanning* decreased with increasing debilitation – for example, in World A $n_{exo} = 1$ *Replan-*

ning goal achievement dropped from 98.15% at $n_{risk} = 0$ to 60.82% at $n_{risk} = 0.75$, with *Continual Replanning* dropping from 98.38% to 83.18%. In the more difficult World B $n_{exo} = 3$, average goal achievement dropped from 93.83% to 15.25% for *Replanning* and 94.3% to 29.52% for *Continual Replanning*. These decreases were associated with the increasing risk of debilitative consequences from failure.

Continual Replanning did partially mitigate such risk (shown by consistently superior goal achievement to *Replanning*). This was due to the continual reformation of $plan_i$s, following every activity execution, and using current beliefs to specific the planning problem initial state; the *Continual Replanning* system attempted to form an optimal $plan_i$ for the $goal_i$ of the selected intention upon *every* reasoning cycle. This meant the reformed $plan_i$ would consequently contain only activities whose preconditions were expected to hold given current beliefs – implicitly removing any remaining activities in the previous $plan_i$ whose preconditions had been violated by exogenous change. In contrast, the purely reactive *Replanning* would only respond when failure actually occurred – a plan containing future threatened activities would remain unmodified until some failure occurred. However, both reactive and continual replanning methods remained vulnerable to *non-deterministic* failures, as a result of their planning operator preconditions excluding less significant, but still failure-risk inducing, states.

Unlike reactive approaches, no significant changes in goal achievement over increasing n_{risk} were observed for *CAMP-BDI.Speed* in World A or B (Figure 12.6), at any level of perturbation. This also applied for *CAMP-BDI.Quality* in $n_{exo} = 3$. These results show our maintenance approach mitigated the risk of post-failure debilitation, by avoiding activity failure(s) in the first place.

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+0.317 (0.565)	+0.167 (0.463)	-0.133 (0.551)
$n_{exo} = 2$	+0.067 (0.85)	+0.233 (0.45)	+0.233 (0.377)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.05 (0.964)	+1.233 (0.076)	+0.8 (0.244)
CAMP-BDI.Qty	+0.417 (0.596)	-0.55 (0.468)	+0.917 (0.16)

World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+0.317 (0.367)	-0.317 (0.4)	+0.383 (0.264)
$n_{exo} = 2$	+0.383 (0.471)	+0.517 (0.331)	-0.567 (0.317)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.05 (0.419)	+1.233 (0.77)	+0.8 (0.31)
CAMP-BDI.Qty	+0.417 (0.526)	-0.55 (0.324)	+0.917 (0.62)

Figure 12.6: Difference (p in brackets) for *CAMP-BDI* goal achievement in World A and B under progressive increase of n_{risk}, for all n_{exo}. Positive values show greater goal achievement at the higher n_{risk}.

The notable decrease in goal achievement for *CAMP-BDI* in World B at $n_{exo} = 3$ led to consideration of whether the deterministic preconditions defined by the *speed* modality were overly-restrictive and preventing generation of maintenance plans. We consequently evaluated *CAMP-BDI.Quality* in $n_{exo} = 3$, which relaxed precondition constraints by using pseudo-probabilistic planning. However the actual difference between *CAMP-BDI* modalities (Figure 12.7) was *not* statistically significant in World B; there were also small but statistically significant advantages for *CAMP-BDI.Speed* in World A for $n_{risk} = 0.5$ (92.18% compared to 90.83%) and $n_{risk} = 0.75$ (91.75% versus 92.98%).

	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
World A	+0.03 (0.965)	-0.43 (0.482)	+1.35 (0.036)	+1.23 (0.031)
World B	-1.45 (0.161)	-1.34 (0.303)	+0.38 (0.766)	-0.15 (0.911)

Figure 12.7: *CAMP-BDI.Quality* goal achievement subtracted from *CAMP-BDI.Speed* (p in brackets) for $n_{exo} = 3$ in World A and B. Positive values indicate *CAMP-BDI.Speed* achieved *more* goals on average than *CAMP-BDI.Quality*.

This suggests *CAMP-BDI* failure arose where *acceptable* maintenance plans could not be formed, for *either* modality; i.e. even with relaxed constraints, there were cases where the best (highest confidence / lowest inverse risk cost) maintenance plan generated for *CAMP-BDI.Quality* was still considered of unacceptable confidence. The small superiority of *CAMP-BDI.Speed* in World A $n_{risk} = 3$ may reflect the consequences of permitting (limited) low confidence activities in maintenance plans for *CAMP-BDI.Quality*, if these activities later suffered non-deterministic failure (and could not themselves be addressed by subsequent maintenance). Although significant, the small absolute difference may also *potentially* be a result of divergence between the two maintenance modalities.

Continual Replanning could be considered as 'aggressive' maintenance, in the sense of reforming plans (albeit using a deterministic domain specification without capability knowledge) on each step. The superior performance of CAMP-BDI (both modalities) against this system shows the value of capability knowledge, which allowed avoidance of non-deterministic failure from low-confidence execution contexts. This is shown by decreasing performance of *Continual Replanning* (and *Replanning*) over increasing n_{exo} – as this entailed greater perturbation, and a greater probability of agent plans containing activities impacted by such states (if not explicitly avoiding them through capability confidence knowledge).

Although we did not require any specific planning method, our results show the determinization employed by *CAMP-BDI.Speed* was effective for maintenance plan generation despite more restrictive precondition constraints; offering the same or better robustness as the *CAMP-BDI.Quality* (pseudo-probabilistic) planning modality. In general terms, they support our core hypothesis that robustness – measured through goal achievement – in realistic environments where failure risks debilitation can be improved by adopting a proactive approach to *avoid* failure rather than relying upon reactive recovery. This applied across increasing levels of perturbation and for both geographies tested – even where our approach performed worst ($n_{exo} = 3$ in World B), it held significant and increasing superiority for goal achievement with debilitation risk ($n_{risk} \geq 0.25$).

12.2 Average Activity Success Rate

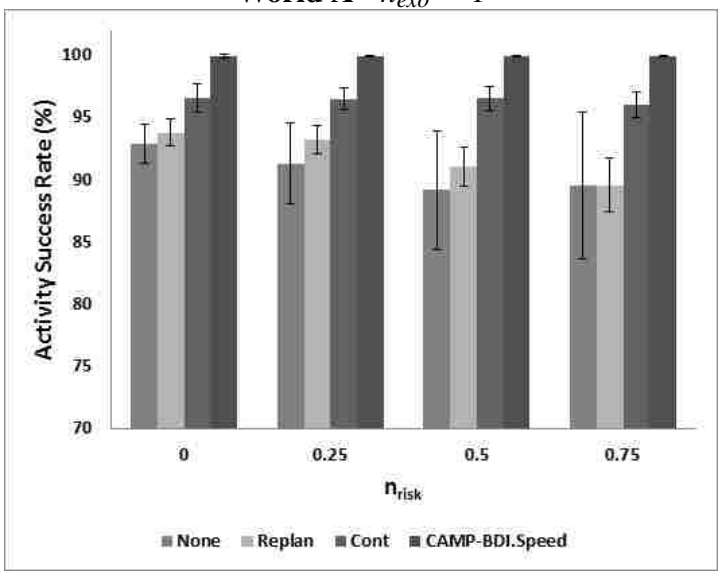
This section considers the activity success rate of *CAMP-BDI*, to verify our proactive approach successfully avoids activity failure. Due to their fundamentally different, reactive, approach towards robustness we expect greater activity failure rates for *Replanning* and *Continual Replanning* approaches than *CAMP-BDI*.

Figure 12.8 and Figure 12.9 gives the activity success rate for approaches in World A and B respectively. We provide specific differences in Appendix B.2 (Figure B.5 and Figure B.6, using the same format as Figure 12.3)[2]. Although generally high for all approaches and configurations ($\geq$ 77%), our results show *CAMP-BDI* had the greatest success rate for all n_{exo} and n_{risk}. For $n_{exo} = 3$, in both World A and World B, *CAMP-BDI.Quality* had a lower activity success rate – these differences were marginal (ranging from 0.081% to 0.157%, both in World B) but statistically significant for $n_{risk} \geq 0.5$ in World A, and all n_{risk} in World B. This is attributed to *CAMP-BDI.Quality* being able to form and accept maintenance plans containing lower confidence activities, where these later suffered non-deterministic failure.

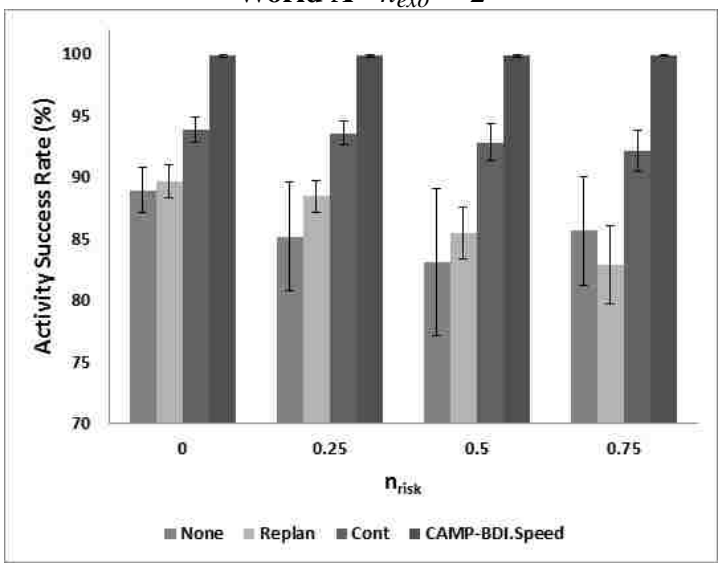
Activity success for *Replanning* decreased with both increasing n_{exo} and n_{risk} – extending to equal or worse rate of activity success than *None* at $n_{risk} = 0.75$, stemming from the increased probability and potential accumulation of post-failure debilitation. *Continual Replanning* exhibited more success in activity execution than *Replanning*, as (attempted) constant revision of agent plans helped prevent preconditions failure. The superior activity success rate of *CAMP-BDI* over *Continual Replanning* reinforces the utility of confidence estimation, combined with Maintenance Policy thresholds, as a trigger for maintenance – preventing *non-deterministic* failures where preconditions held, yet the execution context still increased failure risk.

[2]For brevity, numerical detail for performance metrics from our experiments are given in Appendix B.

World A - $n_{exo} = 1$



World A - $n_{exo} = 2$



World A - $n_{exo} = 3$

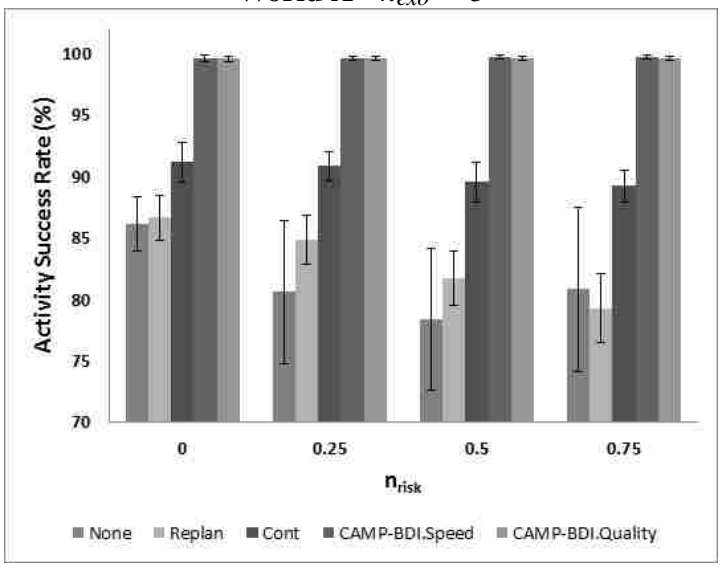
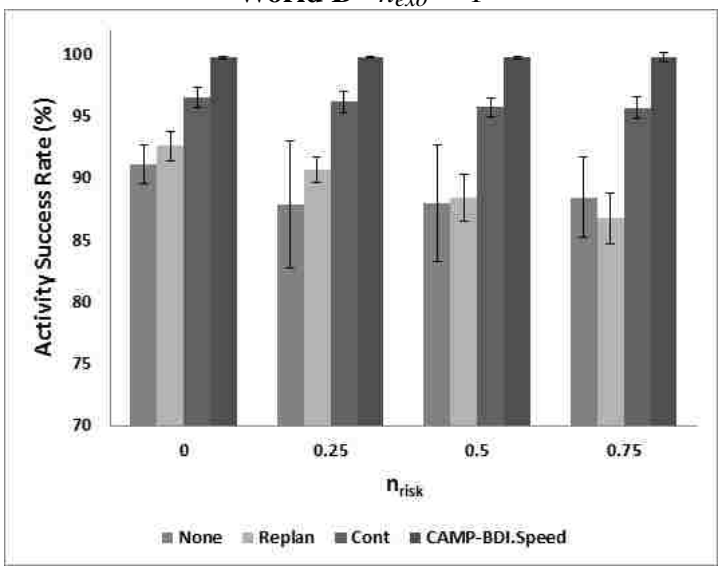
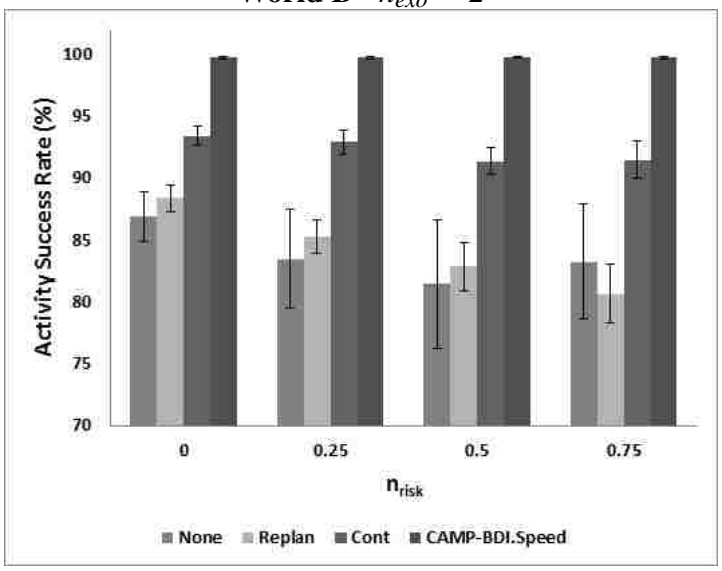


Figure 12.8: Activity Success Rate in World A

World B - $n_{exo} = 1$



World B - $n_{exo} = 2$



World B - $n_{exo} = 3$

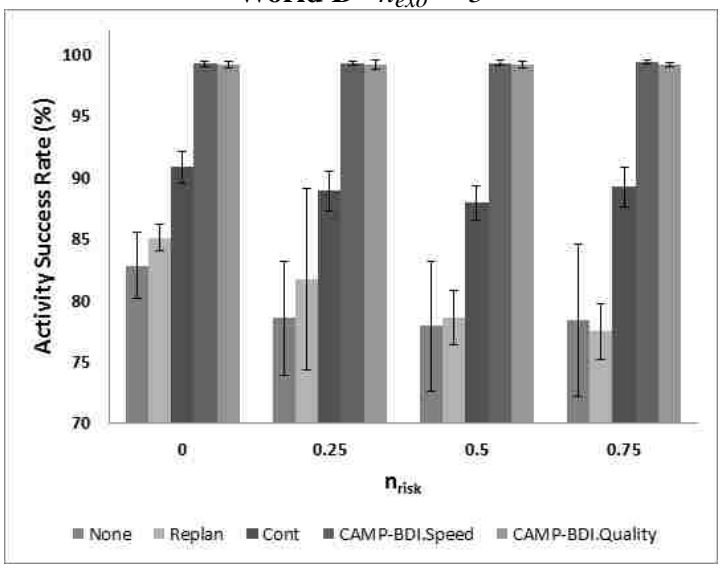


Figure 12.9: Activity Success Rate in World B

These results show *CAMP-BDI* was more effective at avoiding activity failure than *Replanning* or *Continual Replanning*, due to being able to revise plans pre-emptively and, with respect to the latter, using confidence estimation to anticipate increased failure *risk*. Our maintenance approach also retained consistency despite increasing risks of debilitation (Figure 12.10). In both World A and B for all n_{exo} *CAMP-BDI.Speed* did not experience any statistically significant difference in activity success with increasing n_{risk} – evidencing the robustness (in activity failure avoidance terms) benefits of avoiding debilitation by avoiding failure. These results also applied for *CAMP-BDI.Quality* evaluation in $n_{exo} = 3$, showing both modalities were effective at preventing activity failure.

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-0.001 (0.952)	+0.008 (0.606)	-0.005 (0.751)
$n_{exo} = 2$	+0.009 (0.686)	+0.016 (0.384)	+0.01 (0.491)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.009 (0.79)	+0.058 (0.078)	+0.035 (0.298)
CAMP-BDI.Qty	+0.024 (0.546)	+0.02 (0.594)	+0.048 (0.11)

World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+0.021 (0.242)	-0.011 (0.548)	+0.022 (0.306)
$n_{exo} = 2$	+0.007 (0.693)	+0.019 (0.351)	-0.014 (0.465)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.056 (0.108)	-0.039 (0.261)	-0.002 (0.956)
CAMP-BDI.Qty	-0.011 (0.856)	-0.008 (0.889)	-0.015 (0.692)

Figure 12.10: Differences (p in brackets) for *CAMP-BDI* activity success rates under progressive increase of n_{risk}. Negative values indicate decreased activity success rate at the higher n_{risk}.

12.3 Average Delivery Cost (Activities per Goal)

The previous results support our primary hypothesis of improved robustness through proactive maintenance behaviour, reflecting the successful avoidance of activity failure. The *average delivery cost* is intended to identify the *efficiency* impact associated with our CAMP-BDI approach, including from plans added to prevent or (in comparison against reactive approaches) recover from failure. Figure 12.11 and Figure 12.12 show average delivery cost (in activities) for World A and B respectively. Specific differences between approaches are detailed in Appendix B.3; Figure B.9 and Figure B.10 give differences for *CAMP-BDI.Speed* in World A and B, and give those Figure B.11 for *CAMP-BDI.Quality*.

We first consider the performance of *None*, where activity failure entailed immediate *goal* failure. In World A, for all n_{exo} the average activity cost initially rose with increasing n_{risk}, but levelled off such that no significant difference existed between $n_{risk} = 0.5$ and 0.75. In World B delivery costs for *None* were more variable. In $n_{exo} = 1$, only the difference between $n_{risk} = 0$ and 0.25 was significant ($p = 1.914x10^{-5}$); for $n_{risk} = 2$, only that between $n_{risk} = 0.5$ and 0.75 was significant ($p = 0.01$); in $n_{exo} = 3$, *no* differences between escalating n_{risk} were. Where cost decreased in $n_{exo} = 2$ between $n_{risk} = 0.5$ and 0.75, this can be attributed to where the increased debilitation risk led to earlier failure of activities (and consequently intentions) – or potentially that highly likely debilitation led to a shortage of non-*mortal* agents of the types required to *form* a $plan_i$.

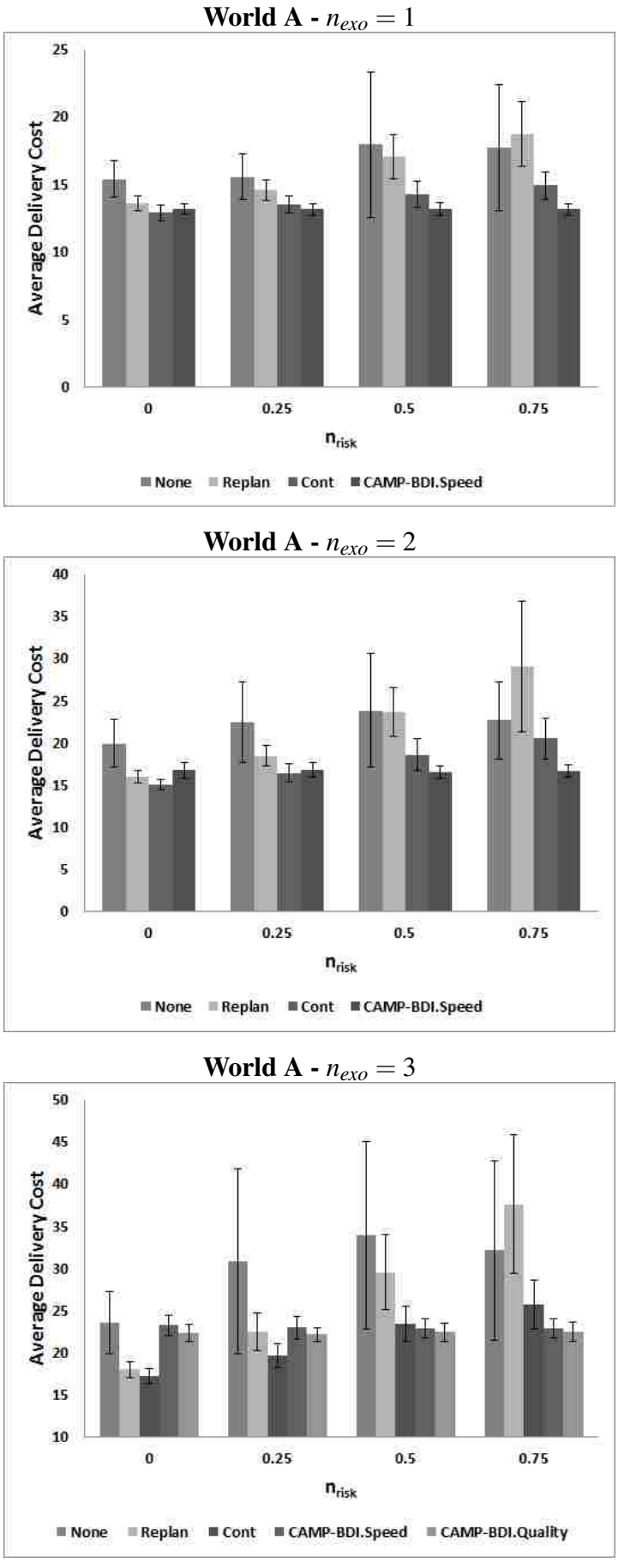


Figure 12.11: Average Activities Per Goal in World A, with standard deviation

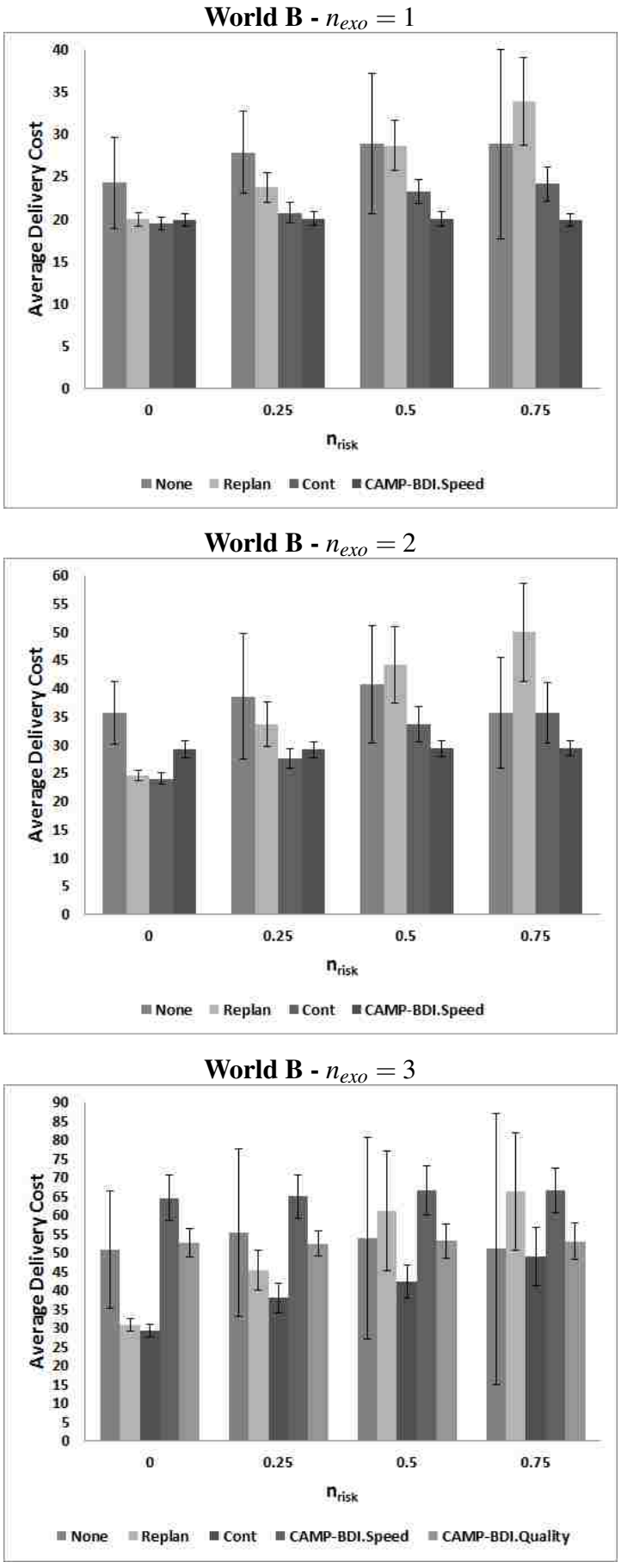


Figure 12.12: Average Activities Per Goal in World B, with standard deviation

In both World A and World B and for all n_{exo}, significant increases in activity cost occurred for both *Replanning* and *Continual Replanning* over progressively increasing n_{risk}. Conversely, neither *CAMP-BDI.Speed* nor *CAMP-BDI.Quality* shown significant (Figure 12.13) differences in average cost over increasing n_{risk}, in any geography or for any n_{exo}. The increasing advantage of *CAMP-BDI* approaches with greater n_{risk} can be attributed to *Continual Replanning* and *Replanning* approaches suffering greater levels of post-failure (cumulative) debilitation, such that recovery was eventually rendered impossible – i.e. more activities were executed for ultimately unsuccessful goals.

Indeed, *None* actually held lower average activity costs than *Replanning* at $n_{risk} = 0.75$, in both geographies and all n_{exo}, due to the extra expenditure associated with trying to recover from activity failure for ultimately impossible goals. Unlike *Replanning*, *None* would fail goals immediately with activity failure – the lack of recovery activity both reduced the cost of failed goals *and* exposure to debilitation caused by failed activities performed in attempt of recovery (but with the cost of unacceptably low goal achievement).

World A	$n_{risk} = 0 \rightarrow 0.25$	$n_{risk} = 0.25 \rightarrow 0.5$	$n_{risk} = 0.5 \rightarrow 0.75$
$n_{exo} = 1$	-0.021 (0.761)	+0.029 (0.715)	-0.0277 (0.737)
$n_{exo} = 2$	+0.092 (0.488)	-0.272 (0.078)	+0.138 (0.285)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.203 (0.34)	-0.112 (0.655)	-0.047 (0.791)
CAMP-BDI.Qty	-0.13 (0.451)	-0.282 (0.501)	+0.017 (0.925)

World B	$n_{risk} = 0 \rightarrow 0.25$	$n_{risk} = 0.25 \rightarrow 0.5$	$n_{risk} = 0.5 \rightarrow 0.75$
$n_{exo} = 1$	+0.197 (0.123)	-0.027 (0.854)	-0.134 (0.265)
$n_{exo} = 2$	-0.021 (0.938)	+0.202 (0.053)	+0.101 (0.699)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.443 (0.694)	-1.532 (0.227)	-0.096 (0.934)
CAMP-BDI.Qty	+0.305 (0.663)	-0.705 (0.317)	+0.1001 (0.91)

Figure 12.13: Differences (p in brackets) in average activities per delivery for *CAMP-BDI* over increasing n_{risk}; positive values where average cost was greater at higher n_{risk}, although this was not significant if $p > 0.05$.

In World A, *CAMP-BDI.Speed* had consistently lower average activity costs for all n_{exo} where $n_{risk} \geq 0.5$; at $n_{risk} \leq 0.25$ *CAMP-BDI* either had lesser or no significant advantage over others. For $n_{risk} = 0$ in all World A n_{exo}, and for $n_{risk} = 0.25$ at $n_{exo} = 2$ (by a small but significant degree) and $n_{exo} = 3$, *Continual Replanning* had lower activity costs than other approaches – likely as an outcome of optimizations offered by continually (re)forming an optimal plan after every activity. *Replanning* also had small but significant advantages over *CAMP-BDI* approaches where $n_{risk} = 0$; likely due to the non-impact of debilitation allowing near 100% goal achievement (i.e. avoiding activities being 'wasted' on failed goals). Finally, we observed a small but significant advantage for *CAMP-BDI.Quality* over *CAMP-BDI.Speed* in all $n_{risk} \leq 0.5$ when both were evaluated in $n_{exo} = 3$, which we attribute to the former being able to (potentially) form shorter route plans by permitting travel through lower-confidence roads (*if* zero-risk movement was impossible, or incurred high enough cumulative cost to override the cost metric values that normally biased against low confidence activities).

World B shown considerable variance in results between $n_{exo} \leq 2$ and $n_{exo} = 3$ – reflecting decreased *CAMP-BDI* goal achievement for the latter n_{exo} value. For World B $n_{exo} = 1$, *CAMP-BDI.Speed* had significantly less average activity cost than *Continual Replanning* and *Replanning*, with that advantage growing with n_{risk} – except for $n_{risk} = 0$, where *Continual Replanning* executed (by a small but significant margin) less activities per goal. For $n_{exo} = 2$, *CAMP-BDI.Speed* executed less activities per goal on average than *Replanning* for $n_{risk} \geq 0.25$, and than *Continual Replanning* for $n_{risk} \geq 0.5$; this advantage again grew with n_{risk}. In World B $n_{exo} = 3$, *CAMP-BDI.Speed* consistently had higher per-goal activity costs than reactive robustness approaches – reflecting significantly decreased goal achievement for this configuration, and the greater likelihood that activities within inserted maintenance plans would *not* lead to eventual goal success. These differences lessened, reflecting decreasing goal achievement of reactive approaches with increasing n_{risk} – extending to no significant difference between *CAMP-BDI.Speed* and *Replanning* at $n_{risk} = 0.75$.

In World A, *CAMP-BDI.Quality* generally mirrored results for *CAMP-BDI.Speed*, with a small (≤ 0.92) yet statistically significant increase in cost for $n_{exo} \leq 0.5$. Within World B, however, *CAMP-BDI.Quality* executed *less* activities on average than *CAMP-BDI.Speed*, and also less than *Replanning* for $n_{risk} \geq 0.5$ (7.939 and 13.144 activities less, on average). Despite goal achievement being *worse* than *CAMP-BDI.Speed* (Sec-

tion 12.1) – i.e. indicating more activities would be performed in ultimately failed plans – this indicates *CAMP-BDI.Quality* agents were able to form shorter length plans by allowing of lower confidence activities. *CAMP-BDI.Speed* agents, conversely, risk additional backtracking to completely avoid low-confidence activities within maintenance plans (particularly for route plans). This difference was likely most prominent in World B due to it's larger geography and reduced interconnectivity limiting the range of possible plans to achieve movement goals.

We can consider the significantly lower goal achievement of both *Replanning* and *Continual Planning* approaches for $n_{risk} \geq 0.25$ in World B $n_{exo} = 3$ as a mitigating factor where (either or both) *CAMP-BDI* approaches held higher costs. Being reactive approaches (with regard to failure handling) meant *Replanning* and *Continual Replanning* 'allow' failure to occur, risking associated debilitation. The high rate of goal failure indicates both reactive approaches increasingly found themselves in irrecoverable failure scenarios; this makes it likely *more rapid* failure would occur, and also that (for $n_{risk}>0$) debilitative effects would accumulate and hasten failure of *subsequent* goal. In contrast, *CAMP-BDI* agents formed and executed maintenance plans until failure *avoidance* was impossible – in this most difficult environment, this can be characterised as the agents acting constantly to avoid failure (or, in a critical interpretation, delay it). Finally, the greater goal achievement of *CAMP-BDI* agents (of either approach) may indicate *CAMP-BDI* expended greater activity cost when achieving those goals than reactive approaches expended before goal failure.

Average activity cost derives from two factors; the efficiency of goal achievement – i.e. the cost of plans which prevent or recover from failure – and the rapidity of failure – i.e. the number of activities an agent expended, including for proactive or reactive failure mitigation, before a goal became impossible. In World A, once debilitation was a risk, *CAMP-BDI.Speed* executed the least activities per goal on average, followed by *Continual Replanning*, *Replanning* and finally *None* – correlating with goal achievement (Section 12.1) and suggesting *CAMP-BDI*'s lower costs were attributable to avoiding 'futile' activities associated with the pursuit of failed goals.

CAMP-BDI.Quality shown broadly similar results to *CAMP-BDI.Speed* in World A; executing slightly (but significant) more activities per goal on average in $n_{risk} \leq 0.5$, but providing effective parity (no significant difference) at $n_{risk} = 0.75$. *CAMP-BDI*

maintained consistent performance across increasing debilitation risk, with greater efficiency advantages over reactive approaches at increasing perturbation levels. The superior performance of *Continual Replanning* over *Replanning* for all configurations in both geographies is attributable to the former constantly reforming plans – allowing both avoidance of preconditions failure, and for *Continual Replanning* agents to always hold the optimal (shortest length) plan possible given current beliefs.

Results for $n_{exo} = 1,2$ in World B reflected those of World A, with *CAMP-BDI.Speed* again showing lower average activity cost and increasingly superior performance with increasing n_{risk} for $n_{exo} = 1$ and 2. For $n_{exo} = 3$, however, *CAMP-BDI.Speed* was consistently *worse*, and over all n_{risk} (excepting *Replanning* for $n_{risk} = 0.75$). Despite this, *CAMP-BDI.Speed* (and *CAMP-BDI.Quality*) maintained consistent costs regardless of debilitation risk for all n_{exo}, whilst reactive approaches faced increasing average activity costs as n_{risk} increased.

The increased costs for *CAMP-BDI* are attributed to an evident marked increase in difficulty for World B $n_{exo} = 3$. This level of perturbation requires more frequent maintenance (with resultant maintenance plans adding to absolute activity costs), with the reduced interconnectivity of this geography reducing possible route plans for road travel (increasing the risk of failed maintenance planning and, ergo, goal failure). The general geographic properties of World B also increased the length of individual route, increasing the possible exposure to exogenous change during plan execution. *CAMP-BDI.Quality* – although still incurring greater activity cost than *Continual Replanning* for all n_{risk} , and greater than *Replanning* for $n_{risk} \leq 0.25$, in World B – did show significantly lower average activity costs than *CAMP-BDI.Speed*. This suggests pseudo-probabilistic planning, which allowed reduced precondition constraints through adding confidence-reflecting costs, may be advantageous if the high probability, or wide distribution of, confidence lowering states in the environment hinders stricter deterministic planning (i.e. as in *CAMP-BDI.Speed*).

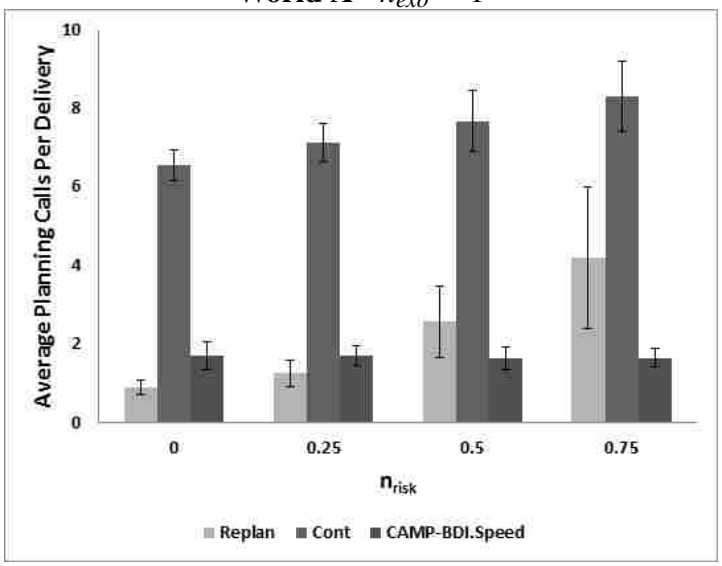
12.4 Planning Operations Per Goal

CAMP-BDI maintenance behaviour employs proactive, pre-emptive planning, using capability knowledge to estimate the execution context of future activities and – by extension – initial state specifications when performing maintenance planning. Unlike

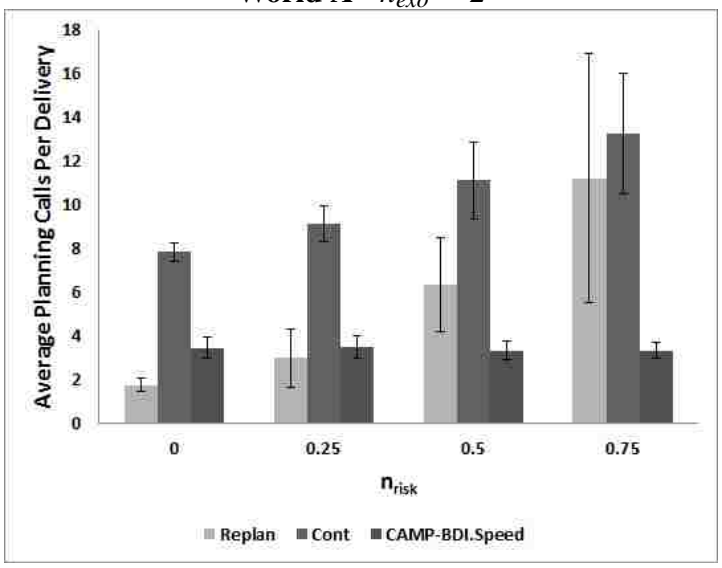
reactive *Replanning*, performed following *confirmed* failure, proactive behaviour risks incurring extra costs from uncertainty – i.e. from false positive anticipation of failure, or if maintaining activities which are themselves removed (before execution) by future maintenance. *CAMP-BDI* also risks additional costs due to performing discrete planning calls for each step of scope escalation during local maintenance, and if dependants adopt maintenance responsibility. Our algorithm uses these multiple planning operations to minimize changes to $plan_i$s, and sacrifices the possibility of planner-specific optimizations to avoid mandating a specific planner implementation.

Section 12.1 showed that *CAMP-BDI*'s proactive approach improved robustness over reactive approaches when failure risked debilitation; here, we evaluate planning cost in terms of average planner invocations per achieved goal. Figure 12.14 and Figure 12.15 shows the average planning calls per achieved goal in World A and B respectively. Detailed differences are given in Appendix B.4 – Figure B.14 and Figure B.15 give differences for *CAMP-BDI.Speed* against other approaches, and Figure B.16 defines differences between *CAMP-BDI.Quality* and replanning approaches for $n_{exo} = 3$ in both geographies.

World A - $n_{exo} = 1$



World A - $n_{exo} = 2$



World A - $n_{exo} = 3$

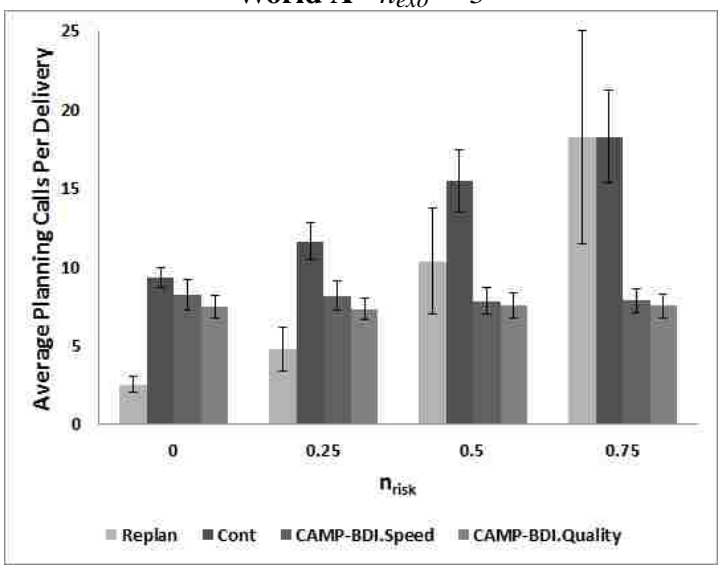
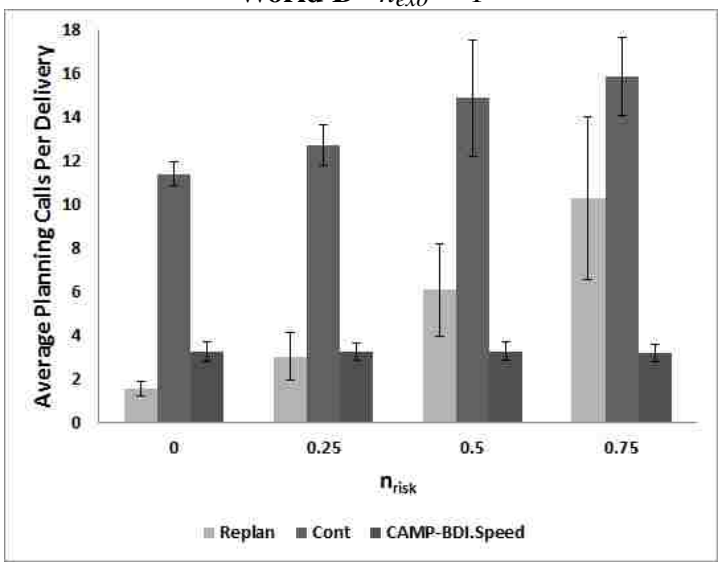
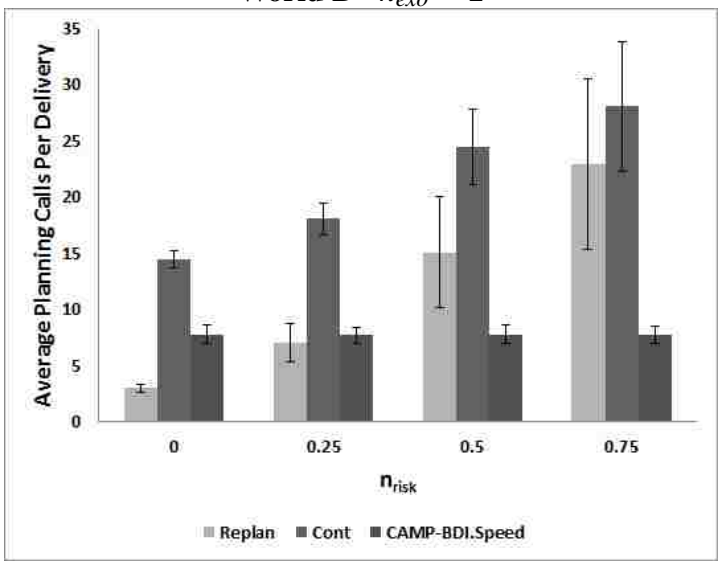


Figure 12.14: Planning Cost in World A

World B - $n_{exo} = 1$



World B - $n_{exo} = 2$



World B - $n_{exo} = 3$

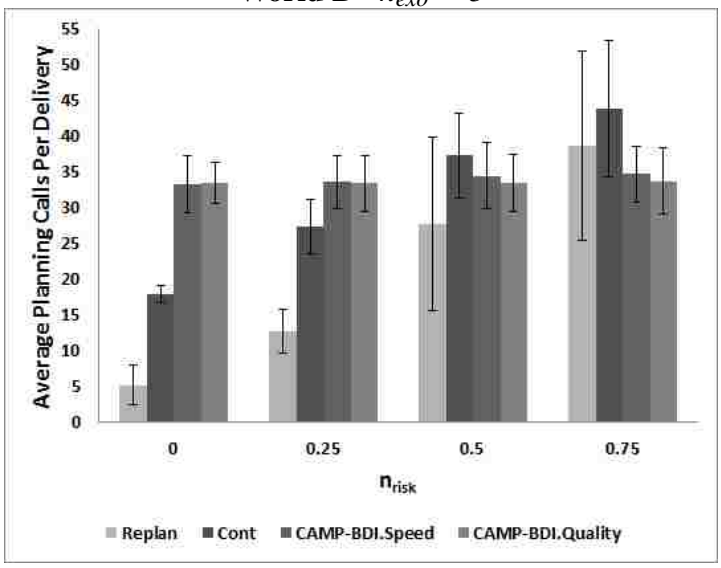


Figure 12.15: Planning Cost in World B

For all experiments (i.e. all geographies, n_{exo} and n_{risk}) *Continual Replanning* required significantly more planning operations per goal than any other approach, as *every* activity – successful or failed – was followed by (re)planning. Increases in average planner call cost also reflects *decreasing* rate of goal achievement with increasing n_{exo} and n_{risk} – meaning more planning calls were ultimately *futile*, particularly where stymied by increasing probabilities and accumulation of post-failure debilitation. These results show *CAMP-BDI* offered an improvement over continual replanning in this context, by making planner invocation non-arbitrary and performed upon the basis of capability-driven threat anticipation.

Both geographies show similar results; *Replanning* initially had lower planner execution costs (less planning operations per activity) – reflecting that replanning was only performed when necessary to respond to *actual*, rather than (potentially incorrectly) anticipated, failure. As n_{risk} increased *Replanning* executed more planning operations per goal than *CAMP-BDI.Speed* in World A for all n_{exo}, in World B for $n_{exo} \leq 2$ where $n_{risk} \geq 0.5$, and in World B $n_{exo} = 3$ at $n_{risk} = 0.75$. Whilst average planning operations rose with increasing n_{risk} for *Replanning*, the planning rate remained consistent (i.e. with no significant differences) regardless of increasing debilitation risk for both *CAMP-BDI.Speed* and *CAMP-BDI.Quality* in all experimental configurations (Figure 12.16). Finally, *CAMP-BDI.Speed* performed slightly – but statistically significantly – more planning operations per goal in World A $n_{exo} = 3$ (for all except $n_{risk} = 0.5$), but not in World B (where no significant difference was recorded). When a statistically significant difference did exist, *CAMP-BDI.Quality* performed a maximum 0.845 more planning operations per goal.

CAMP-BDI typically executed more absolute planning calls than *Replanning* (but less than *Continual Replanning*) in World A for all n_{exo} (Figure 12.17), and in World B for $n_{exo} = 2$ and 3 (Figure 12.18). In World B, *CAMP-BDI.Speed*, followed by *CAMP-BDI.Quality* actually executed more planning calls than *Continual Replanning* for $n_{exo} = 3$. As this occurred for $n_{risk} = 0$ – i.e. where *Continual Replanning* did not see post-failure debilitation stymie recovery planning and reduce calls through earlier goal failure – it likely signifies increased escalation of maintenance responsibility to (adoption by) dependants, where obligants were unable find plans in a highly perturbed environment. It is probable that *CAMP-BDI* agents performed multiple maintenance planning operations as the *density* of low confidence states – especially for *slippery*

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-0.002 (0.957)	-0.056 (0.291)	+0.002 (0.963)
$n_{exo} = 2$	+0.054 (0.55)	-0.164 (0.092)	+0.005 (0.94)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.029 (0.848)	-0.328 (0.068)	+0.031 (0.815)
CAMP-BDI.Qty	-0.138 (0.301)	-0.229 (0.09)	+0.011 (0.942)

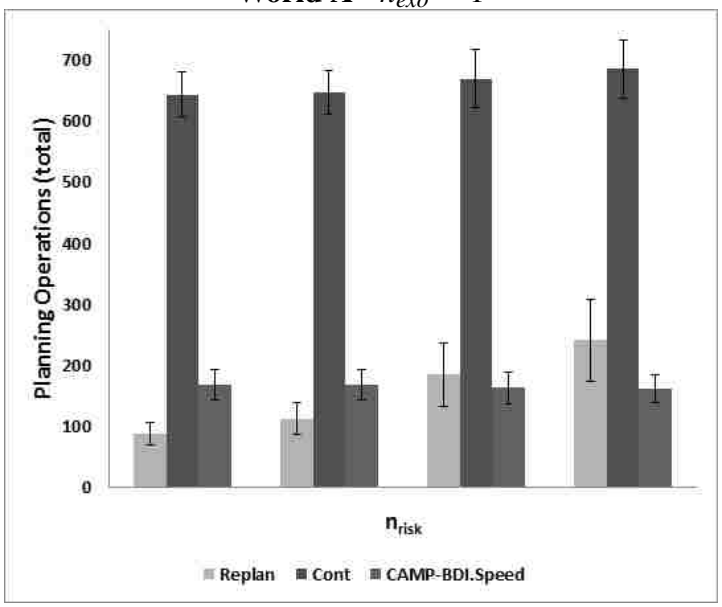
World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+0.01 (0.883)	+0.009 (0.897)	-0.074 (0.254)
$n_{exo} = 2$	-0.091 (0.521)	+0.105 (0.443)	-0.046 (0.751)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.299 (0.687)	-0.889 (0.296)	-0.226 (0.768)
CAMP-BDI.Qty	+0.018 (0.977)	-0.155 (0.836)	-0.172 (0.831)

Figure 12.16: Differences (p in brackets) for *CAMP-BDI* average planning calls per goal in World A and B over increasing n_{risk}; negative values indicate less planning calls per goal were performed at the greater n_{risk}.

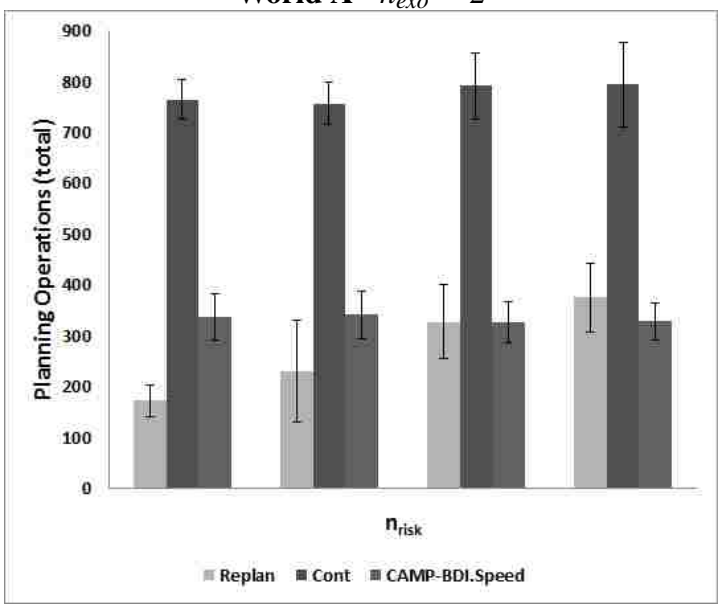
roads – made it difficult to identify alternate obligants in a position to both replace existing, low confidence, obligants *and* avoid confidence reducing states. The similar lack of success for *CAMP-BDI.Quality* could indicate that, even with reduced constraints, in many cases it was impossible to find acceptable maintenance plans.

Consistency in *CAMP-BDI* planning call costs, and increasing average calls for *Continual Replanning* and *Replanning* with n_{risk}, can be attributed to differences in goal achievement rates. This suggests *CAMP-BDI*'s mitigation against goal and activity failure also mitigated additional planning costs by avoiding *futile* planning (where the goal eventually failed due to debilitation). Where *CAMP-BDI* presents excess (absolute or relative) planning operations, these can be justified through superior goal achievement (robustness where $n_{risk} \geq\ 0.25$) or, potentially, as a result of *earlier* total plan failure restricting the opportunity(s) for reactive replanning to add activities.

World A - $n_{exo} = 1$



World A - $n_{exo} = 2$



World A - $n_{exo} = 3$

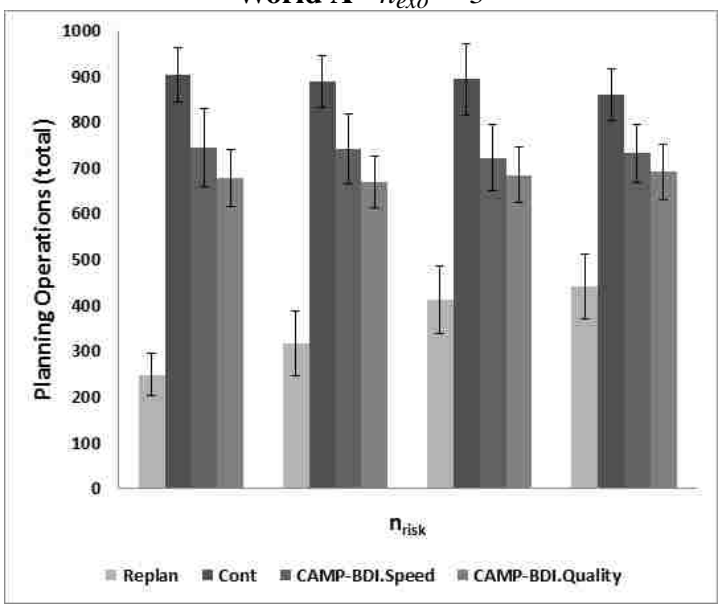
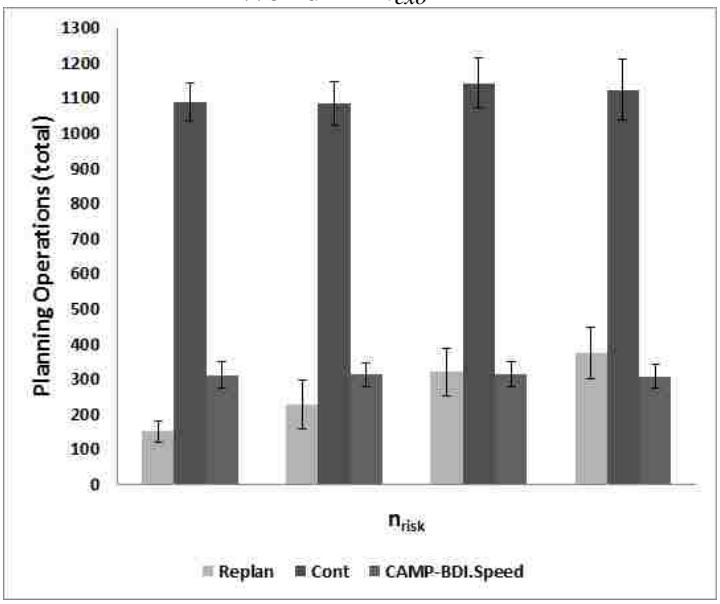
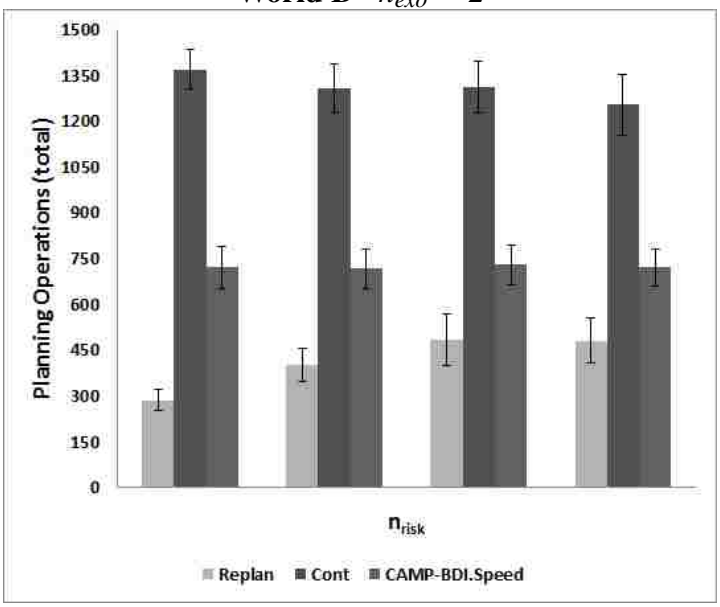


Figure 12.17: Total (absolute) Planner Calls in World A

World B - $n_{exo} = 1$



World B - $n_{exo} = 2$



World B - $n_{exo} = 3$

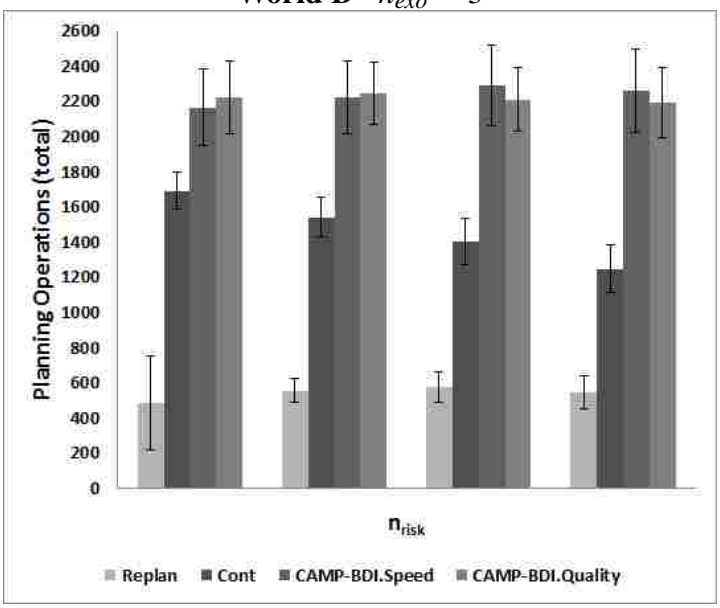


Figure 12.18: Total (absolute) Planner Calls in World B

12.5 Planning Time Costs

Although *CAMP-BDI* may entail more *individual* planning operations than reactive *Replanning*, the previous section showed these were effectively mitigated by robustness improvements under increasing risks of post-failure debilitation. This section examines whether the *context* for invoking planning (proactive, reactive or continual) impacts computational cost. We employ the same third party planning implementation (*LPG-td*) for runtime planning in all cases, without any approach specific optimizations[3]. Although planning time metrics are considered as *approximate* (i.e. as exact values are specific to *LPG-td*), we argue they remain useful, and assume planning time can be applied as a proxy indicator for the relative complexity of problems presented to the planner.

Figure 12.19 (World A) and Figure 12.20 (World B) gives the average time (ns) per planning operation for all n_{exo} and n_{risk}. We do not measure per *goal* achieved as we wish to evaluate the difficulty of individual planning operations, rather than time expenditure per goal. Differences between *CAMP-BDI.Speed* and other approaches are detailed in Appendx B.5; in Figure B.19 and Figure B.20 for World A and B respectively.

CAMP-BDI.Quality was not directly compared against other approaches due to employing the *LPG-td.quality* modality, which defined planner execution as lasting either for a fixed time bound, or until either a set number of (progressively improved, i.e. metric minimized) plans were generated or the state search space was exhausted. Small but significant differences in planning execution time for *CAMP-BDI.Quality* over increasing n_{risk} in Figure 12.21 suggest a certain margin of error ($\pm \leq$ 1.5%) may exist with regard to the fixed time bound; we suggest likely attributable to an increased rate of failure (early termination), although it is also possible (but, we suggest, less likely) factors such as PDDL parsing time had an impact. The (comparatively) high time cost of planning with even a *pseudo*-probabilistic method also supports our assumption that *true* probabilistic planning (being of greater computational complexity than the determinized, cost based method used in *CAMP-BDI.Quality*) would be impractical in realistic domains.

[3]Variance in planning was solely from differences in how planning problems were generated and responses handled – the planner was not aware of the robustness approach invoking it.

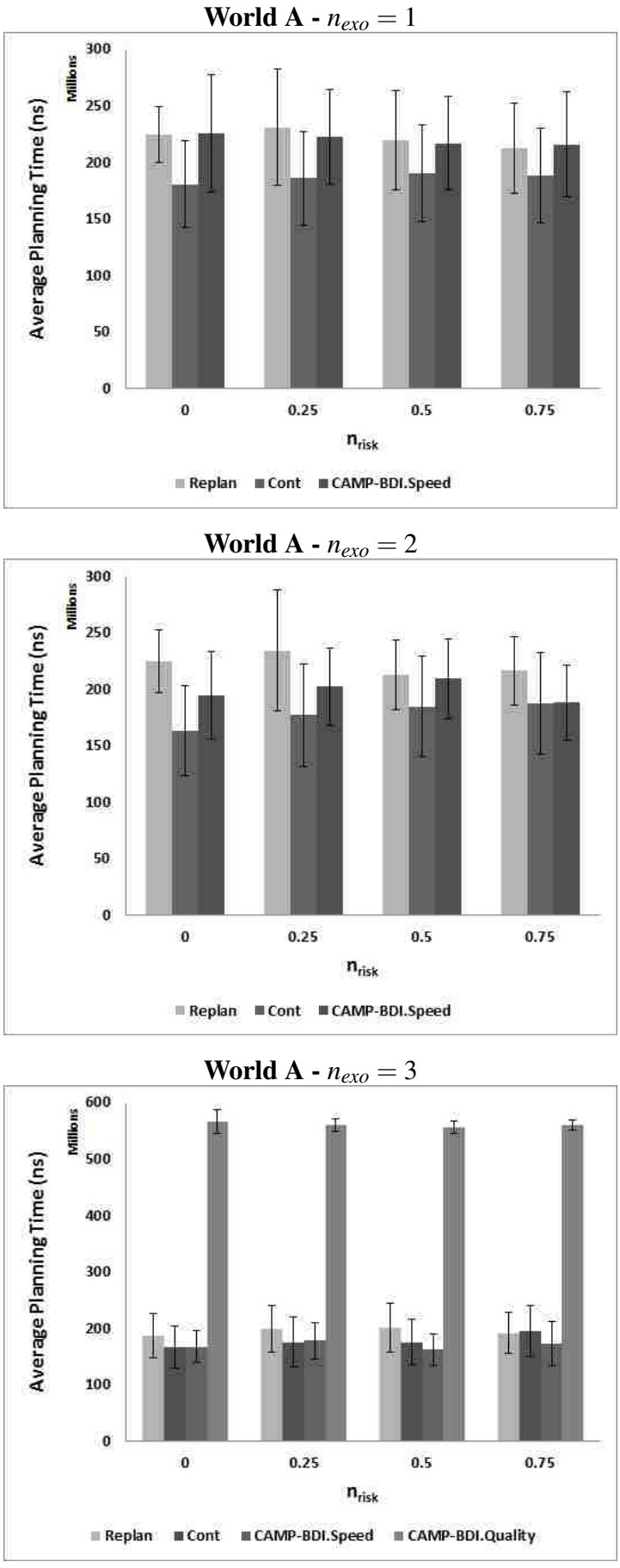


Figure 12.19: Average time for individual planning operations in World A

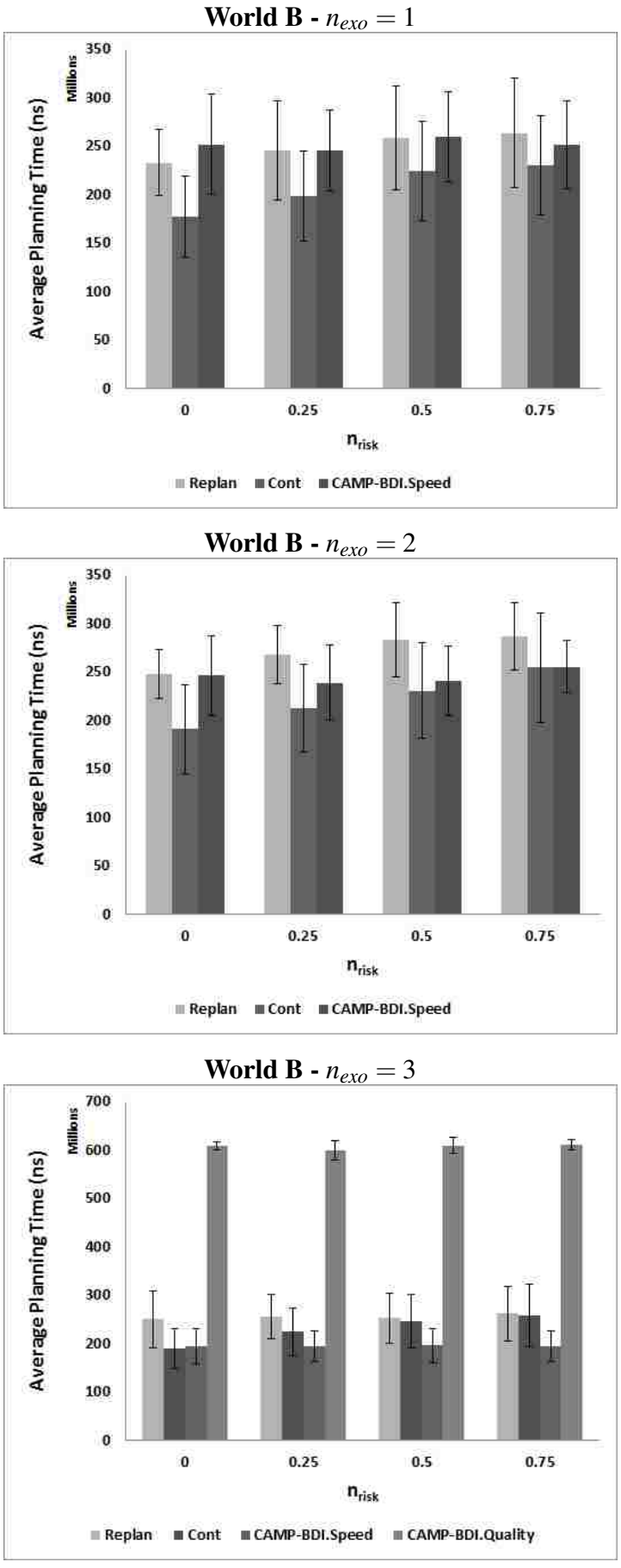


Figure 12.20: Average time for individual planning operations in World B

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-1.326% (0.497)	-2.426% (0.125)	-0.445% (0.791)
$n_{exo} = 2$	$+3.743\%$ (0.097)	$+3.512\%$ (0.069)	-10.07% ($8.15x10^{-10}$)
$n_{exo} = 3$			
CAMP-BDI.Spd	$+6.458\%$ (0.003)	-8.794% (0.0001)	$+6.165\%$ (0.036)
CAMP-BDI.Qty	-1.169% (0.025)	$+0.643\%$ (0.079)	$+0.653\%$ (0.032)

World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-2.64% (0.257)	$+5.814\%$ (0.0003)	-3.241% (0.036)
$n_{exo} = 2$	-3.135% (0.148)	$+0.924\%$ (0.732)	$+6.062\%$ (0.004)
$n_{exo} = 3$			
CAMP-BDI.Spd	$+0.194\%$ (0.92)	-0.591% (0.664)	$+0.701\%$ (0.651)
CAMP-BDI.Qty	$+1.623\%$ (0.002)	-1.703% (0.678)	-0.176% (0.005)

Figure 12.21: Percentage differences (p in brackets) for *CAMP-BDI* average planning time per goal in World A and B over increasing n_{risk} for all n_{exo}. Positive values show increased average planning time at the greater n_{risk}.

In both World A and World B for $n_{exo} = 1$ and 2, *Continual Replanning* had the lowest average planning costs, taking up to 30% less than *CAMP-BDI.Speed* – although these differences became less significant as n_{risk} increased. For $n_{exo} = 3$ any advantages in planning time held by *Continual Replanning* were not statistically significant, and gradually decreased; at $n_{risk} \geq\ 0.5$ in World A and $n_{risk} \geq 0.75$ *CAMP-BDI.Speed* took significantly *less* time on average per operation. *Replanning* generally required the same or greater time per planning operation in all environments as *CAMP-BDI.Speed*; World B $n_{exo} = 1$ and $n_{risk} = 0$ was the sole case where *Replanning* spent, on average, a statistically significant amount of time less per planning call.

The lower planning time for *Continual Replanning* may stem from being invoked continuously – i.e. without threats from exogenous change, and where the current $plan_i$ remained optimal. As Section 12.4 indicates, *Continual Planning* invoked many more planning operations per goal than approaches responding to failure – meaning many such operations would occur where there was no threat to the current plan, and the agent's previous activity would have placed it closer to goal achievement (rather than facing a threat requiring maintenance activity, or recovering from failure with any

associated hindering causal or consequential states). This would ease the difficulty of the planning problem, particularly when invoked in close 'proximity' to achieving the goal (i.e. where forming a plan required selection of only only a small number – or one – of activities).

Where a statistically significant difference existed, planning operations in *Replanning* lasted longer than within *CAMP-BDI.Speed*. *CAMP-BDI.Speed*'s relative performance was generally better (i.e. advantage increasing, or disadvantage decrease) with higher n_{risk}, although not in all cases. In these exception cases where *CAMP-BDI.Speed* relative performance was worse in higher n_{risk} (e.g. $n_{risk} = 0.75$ in World B $n_{exo} = 1$) this may be due to a more *rapid* failure in the reactive systems – i.e. where debilitation meant the planning goal was identified as unsolvable earlier by the planner.

For $n_{exo} = 3$, the difference between *Continual Replanning* and *CAMP-BDI.Speed* was significantly pronounced in $n_{exo} = 3$ – contradicting the shorter average planning time of *Continual Replanning* in $n_{exo} \leq 2$. This may be a product of an increased number of low confidence states (i.e. slippery roads impacting route plans). *CAMP-BDI.Speed* placed additional, more constraining preconditions to prohibit execution of activities with low confidence – our goal success results suggest that in many cases (more than other n_{exo}) maintenance failed to find a plan to prevent failure. This may indicate there were more cases of early planning failure than for other exogenous change configurations, where *CAMP-BDI.Speed* could still form maintenance plans under these confidence-guaranteeing preconditions.

Our results suggest that *CAMP-BDI.Speed* may offer reduced individual planning costs over *Replanning*, particularly as environmental difficulty increases, and particularly considering the reduced individual planning operations per goal observed in Section 12.4. While our algorithms are planner agnostic and these results are partially specific to *LPG-td*, they provide evidence that *CAMP-BDI.Speed* does not present disproportionately difficult planning problems – and computational cost – compared to *Replanning*, supporting the relevance of *average planning operations* as a comparative metric. Although *Continual Replanning* took less time cost for individual planning operations, this needs to be considered against lower goal achievement and higher planning *operations* cost than *CAMP-BDI.Speed*.

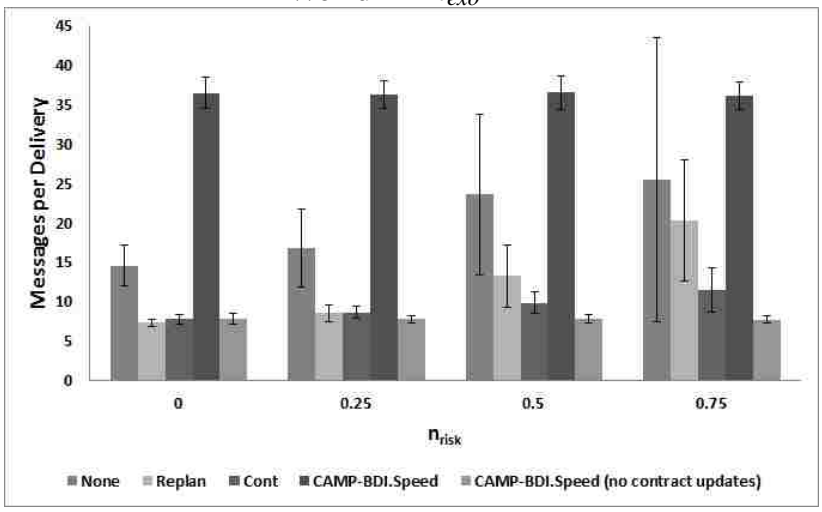
12.6 Messaging Costs

CAMP-BDI requires communication between agents to share contract information and drive adoption of responsibility by dependants during distributed maintenance. This necessitates additional communication costs over reactive approaches, which do not require information to identify threats to activities in advance – only that needed to perform planning *after* failure. We recorded the quantity of messages sent concerned with establishing, cancelling or (for *CAMP-BDI*) updating contracts to determine an average per delivery cost. This indicated the volatility of dependency relationships – i.e. how often dependencies changed due to plan failure, or from mitigatory proactive or reactive changes. For *CAMP-BDI* (both modalities), we also gathered metrics excluding the *updatedContract* message type in order to also the understand messaging costs *solely* related to plan (and delegated activity) changes made by maintenance.

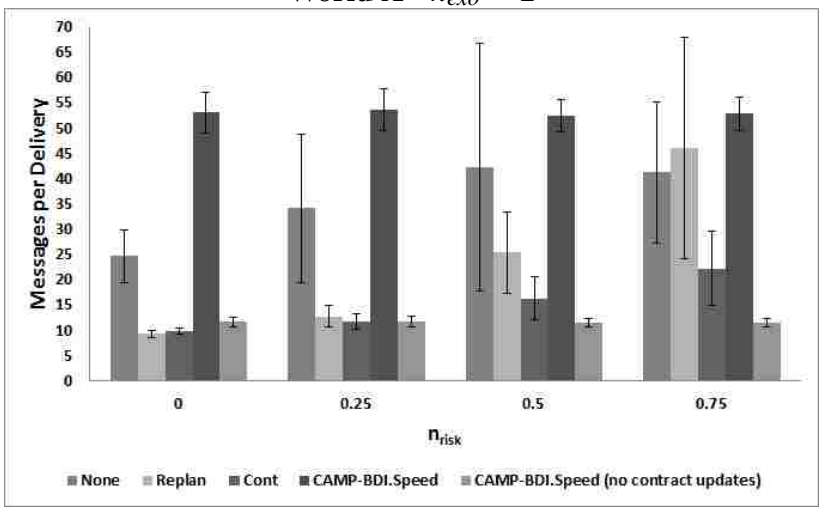
Figure 12.22 and Figure 12.23 respectively show the average messages per goal in World A and B. The differences in message costs (including *updatedContract*) between CAMP-BDI and other approaches are given in Appendix B.6.7.4, by Figure B.27 and Figure B.28 for *CAMP-BDI.Speed* and Figure B.29 for *CAMP-BDI.Quality*; these show both modalities of *CAMP-BDI* incurred significantly higher communication cost per goal than in both World A and B. Figure B.30 also details the differences in message cost for *CAMP-BDI.Speed* and *CAMP-BDI.Quality* with increasing n_{risk} – showing that in all experimental configurations, increasing debilitative risk did not significantly impact the average messaging cost per goal.

The *absolute* message counts (i.e. not divided per goal) sent, as summed across the entire experimental run, are given in Figure 12.24 and Figure 12.25 for World A and B. Appendix B.6.5 details changes in absolute message count with increasing n_{risk} levels; Figure B.25 for World A, and Figure B.26 for World B.

World A - $n_{exo} = 1$



World A - $n_{exo} = 2$



World A - $n_{exo} = 3$

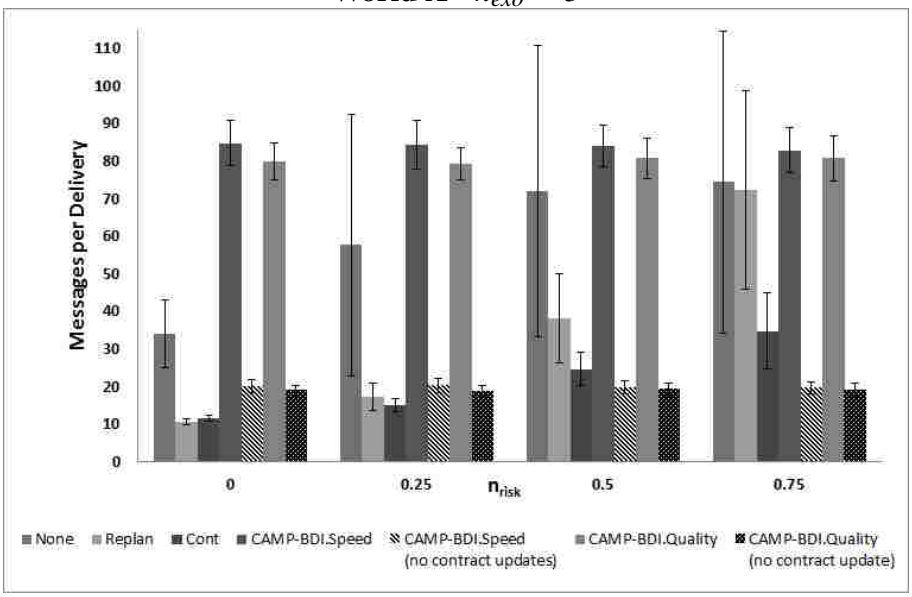
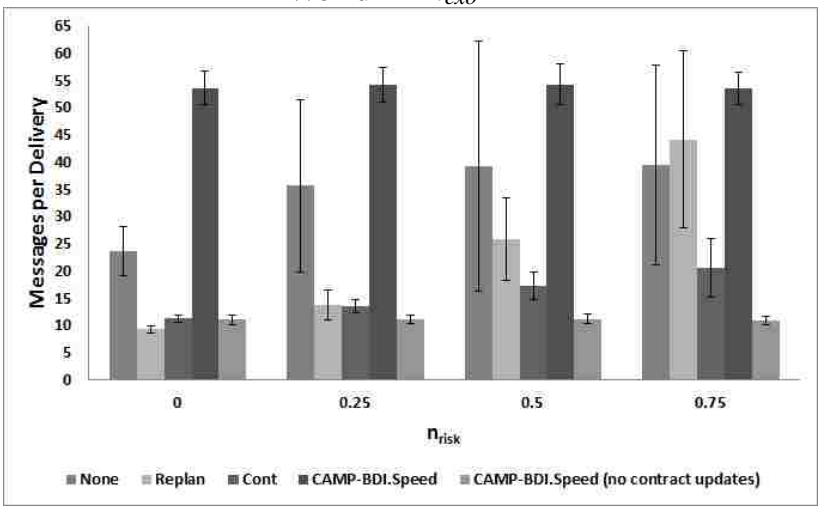
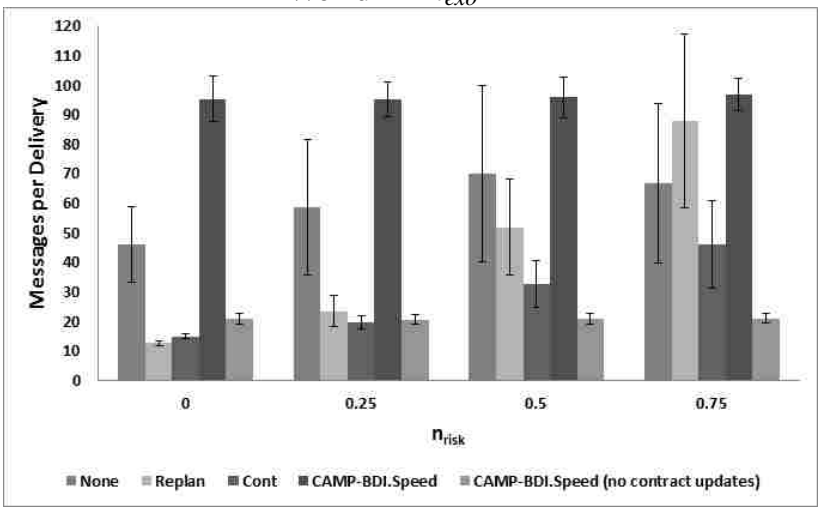


Figure 12.22: Average messages per delivery in World A

World B - $n_{exo} = 1$



World B - $n_{exo} = 2$



World B - $n_{exo} = 3$

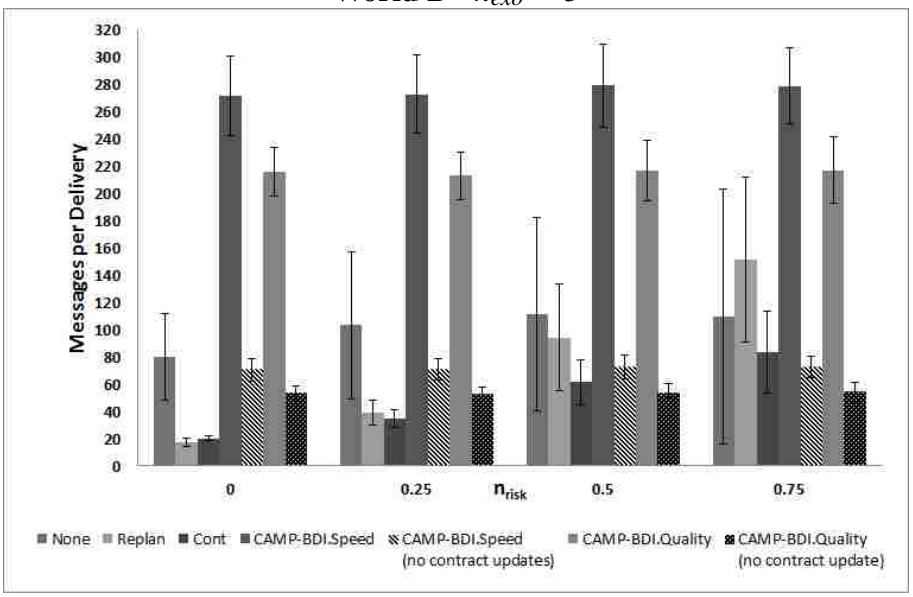


Figure 12.23: Average messages per delivery in World B

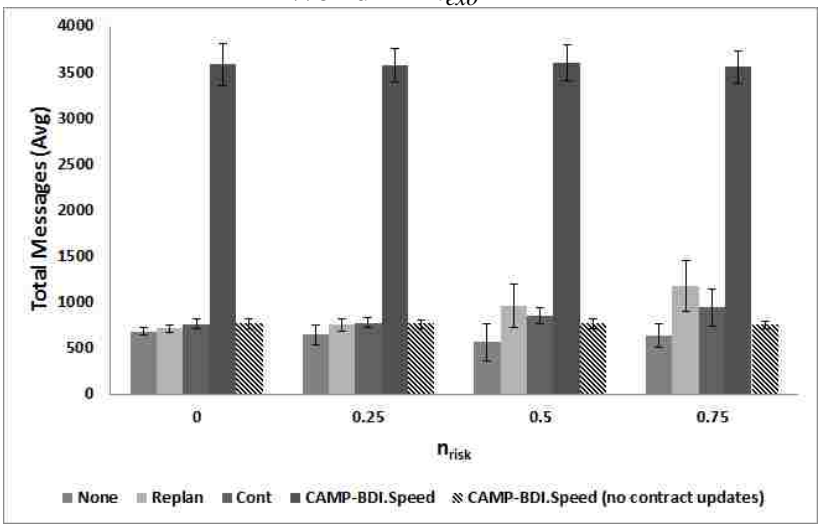
None shows increasing *per-goal* cost with n_{risk}, albeit with variation in the *absolute* number of messages; significant decreases were seen for (averaged) *total* messages with increasing n_{risk} between $n_{risk} = 0$ and 0.5, but a conversely significant *increase* was recorded between $n_{risk} = 0.5$ and 0.75. No significant differences in average total messages were associated with n_{risk} increases in $n_{exo} = 2$ or 3 for World A. Similarly variable results were seen for total message count in World B. Inconsistency in absolute message count over differing n_{risk} values reflects previous observations that perturbation level was the dominant influence upon the performance of *None* – messaging requirements would be determined by the initial plan formed, rather than any proactive or reactive plan modification or replanning behaviour.

Gradual increase of *None*'s messaging costs *per goal* can be attributed to decreasing goal achievement, as the total message count did not show any corresponding upwards trend. *None* consequently did not present a worst case baseline for messaging costs; other approaches required more messaging per goal due to reforming or modifying their $plan_i$s to mitigate against failure, which could lead to adding, modifying (for *CAMP-BDI*) or cancelling dependency contracts. *CAMP-BDI.Speed* sent significantly more messages per goal than any other approach; *CAMP-BDI.Quality* sent significantly less messages per goal than *CAMP-BDI.Speed*, but more than *Continual Replanning* and *Replanning* (Figure B.29).

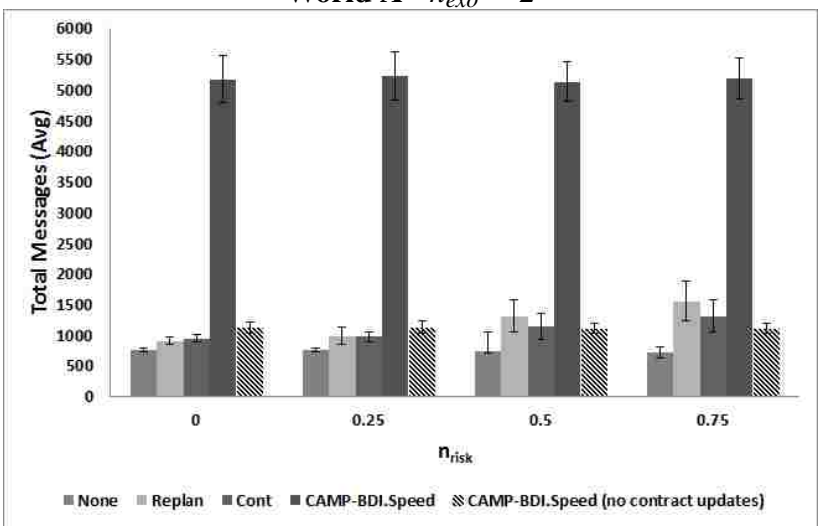
Of the alternate mitigation approaches, *Continual Replanning* generally had lower messaging costs than *Replanning*. This is somewhat surprising, given the former performed constant plan revision, including cancellation of any pre-existing dependencies[4]. As Figure 12.24 and Figure 12.25 show, absolute messaging counts were generally similar – with *Replanning* occasionally sending more messages (E.g. in World A for $n_{risk} = 0.75$, for all n_{exo}).

[4] *Continual Replanning* agents formed dependency contracts for all activities requiring delegation in their current plan, as there was no guarantee over when or whether a new plan would be formed.

World A - $n_{exo} = 1$



World A - $n_{exo} = 2$



World A - $n_{exo} = 3$

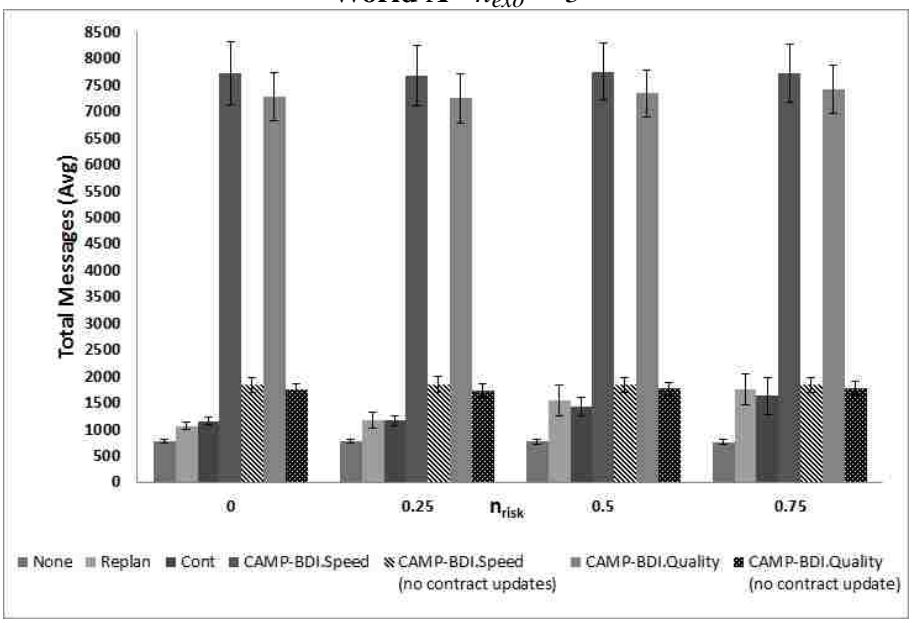
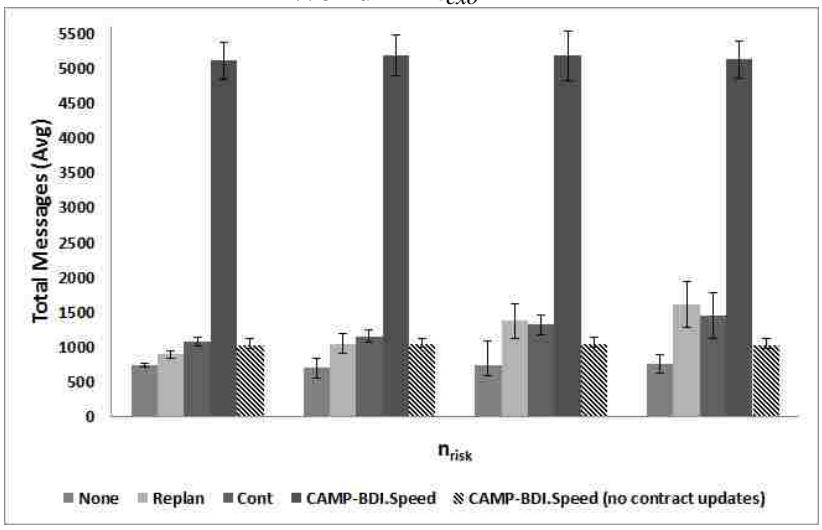
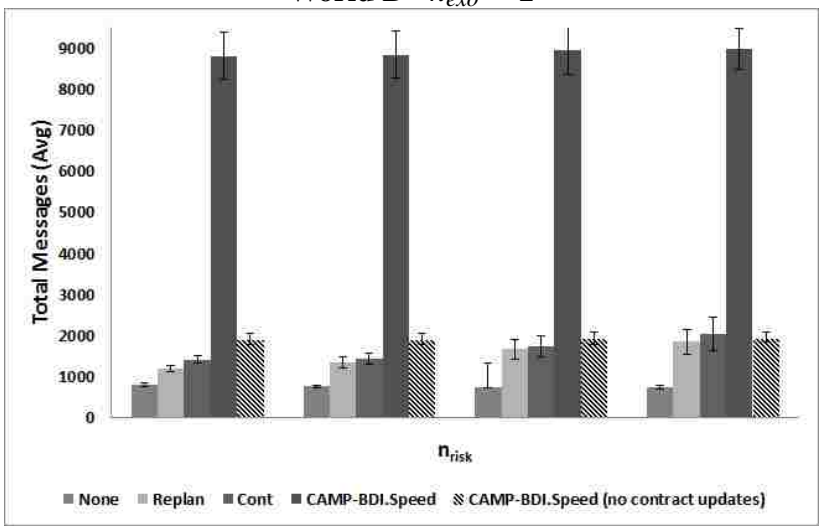


Figure 12.24: Absolute message counts (averaged) in World A.

World B - $n_{exo} = 1$



World B - $n_{exo} = 2$



World B - $n_{exo} = 3$

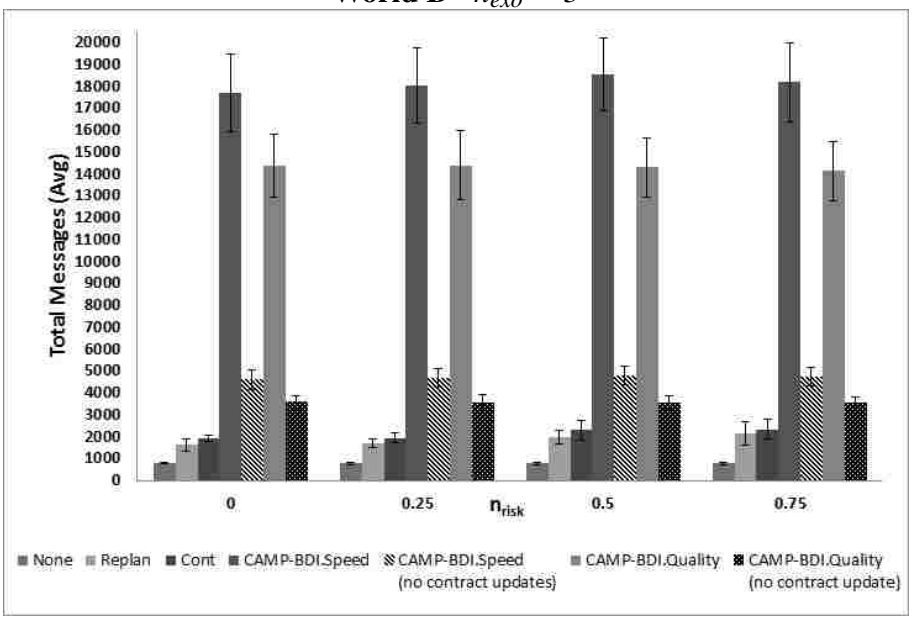


Figure 12.25: Absolute message counts (averaged) in World B.

We attribute these differences to greater activity and success rates for *Continual Replanning*, as agents were less likely to fail in an activity and incur debilitation than *Replanning*. Both replanning approaches adopted decentralized approaches; obligants reported obligation failure if unable to replan following activity failure, with their dependants replanning when a delegated activity succeeded (*Continual Replanning* only) or failed (either approach). *Continual Replanning* had a greater activity success rate due to it's constant revision; lower level agents (such as *Truck* agents being able to reform route plans to avoid failure) suffered less consequent debilitation and were less likely to fail and escalate responsibility to a dependant. *Replanning* agents conversely suffered greater activity failure and consequent debilitation, making local replanning failure more likely (e.g. due to *mortal* damage) and increasing escalation of planning to dependants – who would require alternate (non-debilitated) obligants, requiring formation of new, and cancellation of old, dependency contracts.

Continual Replanning held the lowest per-goal messaging costs of all mitigation approaches. *Replanning* costs rose with n_{risk}, reflecting decreasing goal achievement. In all cases *CAMP-BDI.Speed* costs were significantly higher than all other approaches for all experimental configurations; *CAMP-BDI.Quality* messaging costs were consistently lower than *CAMP-BDI.Speed*, but also consistently higher than all other approaches. This difference decreased with increasing n_{risk}; *CAMP-BDI* (both modalities) consistency in messaging costs (see Figure B.25 and Figure B.26 in Appendix B.6) reflected their consistency in goal achievement, whilst conversely decreasing goal achievement for other approaches increased associated average messaging costs. The lower messaging costs of *CAMP-BDI.Quality* compared to *CAMP-BDI.Speed* can be attributed to reduced average activity costs for the former (Section 12.3); this indicated shorter plan execution, with a reduction in activities likely to be associated with reduced delegation requirements.

Our results show significantly higher messaging costs for *CAMP-BDI* due to continuous messaging of contract updates. These results do present a *worst-case*, as we did not investigate possible optimization and reduction within our implementation – instead focusing on supporting the mechanisms required for robustness. Appendix B.6, Section B.6.8, details specific differences between approaches when *updatedContract* messages were *excluded*; Figure B.31 and Figure B.32 give the exact difference and

significance between *CAMP-BDI.Speed* and other approaches; Figure B.33 provide equivalent information for *CAMP-BDI.Quality*.

When we excluded contract update messages to focus upon messages types associated with cancellation or formation of dependencies, differences between *CAMP-BDI* and other approaches significantly narrowed. In World A, when **updateContract** messages were excluded, *CAMP-BDI.Speed* had significantly lower average per-goal message costs than *Replanning* for $n_{risk} \geq 0.25$ for $n_{exo} = 1$ and 2, and $n_{risk} \geq 0.5$; for *Continual Replanning* in World A, *CAMP-BDI.Speed* sent less messages per goal for $n_{risk} \geq 0.25$ in $n_{exo} = 1$, and for $n_{risk} \geq 0.5$ in $n_{exo} = 2$ and 3. *CAMP-BDI.Quality* shown similar relative results in World A $n_{exo} = 3$, with advantages over *CAMP-BDI.Speed* being both small (< 1.5 average messages per goal) and not significant for $n_{risk} \geq 0.5$.

Results for World B when **updateContract** messages were excluded were similar to those for World A. For $n_{exo} = 1$, *CAMP-BDI.Speed* sent significantly lower messages per goal than both *Continual Replanning* and *Replanning* for $n_{risk} \geq 0.25$. For $n_{exo} = 2$, messaging costs per goal were lower then *Replanning* for $n_{risk} \geq 0.25$, and lower than *Continual Replanning* for for $n_{risk} \geq 0.5$. Finally, in $n_{exo} = 3$ *CAMP-BDI.Speed* sent less messages on average per goal than *Continual Replanning* for $n_{risk} = 0.75$, and less than *Replanning* for $n_{risk} \geq 0.5$. As in World B, *CAMP-BDI.Quality* had lower average messaging costs than *CAMP-BDI.Speed*; this approach sent less messages per goal on average than both *Continual Replanning* and *Replanning* for $n_{risk} \geq 0.5$. In all cases for World A and B, the disadvantages of *CAMP-BDI* approaches against other approaches decreased, and advantages increased, with increasing n_{risk} – reflecting consistent *CAMP-BDI* goal achievement levels, whilst *Continual Replanning* and *Replanning* saw average goal achievement decline – and absolute messaging increase – with increasing risk of post-failure debilitation.

This suggests *CAMP-BDI did* improve stability compared to replanning, by reducing the messaging to cancel or form new dependencies following plan changes – but that this benefit was outweighed by messaging to maintain mutual beliefs. These results also show that consistently higher robustness – i.e. greater goal achievement – can help mitigate higher absolute communication costs, particularly when post-failure debilitation risks stymieing reactive approaches. *Continual Replanning* and *Replanning*

systems also benefited from the assumption *no* communication was required to maintain mutual beliefs during execution; conversely *CAMP-BDI* results represented the *worst-case* scenario, where contract updates occured every reasoning cycle. However, messaging costs remain the principal drawback of *CAMP-BDI*, due to the additional information requirements of threat anticipation.

12.7 Discussion

Our experimental results show superior robustness for *CAMP-BDI*[5] where agents risked debilitation from consequence of failure, with the consistency of goal achievement indicating effective mitigation of debilitative risk – conversely, reactive recovery (*Continual Replanning* and *Replanning*) faced increasing penalties and decreasing robustness under the same conditions. This advantage persisted over increasing perturbation, in both simpler (World A) and more complex (World B) geographies.

In the sole case where *CAMP-BDI* goal achievement was below 90% (World B, $n_{exo} = 3$), our approach maintained consistency in goal achievement and held an advantage over alternative approaches under debilitation risk (i.e. $n_{risk} \geq 0.25$). It is likely reduced *CAMP-BDI* goal achievement for World B $n_{exo} = 3$ reflected greater environmental difficulty – i.e. that increased perturbation and decreased interconnectivity (increasing difficulty when forming alternate route plans for road vehicles, whether to transport cargo or enable delivery through activities such as unblocking roads) made goal achievement less possible overall. This also justified our experimentation in multiple geographies, by allowing analysis of performance within what proved a more difficult environment than World A under the same n_{exo} and n_{risk}.

12.7.1 Goal Success Rates and Activity Costs

Activity success rates show *CAMP-BDI* maintained consistently greater activity success than other approaches. *Continual Replanning* – which allowed a degree of failure prevention as a side-effect of constant plan revision – exhibited superior robustness to *Replanning* but lower than *CAMP-BDI* under debilitation. Our results for

[5] We refer to *CAMP-BDI* where results applied to both *CAMP-BDI.Speed* in all n_{exo} and *CAMP-BDI.Quality* in $n_{exo} = 3$.

CAMP-BDI also evidences the utility of confidence estimation (combined with the *Th* field found in Maintenance Policies) provided through capability modelling, which allowed *CAMP-BDI* agents to react to an increased possibility of failure as well as definitive (expected) failure where preconditions were not expected to hold.

Whilst focusing upon Goal Achievement to verify our hypothesis regarding *CAMP-BDI* robustness, we gathered additional metrics to understand performance – particularly as proactive behaviour risks costs from false positive anticipation of failure. We did not directly compare the quality – commonly viewed in terms of plan length – as each approach would present different forms of planning problem. For example, we could expect *CAMP-BDI* maintenance plans to be shorter then those for replanning, due to our algorithm attempting to minimize the scope of maintenance planning. Instead, we used average activity cost per goal as a proxy, as this metric included activity costs from plans formed to (proactively or reactively) mitigate failure.

In all n_{exo} configurations for World A, and $n_{exo} = 1$ and 2 for World B, *CAMP-BDI* performed less activities per goal on average than *Replanning* once debilitation risk was introduced ($n_{risk} \geq 0.25$), and less than *Continual Replanning* for $n_{risk} \geq 0.5$. *Continual Replanning*'s consistently greater efficiency terms than *Replanning*, and than *CAMP-BDI* at lower levels of n_{risk}, reflects continual optimization from forming plans (i.e. where a newly generated, optimal activity length, plan would replace the current $plan_i$) following every activity; this advantage, however, was associated with additional *planning* costs.

In World B, at $n_{exo} = 3$, agents faced a significantly increased difficulty – although *CAMP-BDI* still shown superior robustness, it suffered significant decreases in goal achievement compared to other experimental configurations (geography and n_{exo} combinations). *CAMP-BDI.Speed* had greater activity costs than reactive approaches, although *CAMP-BDI.Quality* did show lower activity costs over *Replanning* for $n_{risk} \geq 0.5$. This may be partially due to earlier goal failure (or even an inability to form initial plans following cumulative agent debilitations) in both *Replanning* and *Continual Replanning* ($n_{exo} = 3$ saw *Replanning* goal success drop from 45.22% to 15.25%, and *Continual Replanning* from 57.05% to 29.52% over $n_{risk} = 0.25$ to 0.75).

We suggest that the efficiency of *CAMP-BDI* (either modality) may be significantly

reduced in environments where preventative activity is less *feasible*, or debilitation is of lower risk. This – and the other cost metrics gathered – would require domain specific consideration over whether (and how much) additional activity cost is justified to improve robustness and achieve MAS goals. Maintenance Policies can be of use in such cases by allowing control over which agents maintain what activities, and to what level of (confidence) sensitivity.

12.7.2 Planning Costs

Planning costs represented another key consideration, particularity as our algorithms traded off a potential multiplicity of planning operations (associated with increasing scope of maintenance planning) to allow planner agnosticism. This lead to consideration of two planning cost metrics – the average number of operations invoked per goal achieved, and the average execution time (as a proxy for computational difficulty) per operation. These metrics were selected to evaluate the planning 'load' upon agents, and determine if any approach defined planning problems of significantly greater complexity. In the latter case, planning time will result from a combination of multiple factors; as well as different planning contexts and problems presented by alternate approaches, time based results would be influenced by the actual planner implementation and potentially even external factors such as CPU thermal throttling or operating system process management (although we assumed these, if present, were of negligible influence). This leads us to treat planning time results as being indicative, as they will contain elements specific to the *LPG-td* planner used by all approaches.

In terms of individual planning operations, *Continual Replanning* consistently executed more operations per goal – an obvious and expected drawback of it's continual revision approach. In certain cases (E.g. World A, $n_{exo} = 3$) *Replanning* executed a similar average number of planning operations per goal to *Continual Replanning* (albeit with greater deviation); reflecting an increased proportion of planning operations being performed when pursuing delivery goals that eventually failed. Whilst *Continual Replanning* and reactive *Replanning* both saw planning operations cost rise with increasing n_{risk} – and decreasing goal achievement – *CAMP-BDI* (either modality) maintained consistent average planning costs. Where *CAMP-BDI* did have greater *absolute* planning costs (World B $n_{exo} = 3$), these may be justified by superior robustness.

For average planning time, *Continual Replanning* calls generally took less time than in either *CAMP-BDI.Speed* or *Replanning*. This difference is attributed to continual planning also occurring in *non* failure states, which can present simpler planning problems – e.g. if only a few (or even solely one) easily identified activities were necessary to achieve the required goal. In comparison to reactive *Replanning*, where a statistically significant difference did exist, *CAMP-BDI.Speed* spent less time on average for each planning operation. As time results will be partially specific to the *LPG-td* planner itself, we do not definitively conclude *CAMP-BDI.Speed* offers lower individual planning time or easier to solve planning problems. We argue, though, our results provide sufficient evidence that *CAMP-BDI* maintenance does not present *significantly more difficult* planning problems, and that we can reasonably compare planning costs across approaches using a metric of planning operations per goal.

Finally, *CAMP-BDI.Quality* spent overwhelmingly longer each planning operations. This was due to utilizing an alternate planner modality (*LGG-td.quality*), whose execution time was effectively set to last for a predefined period, rather than (as for *CAMP-BDI.Speed*, *Replanning* and *Continual Replanning*) how quickly a plan could be found. This expansive planning time was seen to be of magnified effect within our experimentation due to the truncated duration of activities; in a realistic environment, with longer activity durations, this impact may be less proportionately severe and justified through the reduced activity and messaging costs identified for the quality modality. Another possibility is that we may be able to identify an alternate (pseudo or otherwise) method for probabilistic planning with less time cost – although our review in Chapter 6 (reflected by our consequent assumptions) suggests true probabilistic methods are likely to be intractable in a realistic environment.

12.7.3 Messaging Costs

The primary drawback of *CAMP-BDI* approaches was communications cost. Our maintenance algorithm required continual communication to maintain the beliefs held by both obligants and dependants regarding delegated activity, in order to support distributed maintenance; this entailed a much higher messaging costs per goal, as agents would communicate contract updates after every activity execution.

One advantage *CAMP-BDI did* exhibit was for messaging *excluding* contract up-

dates. This shows *CAMP-BDI* agents generally performed less communication to change (cancel or form) dependency relationships – indicating greater plan stability over *Continual Replanning* and *Replanning*. Like the activity and planning costs benefits identified, this is attributed to greater robustness (higher goal achievement) mitigating any greater *absolute* cost – i.e. less messaging cost was expended (was futile) on ultimately failed goals. Avoiding failure-associated debilitation reduced obligant failure, reducing adoption of maintenance responsibility by dependants *and* the consequent selection of alternate obligants (and therefore formation of new dependencies) by those dependants following maintenance changes.

Our implementation of contract update messaging was relatively crude, as our focus was upon *use* of the resultant information. We believe it may be possible to mitigate costs with optimization work; we discuss several potential approaches in the following chapter (in Section 14.4.3).

12.7.4 Summary of Results

Our experimental results show *CAMP-BDI* offered superior robustness where there is a risk of debilitation, and that excess activity or planning costs were mitigated by superior goal achievement (meaning proportionately less absolute costs were futile). In our experimentation, we employed a single environment – *Cargoworld* – in order to focus upon an in-depth examination of the impact of increasing perturbation and debilitation. We justify this singular domain focus through the widespread existing usage of similar *Transport* domains for planner benchmarking (evidenced by, and a result of, this type's predominance within IPC domain sets) Long and Fox [2000] similarly note transportation is a "*common feature of planning problems, either as a central or an incidental component*" when suggesting methods for inferring the *generic type* for a planning domain.

Our evaluation domain also provided stochasticity and non-determinism through unpredictable exogenous change, and was non-episodic due to the persistence of exogenous changes or agent debilitation. The distribution of agents and resources (i.e. cargo), combined with heterogeneous agent capabilities (including those of logical brokers) mirrored typical motivating factors for MAS approaches, with the enablement (i.e. to allow agents that actually *carry* cargo to achieve their goals) role of

agent types of APC and Bulldozer agents allowing for examination in the context of inter-agent coupling, and providing hierarchical team meta-organizations that reflected similar structures in real-world organizations (such as described by Killion [2000]).

Given the infinite variety of possible real world domains, we will not claim our results inherently hold in all possible circumstances. Rather, we suggest they are indicative for a variety of realistically plausible domains – such as discussed in Chapter 2 when describing our motivation. *CAMP-BDI* is intended, and suited, to agents that perform discrete sequences of goal-driven activity (i.e. based on goal selection and plan invocation), representable in world state transition terms – rather than less plan-orientated agents concerned with continuous monitoring and constraint tailoring such as for power management (Catterson *et al.* [2012]). *CAMP-BDI* is most suited for environments where the rate of exogenous change is high enough to introduce a realistic threat to agent plans, yet not so high or chaotic as to render any form of long term planning – whether for maintenance or to enable resource reservation after intention formation – unrealistic.

CAMP-BDI *can* result in additional costs in more difficult environments, shown by our experimentation in World B $n_{exo} = 3$, where maintenance is effectively only *postponing* inevitable failure. These cases may require work to refine Maintenance Policies to allow the 'target' of maintenance behaviour to focus upon where the relevant $goal_i$ is (believed to be) achievable. Mutual belief maintenance required by *CAMP-BDI* entails an excess communications cost; although we are optimistic future work could reduce this weakness, *CAMP-BDI* would not be suitable for domains where communications cost is of key importance. Any increased costs (of any type) should be considered with regard to the robustness benefit offered by *CAMP-BDI* – in certain domains, excess cost may be *justified* to preserve higher goal achievement or avoid particularly severe activity failure consequences.

Our results for $n_{risk} = 0$ show that *CAMP-BDI* does not hold an advantage over reactive systems where failure is effectively consequence free – this is as expected (we cannot assume *perfect* failure anticipation), and befits our motivating domains being those where debilitative risk *does* exist. In a realistic domain it is likely the risk and types of debilitation will vary based upon the particular activity types; and that in a *practical* implementation *CAMP-BDI* would be paired with complementary robustness

methods, including reactive activity failure handling. This justifies use of *Maintenance Policies* – which allow conditions for triggering maintenance task generation to be externalized and configured on a per agent, per capability, or per agent-capability pair level basis, meaning *CAMP-BDI* behaviour could be reduced in sensitivity or disabled (with failure handled by reactive mechanisms), to mitigate maintenance costs if an activity faces reduced risk of post-failure debilitation.

Finally, we considered *Speed* and *Quality* modalities for *CAMP-BDI* to investigate the increased *CAMP-BDI.Speed* goal failure in World B n_{exo}. This illustrates our algorithm's flexibility in supporting different planning methods (albeit as different *LPG-td* modalities) through generation of different planning problem specifications. We limited our experimentation for *CAMP-BDI.Quality* to $n_{exo} = 3$, as high goal achievement rates in $n_{exo} = 1$ and 2 signified there was insufficient scope to exhibit any robustness improvement; i.e. that maintenance plans using the constrained preconditions employed by *CAMP-BDI.Speed* could be formed without the need to consider use of lower confidence activities, and that *CAMP-BDI.Quality* would therefore likely produce identical plans to *CAMP-BDI.Speed*. *CAMP-BDI.Quality* offered lower activity and messaging cost, at the expense of longer planning duration – although we did not observe superior goal achievement, and actually observed (marginally) inferior robustness in World A at higher n_{risk}. Further experimentation may be useful to differentiate between different planner modalities – or to investigate use of *heterogeneous* planning methods, selected depending upon the computational constraints of individual agents.

12.8 Conclusion

This chapter evaluated the *CAMP-BDI* implementation over multiple experimental runs, for agents acting within geographies of differing complexity, and over increasing and independently scaled probabilities for perturbation and post-failure debilitation. Through comparison against both a reactive post-failure replanning system and one using continuous deterministic replanning, we shown that, where activity failure risks debilitation, *CAMP-BDI* improved robustness over reactive approaches in our experimental *Cargoworld* environment. Although proactive behaviour incurs potential costs from planning operations and maintenance, our results suggest this can be mitigated by overall robustness benefits – although further work will be required to optimize the communication efficiency of mutual belief maintenance.

Chapter 13

Applicability of CAMP-BDI

Chapter 11 evaluated CAMP-BDI within a simulated *Cargoworld* environment – a domain expressly designed to provide explicit specification of environmental properties and allow examination of robustness under scaled risk and perturbation. This chapter further discusses the applicability of CAMP-BDI as a robustness method within the motivating domains discussed in Chapter 2, and for other example real-world application domains.

13.1 General applicability

We can characterise CAMP-BDI as primarily applicable in domains where activity failure results in a debilitated state to hinder reactive recovery, agents form plans in advance of execution (are aware of intended *future* activities), and exogenous changes can be detected. CAMP-BDI requires domains to be relatively *stable* in order to anticipate failure through estimating future execution contexts – i.e. if exogenous change *does* threaten planned activity, it should be reasonable to perform maintenance under the assumption threats are unlikely to be removed by later exogenous change. This domain stability can be considered as an inherent requirement for agents to employ deterministic plans *in general*, as constant, chaotic levels of exogenous change would otherwise render activity post-conditions too unpredictable.

CAMP-BDI may carry additional costs, particularly from communication, meaning use must be balanced against the potential risks of debilitation (considering both failure of the initial activity and any subsequent recovery activities) and the cost of goal failure(s). In the latter, we observed additional per-goal costs (in activity, commu-

nications or planning tems) of CAMP-BDI may be justified where goal failure carries sufficient consequences – such as destruction of resources, or even loss of life. The importance of contract update messaging within distributed maintenance means the viability of CAMP-BDI may be challenged where communications are unreliable or required resources are limited, although optimisation of communications requirements remains an area for future work (as discussed in Chapter 14).

A key element of our contribution is CAMP-BDI's provision of *distributed* maintenance behaviour; whilst still applicable for individual agent robustness, CAMP-BDI will offer greater benefits in a *multi* agent, team-forming system. Similarly, our capability based approach supports heterogeneous agent sets through providing a generalized model, intended to allow sharing of information without requiring *semantic* knowledge be held and understood by dependants. In homogeneous agent systems (i.e. with limited agent types), the associated homogeneity of agent capabilities may make it practical to implement domain specific robustness methods that *share* (and use) semantic knowledge between agents. While CAMP-BDI is still applicable within such homogeneous MASs, it may be more effective to implement domain-specific robustness approaches if agents *can* be assumed to have semantic understanding of each others' capabilities.

13.2 Space Domains

Chapter 2, discussed the International Planning Competition (IPC) *Rovers* and *Satellites* domains (Fox and Long [2003]), concerned with activities outside of Earth and deriving from real world NASA (National Aeronautic and Space Agency) scenarios. These domains did not model exogenous change or debilitative failure consequences, due to their origins as IPC domains for planner (and plan quality) evaluation, rather than to test agent handling of environment change. We have suggested plausible extension to include such properties, and argue such extension can be justified with the existence of real world equivalents. Autonomous behaviour is an area of active research interest (such as NASA's Intelligent Systems project, described by Morris *et al.* [2004]) in real-world equivalent Space domains, due to the costs associated with Earth based personnel and the time latencies of communication (line-of-sight based communication also risks being stymied by the relative positioning of the involved parties). This suggests similar benefits in augmenting such autonomy with proactive robustness

approaches (such as CAMP-BDI) – particularly as the sheer distances involved can heavily restrict reactive repair of, or recovery from, failure-associated damage.

The inherent uncertainty associated with exploration suggests a strong need for robustness approaches to allow adaptation to new discoveries and threats; this particularly applies if plans are being generated offline and transmitted from Earth, as is typical for planetary rovers. For example, the CASPER (Continuous Activity Scheduling, Planning and Replanning) system (Estlin *et al.* [2003]), intended for use in automated terrestrial rovers or spacecraft, addresses uncertain environments through continual planning and by giving the ability to modify goals in response to exogenous events – for example, removing goals upon unexpectedly high energy use during movement. This motivation for CASPER also applies to justify CAMP-BDI, which additionally offers support for robustness within the context of *distributed* activity.

In both the *Satellites* and *Rovers* domains, the potential severity of failure consequences supports the likelihood of detailed domain and sensory information being available to support capability modelling. The distances that agents may be required to travel suggest CAMP-BDI's proactive behaviour can expand preventative options through earlier response – for example, allowing a backup satellite to re-orient in a new orbit, ready to receive communication or perform observations when serving as a new obligant.

Confidence estimation can be valuable due to allowing small sensed faults – otherwise insignificant in precondition terms – to be related to greater long term threats. As part of the NASA Intelligent Systems project, Hofbaur and Williams [2002] describe use of Hidden Markov Models to predict the probability of 'failure modes' arising given current states; their approach seeks to identify where small state changes, individually indistinguishable from sensory noise, combine to risk future catastrophic failure. Like our confidence model, they seek to detect if seemingly insignificant (i.e. non-preconditions violating) state changes increase failure *risk*, and to allow preventative response. Their approach can be seen as validating the utility of CAMP-BDI's preventative focus *and* the use of capability confidence estimation within such domains; it may also provide a method *for* estimating confidence (generalization within our capability model also allows flexible implementation of confidence estimation).

The *Satellites* domain is inherently more multiagent in nature, due to being based around *constellations* – groups of satellites equatable to an agent team. Damiani *et al.* [2005] describes a satellite constellation system (*D-SpaCPlanS*, or *Distributed Spacecraft and Coordination Planning and Scheduling*), based upon a French Space Agency mission definition. The organizational structure and description of decentralized behaviour in *D-SpaCPlanS* closely resembles our supporting assumptions regarding hierarchical (dynamically formed) teams, as used in design of CAMP-BDI distributed maintenance. In contrast, the distances and difficulty of interplanetary transport suggest *Rovers* domain instances are likely to see only a limited number of physical vehicles represented within a MAS.

The exploratory nature of *Rovers* domains may also see such agents as physically distant, and raise challenges in terms of the ability to perform communications required for distributed maintenance (of course, this raises the issue of whether such distance is prohibitive of co-operative teamwork anyway). One other possibility is to model individual agents as *holons* (Schillo and Fischer [2003]), with constituent physical components (e.g. wheels, motors, effector arms) *also* modelled as individual agents. Physical rover activities would consequently resolve into planned sequences of physical component activities, where involved component agents form a team, acting as obligants to the dependant rover agent. CAMP-BDI can offer further utility under such an approach; for example, dangerous temperature buildup in motor components could be accompanied with decreases in confidence, with maintenance behaviour being triggered to switch component use and avoid failure and potential damage. The benefits of proactivity could be further pronounced if individual component failures more risk widespread impact – such as if the aforementioned motor overheating lead to a crash, causing damage to multiple components. Such behaviour does rely upon alternate capabilities existing; the potential benefits of CAMP-BDI are likely to be limited where components are highly specialized and no alternative means exist (maintenance plans cannot be found) for a threatened activity – although such component specialization could also constrain *reactive* recovery, following component damage from failure.

The uncertainty present will vary between both Space domains. In *Rovers*, where agents will be exploring a new region of a (most likely) lifeless planet, the principle source of uncertainty is likely to stem from a lack of geographic knowledge – i.e. where belief changes arise from sensing new information about a relatively static en-

vironment, rather than environment *change*. Where environment change does occur, we suggest it is most likely to stem from changes in weather systems – which may be detectable in advance and from distance (particularly if supporting resources such as satellites or remote sensors are present). Given this, it is plausible to estimate future execution context from current state, and to consequently anticipate activity failure – aiding the efficacy of CAMP-BDI.

In the Satellite domain, the sources of uncertainty may present greater challenge to CAMP-BDI. Threats such as micro-meteorites or debris impacts (suggested by Damiani *et al.* [2005]) may occur without warning, preventing any pre-emptive behaviour[1]. CAMP-BDI will obviously not offer utility where such events cause immediate failure of a currently executing activity; but if impact damage does not cause immediate failure but results in a gradual debilitation and degradation of performance, then this can be represented by capability confidence loss – and with CAMP-BDI able to offer proactive, preventative robustness behaviour in response.

Perhaps the main difficulty with employing CAMP-BDI in Space domains stems from resource limitations. Limited energy resources may serve to prohibit the frequency of communication required by our distributed maintenance, and may prevent runtime maintenance planning by limiting computational hardware resource. Although CAMP-BDI is planner agnostic, limited environment knowledge (e.g. if exploring a new region) may restrict the viability of recipe based maintenance planning approaches, much as uncertainty hinders the use of offline generated and transmitted initial plans. The decision whether to adopt CAMP-BDI for robustness within a space domain, will therefore likely be determined by whether benefits of avoiding activity failure (and debilitation) outweigh the associated energy costs of performing computation and communication.

13.3 Transport Domains

Transport domains, describable as *logistics* or *mobile problem* domains, present scenarios where goals are achieved through movement of some portable object from an initial to goal location. As Long and Fox [2000] note, mobile problems are common

[1] Although it is possible *some* orbital threats such as larger pieces of debris may be tracked, and therefore may lead to state changes that then trigger CAMP-BDI maintenance to avoid impact.

(implicit or explicit) aspects of many agent domains. The *LS/ATN* (Living Systems / Adaptive Transportation Networks) represents one real world MAS application (Dorer and Calisti [2005]) in the logistics domain, adopted by ABX (a European logistics company) due to (experimentally proven) efficiency benefits in truck utilization. Like CAMP-BDI, LS/ATN models heterogeneous agents (vehicles vary in properties and capabilities, with differences including cargo capacity and load/unload abilities), and respond and reconfigure plans upon disruption from exogenous change (e.g. traffic, breakdowns or accidents). Pokahr *et al.* [2008] also note the appropriateness of MAS approaches within transport domains, by defining simulation methods to ease the transition of MAS implementations from simulated experimental evaluation to real-world application use.

The *Cargoworld* domain shares similarities with – and partly derives from – mobile problem domains, through adoption of a road network and generation of cargo delivery goals. However, *Cargoworld* does present a more heterogeneous agent-capability set, and is more explicitly designed to entail decompositional team formation, than the domains described here. Regardless, we suggest many of Chapter 12's observations of CAMP-BDI performance are also applicable to this domain type.

We suggest CAMP-BDI is beneficial in such domains due to the usage of a road network. Plans will inherently involve sequences of deterministic movements along roads; CAMP-BDI can therefore relate detected exogenous change threatening intended use of roads, to requirements for plan maintenance, allowing proactive avoidance. We can intuit proactive robustness will offer benefits over reactive response in terms of reduced backtracking, through allowing earlier route plan changes (rather than only upon the point – and location – of failure). Additionally, (at least some) movement failures can be reasonably associated with debilitation risk – as if the agent were still *able* to move they would likely have completed said movement. Finally, if current conditions cause movement to fail midway along a road, these will likely *further* hinder any reactive recovery movements required to use that same road.

The *Tileworld* agent domain (Pollack and Ringuette [1990]) presents a highly abstracted environment, where CAMP-BDI is of limited utility; the lack of debilitative consequences to failure makes reactive recovery an effective strategy, with the fully connected grid geography minimizing backtracking costs. Whilst *Tileworld* can be ex-

tended with properties more suitable for CAMP-BDI, extension also risks introducing an inadvertent bias *towards* our approach (a risk noted by Lees [2002] for extending *Tileworld* in general). As such, we will focus upon more realistic domains when evaluating CAMP-BDI applicability – particularly as *Tileworld* is sufficiently abstract that it could arguably be extended to *form* many of the domains discussed in this chapter.

The *Triangle-Tileworld* domain (Little and Thibaux [2007]) represents an extension of Tileworld for probabilistic planner evaluation, using a triangle rather than grid geography. Again, this presents an abstracted, unrealistic domain which – being an IPC domain for evaluating the quality of formed plans – does not model exogenous change; this means CAMP-BDI cannot offer a robustness benefit without domain modification to include exogenous change events, under the same bias risk as noted for *Tileworld*. Although failure and debilitation probabilities – i.e. of a puncture occurring – are modelled for road movement, these are constant and intended to be handled during plan formation. Finally, whilst this domain *does* offer the potential of an agent to load spare tyres to prevent total goal failure, doing so only facilitates *recovery* from failure, rather than offer a means of prevention.

The applicability of CAMP-BDI increases within the more realistic scenarios presented by the *Trucks* (Edelkamp *et al.* [2011]) and *DriverLog* (Gregory and Lindsay [2007]) IPC domains or the *Truckworld* (Hanks *et al.* [1993a]) simulated environment. These domains present mobile problems, concerned with use of a truck to deliver some package to a given destination. As the two IPC domains do not present exogenous change or debilitative failure (due to being for evaluation of generated plan quality rather than agent robustness), we focus upon the *Truckworld* simulated environment – which does provide such characteristics, as an experimental testbed for reactive planning by agents[2].

These domains may be single or multi-agent; in the latter case, the mobile problem includes determining the appropriate vehicle to perform a delivery goal, implying a manager agent exists that can perform such delegation. In the single agent case, CAMP-BDI's appropriateness would be reduced due to the inapplicability of our *distributed* maintenance design, and with the absence of options for failure prevention

[2] As *Trucks*, *DriverLog* and *Truckworld* present very similar problems, we suggest the environment properties of the latter provide plausible guidance for the properties of any realistic simulated environment for the former two.

through delegation and team formation (a reduced set of agents would also reduce *reactive* recovery options). As noted earlier, CAMP-BDI is more appropriate in heterogeneous multi-agent domains, where *distributed* maintenance can be employed and where – depending upon advertisement visibility – maintenance planning can take advantage of the more varied options available with a heterogeneous agent capability set.

In *Truckworld*, exogenous changes include roads becoming muddy under rain, or placement of bombs; failure-associated debilitation includes agents skidding off roads under the former and being damaged by explosions from the latter. CAMP-BDI can respond to such exogenous threat states through maintenance to avoid activity failure, avoiding such risks of debilitation. These scenarios closely correlate with *Cargoworld*, which similarly models wet (i.e. flooded or slippery) road states, and where a *dangerZone* at a junction has consequences equivalent to the presence of a bomb. Aside from general benefits from reducing backtracking, the efficacy of CAMP-BDI is aided through domain specification of advance *mitigatory* activities (e.g. fitting tyre chains before travelling along wet roads) to reduce failure risk.

Although *Truckworld* presents a homogeneous agent set (i.e. only contains a Truck vehicle), it models the constituent *components* of such an agent; such as fuel tank, tyre, loading arm and cargo bay state. This information can be employed towards capability modelling, both to specify activity preconditions and consider component state changes as part of confidence estimation. Modelling of component internal health state (which also implies corresponding introspective sensory capacities) aids CAMP-BDI by providing detailed information for capability modelling and advertisement, which can be used to improve the effectiveness of maintenance reasoning.

The value of detailed component health information will partly depend on whether agents can repair themselves (i.e. improve health state) or whether similarly capable agents exist in the domain. If a truck can sense its own suboptimal health, but neither repair itself or see a dependant agent reassign consequently threatened goals (i.e. through distributed maintenance processes) to an alternate obligant, then any sensory capacity and CAMP-BDI's proactive approach will hold little benefit – as no options will *exist* for prevention, regardless of the ability to anticipate failure.

One challenge for CAMP-BDI within *Truckworld* stems from the *visibility* of exogenous change. *Truckworld* can model distance and noise constraints upon agent sensing, potentially restricting the ability of trucks to detect change. This can restrict the viability of CAMP-BDI by preventing or delaying detection of exogenous change until close to execution of a threatened activity (although this can be partly mitigated if agents are sharing observations with each other). In the former case, CAMP-BDI obviously cannot anticipate a failure if unable to detect the causal state; in the latter, CAMP-BDI can prevent failure but risks losing benefits associated with earlier detection. This issue of sensory noise and information availability is applicable to *all* potential CAMP-BDI application domains, and must be considered as part of any decision whether to employ CAMP-BDI as a robustness approach within that domain.

13.4 MAS Disaster Response Domains

The real world application of MAS technology for Disaster Management is of increasing research interest (Jain *et al.* [2012]), particularly given recent natural (e.g. the 2004 Asian Tsunami of 2004, Hurricane Katrina in 2005 or 2010 Haitian Earthquake) and man-made events (e.g. 9/11 or London 2005 terrorist attacks). Such domains are concerned with management of emergency services, including use of military and/or humanitarian aid forces to perform rescue and evacuation – in the military context, *Military Operations Other Than War* or *MOOTWs* (Smart [2008]). Chapter 2, discussed three disaster management domains; *Pacifica/PRECiS* (Planning, Reactive Execution and Constraint Satisfaction) (Reece *et al.* [1993]), *Blogohar* (Sensoy *et al.* [2010]) and *Robocup Rescue* (Kitano *et al.* [1999]) – all of which see heterogeneous agents co-operate to perform emergency response tasks over a distributed geography.

Emergency response domains present scenarios where goal failure may hold severe consequences; such as failure to evacuate civilians leading to injury and death in *Pacifica* scenarios, failure to escort humanitarian vehicles leading to their destruction in *Blogohar*, or failure to extinguish fires leading to building collapse or civilian casualties in *Robocup Rescue*. These domains present significant uncertainty; as well as the possibility of exogenous events such as rainstorms or landslips (Pacifica), insurgent activity (Blogohar) or fire spread and building collapse (Robocup Rescue), time constraints (i.e. a need for immediate, rapid response) upon disaster scenarios may force the adoption of intentions while the exact environmental state is still being discovered.

The organizations in such domains often adopt hierarchical command structures – such as *Strategic-Tactical-Operational* models (Killion [2000]) – which reflect assumptions behind (and supports the utility of) CAMP-BDI distributed maintenance design.

Unlike the transport domains in Section 13.3, the heterogeneous nature of the MAS introduces the possibility of agents employing their or obligants' capabilities to remove threatening states (e.g. prohibiting travel through a road or location). This allows CAMP-BDI to provide further benefits; proactive maintenance planning can identify suitably capable agents for performing preventative activities to explicitly remove failure causing states, and avoid contention over necessary agent resource through enabling earlier identification of, and contract formation with, required obligants.

CAMP-BDI allows agents to adapt intended plans following belief changes; this is particularly beneficial upon discovery of states that add *danger* to an intended activity, and consequently increase the risk of debilitation as a cause or consequence of subsequent failure. These domains also present non-episodic environments (e.g. *Blogohar* scenarios last over two days), such that post-failure debilitation can have lasting impact upon future activities – and that avoidance of failure and debilitation holds implicit benefits beyond achieving the current goal. Our confidence model allows recognition of where such dangerous states do not *prohibit* activity, but do increase risk – and our maintenance policy approach allows control over whether agents perceive such risk as prohibitive (i.e. requiring maintenance), with an ability to perform runtime modification to reflect changes in environment state or agent set.

Pacifica defines a number of heterogeneous agent types and capability sets (for example, aircraft have different constraints upon operation including turnaround time, carrying capacity, and which runways can be employed), supporting the utility of our generalized capability model and allowing a range of adaptive options for CAMP-BDI maintenance to employ. *Blogohar* similarly models humanitarian and military agents, where communications restrictions may exist – these can be reflected through constraints on capability advertisement, and are further supported through our capability model explicitly *not* requiring communication of semantics (which may be barred by such restrictions). *Robocup Rescue* does support a reduced degree of heterogeneity, as each key activity in the domain (namely, clearing roads, providing medical care or extinguishing fires) can only be performed by one agent type – although we can still

consider property changes within maintenance reasoning, such as setting confidence in a fire engines' capability to extinguish fires to reflect current water reserves.

The challenges to CAMP-BDI identified within Transport domains in Section 13.3 similarly apply within emergency response domains – namely, the risk of communications difficulty, and potential limits to sensory perception. There is a risk that some exogenous changes which threaten activities may only occur at execution, and be undetectable or unpreventable – particularly in the case of *hostile* actors such as insurgents within *Pacifica* and *Blogohar*. Such changes present threats and are perpetrated in a manner designed to *avoid* detection. CAMP-BDI may still be offer benefits in these scenarios, however – if knowledge requirements are modelled within activity preconditions, or lack of knowledge reduces confidence, maintenance can modify plans to include active sensing in response. For example, a lack of knowledge regarding the safety of a location in *Blogohar* may manifest as reduced confidence in moving through that location – with the generated maintenance plan including activities to send reconnaissance units that will perform the required sensing.

Cargoworld bears a close similarity to – and indeed was inspired by – these domains, including the use of a road network, the presence of heterogeneous agents either able to achieve the system goal or perform enabling activities, and equivalent exogenous threats. Our cargo delivery goal can be considered as, in abstracted terms, equivalent to mobile problems solved by agents in these domains – i.e. unloading cargo objects at a destination can be reduced to a task to travel to a defined location and perform a goal achieving activity. For example, *dangerZones* resemble the threats of IEDs or hostile areas in *Blogohar*, and road blockages (landslips or floods) can be seen to derive from similar threat events in *RoboCup Rescue* (building collapses) and *Pacifica* (which sees similar threats to traversal). As such, we suggest our *Cargoworld*-based evaluation results can be considered broadly applicable towards these domains.

13.5 Further Industrial Application Domains

A number of real-world industrial applications also suggest possible domains where CAMP-BDI may be applied. In a survey of AAMAS'05 (The 2005 International Conference on Autonomous Agents and Multiagent Systems) industry track highlights, Pěchouček *et al.* [2006] highlight domains including where data for decision making

is decentralized, and those requiring time-critical response and robustness – the former requirement being accounted for in CAMP-BDI's distributed maintenance design assumptions (i.e. by using a general capability model design to allow communication of activity performance information, without mandating sharing of semantic details, and allowing autonomous local adoption and performance of maintenance, without centralized control), and the latter corresponding to the purpose of our contribution.

Pěchouček and Mařík [2008] review further examples of industrial MAS deployments, noting that automated reconfiguration is a highly desirable property for manufacturing tasks, as is collision avoidance within robotics; the former of these can be provided through CAMP-BDI's decentralized, distributed maintenance design (where dependants effectively 'reconfigure' teams if maintenance determines new activity sequences and obligant sets), and the latter as a general property of our proactive focus (i.e. avoiding failure includes those due to collisions).

As exogenous change is an inherent property of real world environments, agents in industrial applications must handle uncertainty and adapt plans accordingly to change. We consider CAMP-BDI as generally offering benefits in those domains where failure causes are anticipatable, it is feasible to assume communication is possible between agents (to facilitate information sharing upon contracts and capabilities, to enable distributed maintenance and team formation), and where it is possible for failure to result in difficult – or costly – to recover from scenarios.

One potential application for CAMP-BDI lies within UAV domains, where agents control and co-ordinate Unmanned (autonomous) Aerial Vehicles. For example, Baxter and Horn [2008] describes a system, developed as part of research program for the British Ministry of Defence, where a human user controls a team of autonomous UAVs by assigning tasks to locate or destroy targets. Ondráček *et al.* [2015] also describe an application where UAVs monitor oil pipelines in remote or dangerous areas The MAS must both organize the allocation of recharging stations and maximize monitoring coverage; failure of the latter risks monitored infrastructures being damaged (by natural or artificial causes).

In these domains there can be obvious benefits from proactive robustness; both in avoiding potential debilitation, and (as UAVs frequently serve reconnaissance pur-

poses) in preventing dangerous exogenous changes being missed when active sensing (i.e. information gathering performed through planned activity) fails. Challenges to CAMP-BDI use will again stem from whether threats can be anticipated (i.e. whether threats from exogenous changes can be detected in advance of activity execution), and to what degree preventative activity is possible. For example, a UAV controller may be able to respond to higher than expected power use from a UAV obligant (as modelled through confidence decreases) by reassigning tasks to another – but ambushes from hidden hostile forces in a military domain may be less preventable and require reactive recovery. Of course, if enemy forces *are* detected, CAMP-BDI can offer mitigation by triggering maintenance (based on reduced confidence, or violated preconditions) such that UAV adopts a less dangerous route or calls in armed escorts. Again, a benefit of CAMP-BDI is that such behaviour can be performed pro-actively rather than in reaction to failure, allowing earlier identification and reservation of required resource.

Somewhat similarly to Space domains, UAV domains may involve agents operating at great distances, and with limited power and communications ability. This means that the communications required for distributed maintenance may prove difficult, even with our assumption of *other* robustness methods to ensure correctness (i.e. against noise or corruption), due to positional constraints upon line-of-sight communications, or the energy costs of transmission. Further difficulties may stem from weight limits upon a physical agent, which may restrict the ability to provide computational resources (hardware) for maintenance reasoning – particularly if maintenance employs (more computationally demanding) runtime planning.

Manufacturing system domains present dynamic environments, where agents co-operate to perform some manufacturing process; activities involve material transport of material and tool deployment, as well as assembly of products. Leitão [2009] note an increasing tendency of manufacturing organizations to specialize and co-operate to maintain competitiveness; this is unsuited to traditional centralized manufacturing processes (where, as Colombo *et al.* [2005] note, one failure can shut down the entire system), but where a MAS approach provides the required flexibility and adaptability. They note such environments may be chaotic, with "*unexpected disturbances that leads to deviations from the initial plans and usually degrades the performance of the system*" – highlighting the same risk from exogenous change that CAMP-BDI seeks to address, and which *Cargoworld* provides within our evaluation.

For example, Bussmann and Schild [2001] describe a manufacturing domain, successfully prototyped by the DaimlerChrysler car manufacturer, where production is optimized through using agents to represent manufacturing machines, with dynamic assignment of work performed through an auction system (rather than through advance scheduling).

Mařík *et al.* [2005] describe the MAST (Manufacturing Agent Simulation Tool), a simulation developed at Rockwell Automation, which served as a demonstrator for flexible manufacturing processes. MAST included simulated events, such as broken components that force adaptive system responses – e.g. re-routing products to avoid a broken conveyor belt. Intended to aid the practically applicability of MAS approaches, MAST was employed to develop a reconfigurable control system for a US Navy Ship Chilled Water System (CWS). Here, agents represent physical components of the CWS, with diagnostics models and plan recipes (for fast reaction) reconfiguring agent relationships in response to partially damaged equipment.

Pěchouček *et al.* [2007] described *ExPlanTech*, a multi-agent planning technology, developed for and industrially deployed by SkodaAuto to mass-produce car engines. *ExPlanTech* provides long term (six week) production plans, employing monitoring tools to perform system reconfiguration and real-time replanning to respond to changes in production demand or anomalies (errors) such as late material arrivals. Planning in *ExPlanTech* is decomposed, with agents attempting to resolve detected errors locally, before escalating responsibility to manager agents – this behaviour resembles that of CAMP-BDI distributed maintenance.

Manufacturing domains present a heterogeneous agent environment, with agents representing physical resources such as machine tools, robots, automated vehicles, products or logical objects such as schedules. The use of long term production plans, combined with a likelihood of exogenous change during execution, means the proactive behaviour offered by CAMP-BDI may be beneficial by preventing failure. Although delays to expected production are more likely consequences than outright agent debilitation, failures may still lead to the loss – with associated costs – of manufacturing material. Our confidence model may be employed to represent fatigue upon agents *if* sensors exist to detect such information; if agent activities are at risk from fatigue,

the resultant confidence loss can trigger mitigatory maintenance planning, allowing the obligation to cancelled before failure occurs and risks more permanent damage. This release of agents from obligations may also allow agents to adopt (self) repair goals, previously barred by the commitment to the (cancelled) obligation.

We would not expect communication to pose as significant an issue within such domains, as agents are likely to operate within a close physical environment (e.g. a factory or warehouse), and similarly are likely to have a reliable, consistent power supply (to support continuous manufacturing). Our assumptions of contract pre-formation are likely to hold, as efficient manufacturing will require ensuring the necessary agents (e.g. tools or transporters) are available for use. The main challenge to the applicability of CAMP-BDI is likely to arise from whether the consequences and costs of failure exist to justify the effort of capability specification, and whether reactive recovery methods would suffice. This may depend upon the specific products being manufactured, and the risk of damage to them from failure; for example, assembling a cheap disposable widget will justify preventative costs less than assembling a more expensive car engine.

One benefit of our capability advertisement approach can be to enable a degree of dynamic optimization. If performance of an agent drops, this can be reflected in decreased confidence; if the MAS has seen changes to the agent-capability set (e.g. addition of some newly purchased machine or tool, accompanied with advertisement of their capabilities), triggered maintenance planning will employ an updated *EC* set - allowing consideration of the newly added system agent capabilities. This benefit may be more generally applicable, in domains where the MAS is *open* (allows dynamic addition of agents, provided they perform the contract formation and capability advertisement required by CAMP-BDI) – including those previously considered in this chapter (particularly as disaster management domains may see addition of new agents, where governments and humanitarian organizations may dispatch assistance as the situation unfolds).

13.6 Conclusion

This chapter discussed how CAMP-BDI may be applicable within other domains, and considered selected real world application domains. The exact suitability of CAMP-

BDI will vary depending upon domain specifics – including the risk of debilitation following failure, the ability of agents to detect exogenous change and relate it to threats upon activities, and whether it is plausible to avoid activity failures given the capabilities existing within the MAS. Constraints upon our approach centre around whether computational, energy and communication resources exist to enable CAMP-BDI behaviour, particularly within the context of capability advertisement and distributed maintenance behaviour (i.e. for contract updates). However, we suggest our approach can offer broad applicability as the core domain properties targeted by CAMP-BDI – exogenous change during plan execution and debilitative failure – can be viewed as common within many realistic application domains.

Chapter 14

Conclusion

This thesis presented CAMP-BDI, an approach for proactive plan execution robustness. CAMP-BDI provides BDI agents with algorithms (and supporting meta-knowledge) for anticipation of future activity failure, and for performing modification of intended plans to avoid that failure. Although existing BDI frameworks typically employ reactive behaviour to recovery from activity failure, this may be stymied in domains where failure risks debilitative consequences. In Chapter 2, we described examples of such motivating domains based upon existing planning and multiagent domains. We also introduced the *Cargoworld* domain, employed for both examples of agent behaviour and experimental evaluation.

Chapter 3 focused upon understanding the critical mental state and behavioural components of BDI rationality, and their extension to multiagent activity. Agent robustness approaches were discussed in Chapter 4, allowing our approach to be placed within the context of general agent robustness and providing a definition of robustness for use in evaluation.

Plan formation and execution were identified as critical in BDI rational behaviour, leading to a focus upon *plan robustness*. Chapter 5 examined automated planning, reasoning that information used in plan formation could be applied to reason about plan activities during execution – allowing identification of knowledge requirements for our capability model. Methods for avoiding failure during planning or execution were evaluated in Chapter 6; this established the necessity of plan failure mitigation in realistic domains, and provided models for triggering or performing such mitigation (such as Plan Execution Monitoring and Plan Repair).

Our literature review led to specification of desired behaviour in Chapter 7, which extended the *Cargoworld* to describe an example MAS – subsequently used for both illustrative examples and as a specification for experimental implementation. Our design was presented in the following chapters; covering provision of required information (the *supporting architecture*, in Chapter 8), agent-level algorithms using that information to identify and handle threats (Chapter 9), and finally extension of local behaviour to perform decentralized, distributed maintenance (Chapter 10).

Our experimental CAMP-BDI implementation was evaluated against MASs employing the same agent design, but with (reactive) Replanning or Continual Replanning strategies. Chapter 11 described our experimental protocol, which evaluated CAMP-BDI within a *Cargoworld* simulation over scaled levels of perturbative exogenous change and debilitative failure risk. We evaluated overall robustness, and the (per-goal) efficiency in terms of three factors – activity cost, planning operations, and messaging cost.

Our results, given in Chapter 12, supported our hypothesis by showing, if failure risked debilitation, CAMP-BDI offered superior robustness over the other evaluated approaches for all experimental configurations. Under the majority of circumstances, CAMP-BDI also had lower per-goal activity and planning costs. Although high messaging costs from contract updates represent a disadvantage of CAMP-BDI and an area for future investigation, our results also show CAMP-BDI generally sent less messages concerned with dependency formation or cancellation – indicating plan stability benefits over other approaches.

Finally, Chapter 13 discussed the general applicability of CAMP-BDI, considering potential benefits (and challenges) within other planning domains and agent environments (including the motivating domains described in Chapter 2).

14.1 Contributions

The core contribution of this thesis is the *CAMP-BDI* (Capability Aware, Maintaining Plans - BDI Agents) approach for proactive plan modification; a design for improving plan execution robustness in domains where activity failure risks consequent debilita-

tion. This can be divided into the following sub-contributions:

- **Algorithms for performing pre-emptive maintenance**, which use introspection to identify activities at risk of failure following exogenous change and performs plan modifications to prevent failure of the relevant intended plan. CAMP-BDI extended the generic BDI reasoning cycle (Rao and Georgeff [1995]), to first form a priority ordered agenda of *maintenance tasks* – each representing a plan activity at threat – and then to handle threats defined by that agenda.
- A **capability meta-knowledge model, supporting introspective reasoning in maintenance algorithms** by enabling both threat identification (deterministic *or* non-deterministic) and specification of planning problem goals for handling resultant maintenance tasks. Our capability definition also provided a shared model for communication regarding *delegated* activity, and allowed maintenance to use the same algorithmic reasoning for internally and externally performed activities. Although capability knowledge suggests additional specification costs, we argue much of the required information is necessary *regardless* for forming operator specifications or plan libraries. Specification costs may also be mitigated by potential applicability elsewhere in BDI reasoning, such as to guide desire and intention selection.
- **Structured messaging behaviour to extend individual agent maintenance behaviour into the distributed case** of a plan-executing team, including provision of obligation and dependency knowledge within our supporting architecture. CAMP-BDI uses post-maintenance contract updates to drive autonomous adoption of maintenance *responsibility* by dependants, providing distributed maintenance behaviour that mimics re-refinement HTN plan repair.
- **A policy based mechanism to tailor maintenance**, allowing runtime modification of key variables and constraints used by our algorithms. This employed policies as behaviour modifiers, providing both a mechanism for tailoring maintenance costs and a framework for further extension (e.g. to support reuse of CAMP-BDI agents across different environments).

CAMP-BDI provides a novel approach, embodying agents with capability knowledge and proactive reasoning for failure avoidance; in our literature review we did not identify any other approaches which defined capabilities for this form of introspective reasoning (e.g. Busetta *et al.* [2000] and Braubach *et al.* [2006] view capabilities

as modular components for *building* agents). Although Plan Execution Monitoring (PEM) or replanning (such as *FF-Replan* by Yoon *et al.* [2007]) respond to divergence between actual and assumed states, none of the approaches surveyed employed the same combination of approaches as CAMP-BDI; i.e. using proactive threat identification (similar to PEM) to drive plan repair, and to define such within an explicit context of BDI behaviour and multiagent team activity.

We do not suggest CAMP-BDI can *replace* reactive methods; we cannot assume all failure is preventable, and some failures may be trivially easy to reactively recover from (i.e. reducing the value of our approach). However, our contribution provides a *complementary* approach which can offer robustness benefits where debilitative failure consequences exist to (potentially) stymie reactive methods. Our experimental evaluation shows beneficial performance within the context of a transport domain; Long and Fox [2000] note this generic type is common to many domains, whether as an explicit or implicit aspect. This suggests the robustness benefits of CAMP-BDI may be applicable beyond the specific *Cargoworld* environment or Logistics domain – although further research will be required.

14.2 Discussion

In this section, we will discuss the outcomes of the work presented in this thesis; both in terms of our original research aims, and also in assessing our contribution.

14.2.1 Achievement of Research Aims and Objectives

Section 1.3 defines this thesis as aiming *to identify and design an approach towards plan execution robustness for BDI agents, based upon the proactive modification of plans to avoid anticipated (risk of) activity failure*. To achieve this aim, four individual research objectives were formed, which we discuss and evaluate below.

1 ***To determine knowledge requirements for agents to anticipate where an intended activity risks failure, following exogenous change.***

Our review of information requirements for automated planning (Chapter 5) and plan robustness under uncertainty (Chapter 6) led us to identify a requirement for capability meta-knowledge, to enable agent introspection regarding their intended plan

and activities. We also inferred a requirement for communicating such information, as regarding delegated activity(s), due to our wish to also consider distributed plan execution; this led to modelling of such information as fields within contracts. The resultant supporting architecture, given in Chapter 8, acts to satisfy this objective by defining the knowledge required by and provided to our resultant algorithms – including relating such information to BDI mental states and, for distributed activity, agent *obligations* (discussed in Section 3.3) and dependencies.

The supporting architecture and our maintenance algorithms are interdependent; the former was defined as a realistic provision of information based upon our literature review, but was also modified as our algorithmic design evolved towards the final version presented in this thesis (particularly as our initial local algorithms were expanded to the distributed context, with maintenance policies being employed to provide implicitly synchronized behaviour). Our objective was therefore effectively refined as being *to satisfy knowledge requirements for our designed behaviour*, in recognition of these mutual dependencies. Consequently, these identified knowledge requirements may not apply as a universal definition for proactive plan execution robustness *in general* – i.e. be appropriate for *all* hypothetical proactive approaches.

Finally, we note our maintenance policy concept represents an extension upon our initial objective, having been adopted to improve flexibility based upon observations of existing (non-robustness centric) work such as *CoSAR-TS* (Uszok *et al.* [2004], Tate *et al.* [2004]). Maintenance Policies were not necessary for our objective to define robustness behaviour – such information could be held and defined at implementation time, or incorporated as contract fields for the distributed maintenance case – but supported an informally formed objective to provide a more *practically applicable* design. The achievement of that objective, however, can only be fully evaluated with the benefit of future observation regarding the impact and adoption of our contributions.

2 ***To provide BDI agents with behaviour to anticipate activity failure and avoid resultant intention failure through proactive plan modification.***

This objective was met by Chapter 9, which defined the *maintain* function and it's invocation within the temporal context of BDI reasoning. Following our literature review of the multiplicity of agent robustness approaches, we viewed our behaviour contribution as necessarily part of a larger set of potentially complementary robustness

behaviours; with it's bounds defined through forming assumptions upon which aspects of robustness would be addressed through *other* methods (e.g. to assume belief and communications correctness lay outside our area of concern).

The behavioural specification in Chapter 7 allowed refinement of this second objective as *to provide plan-execution and monitoring, with plan repair-style modifications being performed in response to detection of where activities risked failure*. Our designed behaviour specifically seeks to form appropriate planning problems which can be passed to an (abstracted) planning module, with the generated result (if found) inserted (to modify) the $plan_i$. We adopted a plan repair approach based upon our literature review of plan robustness techniques, and sought to balance the costs of maximizing plan stability against the associated stability benefits for distributed planning (reflecting observations by Fox *et al.* [2006] and Nebel and Koehler [1992] upon experimental and theoretical complexities of plan repair).

Our design's planner agnosticism (Section 9.4.2) may impact practical applicability; our approach requires agents possess either plan generation capacity *or* sufficiently detailed plan libraries. This impacts viability of our approach if such plan generation is overly costly, or anticipation and identification of maintenance plan recipes unrealistic; but it may conversely *aid* viability through providing flexibility to select the most appropriate planning component implementation for the environment, or even on a per-agent basis (discussed in Section 14.4.2). Our generalization of the planner component does abstract part of the behaviour provided by CAMP-BDI; we argue this is in line with our focus upon *agent* behaviour rather than planning techniques, and recognizes that adopting BDI agent reasoning does not mandate a specific planning method[1].

As Section 14.2.3 discusses, our contribution is potentially applicable in *non* BDI agent reasoning approaches, provided supporting architecture and temporal requirements can be met. This means our contribution arguably exceeds the BDI specificity of our original objective. Conversely, it does not necessarily stand our contribution is applicable for *all* BDI framework implementations; we model intentions as goal-plan pairs ($i = goal_i : plan_i$), which will not hold if this information is either not represented or inferable from the contents of the I set. Similarly, our design does assume support

[1] i.e. we regarded our behavioural objective as invocation and performance of plan modification activities using an appropriate planner 'tool', but not the – domain-dependant – planning processes of activity selection.

for our *distributed* approach can be implemented – i.e. that agents form and communicate contractual information, and that support exists for capability advertisement.

3 ***To provide agent team level behaviour that provided such proactive robustness within the context of distributed plan execution.***

Chapter 9 contributed *local* maintenance behaviour (proactive plan modification in anticipation and prevention of failure), as required by our second objective. Our third objective sought to provide equivalent behaviour for execution of *distributed* plans, as the necessity of multi-agent team activity is a typical motivator for a MAS approach. Chapter 10 – drawing from our literature review of mental state concepts for distributed activity, agent robustness, and HTN plan repair techniques – provided this design, where structured messaging was used to drive the autonomous adoption of maintenance responsibility by members of hierarchical agent team.

We defined a further requirement *to provide a decentralized approach* for this objective, recognizing the likely infeasibility of centralized methods in realistic environments. This influenced our supporting architecture, requiring the sharing of capability information both between agents within an activity delegation relationship, and more generally upon those activities which *could* potentially be delegated.

Our design extended Chapter 9, by defining communications requirements and invocation conditions for maintaining both dependant intentions and those motivated as (contractual) obligations. Distributed and non-distributed maintenance also differed in the provision of capability and maintenance policy information by the supporting architecture, but with the source of these knowledge objects kept transparent to the *maintain* function. By using the same representation model was used in both local and distributed cases, with semantic details of the source encapsulated within methods such as *getCapability* (Section 8.2.4) or *getPolicy* (Section 8.3.2), we could therefore reuse *maintain* reasoning without modification.

Our messaging driven process was inspired by work upon exception handling and aggregation (Section 4.3), for the propagation of responsibility – resulting in distributed plan repair behaviour intentionally similar to re-refinement repair of HTN plans. Maintenance policies, specifically with their usage within contracts, provided

synchronization through defining shared maintenance conditions for agents in a (i.e. contractually formed) delegation relationship.

A number of simplifying assumptions were made; specifically to abstract task-allocation and contract formation processes as implementation dependent components, with our work focusing upon use of the resultant *information* (i.e. the dependency or obligation contracts for delegated activities). The distributed maintenance design does have a potential restriction upon applicability to the importance of contracts as information 'carriers'; our approach would – at best[2] – struggle if agents performed *ad-hoc* distributed activity without requiring advance agreement upon delegation. However, we argued this assumption of contract-formation is justified as being a likely requirement to guard against agent resource contention in realistic domains.

4 ***To show our proactive plan modification approach can offer superior robustness over reactive approaches, where environments possess properties befitting our motivation.***

To address this objective, we first more formally defined robustness; establishing that '*the efficacy of our approach is to be measured through goal achievement rate under perturbation; the latter defined as the rate of exogenous change*' (Section 4.1). Our experimental results (Chapter 11 showed CAMP-BDI offered superior robustness,with that advantage increasing with increased perturbation (i.e. greater probability of exogenous change and/or post-failure debilitation).

This identification of superior robustness satisfied our initial objective. We further expanded our evaluation to also consider planning and messaging costs – relevant towards the *practical* applicability of CAMP-BDI; i.e. *to show that our proactive robustness approach did not carry associated excess costs*. We verified that robustness improvements from CAMP-BDI were not accompanied with excessive planning costs, although the messaging costs associated with contract updates were identified as requiring further optimizations (Section 14.4.3). Our messaging cost evaluation did also observe lower per-goal number of contract cancellation and/or formation messages for CAMP-BDI; this suggested a stability benefit, helping validate our reasoning for adopting a plan repair style approach for the distributed maintenance case.

[2]Specifically, external capability information could be employed to reason about potential delegations in the absence of specific contract knowledge – but such information would be of reduced specificity, for reasons discussed earlier in this thesis.

Although we regard our objectives – and consequently our research aim – as being largely achieved, our results are viewed as indicative rather than definitive. Our review of planning and multiagent domains (Chapter 2) described realistic and plausible extensions to provide existing domains with our motivating properties, but also identified a number of issues with existing implementations of such domains. This lead to the design and adoption of the *Cargoworld* for evaluation; use of this custom domain allowed for (and shifted our objective somewhat towards) a deeper evaluation of performance, by facilitating the explicit scaling of properties influencing environmental perturbation. We also discussed the potential applicability of CAMP-BDI in other domains in Chapter 13, highlighting how and where our approach could be suitable.

14.2.2 Relationship and dependencies between CAMP-BDI and BDI

The BDI rational reasoning approach plays an important role in our contribution, by providing an experimentally validated approach towards rational reasoning for our eventual definition of CAMP-BDI and offering immediate practical applicability through BDI's own status as a de-facto standard (Wickler *et al.* [2007]). We adopted BDI in favour of a bespoke agent architecture with integral proactive robustness behaviour, as the latter would require (re)validation of *general* correctness and lose benefits associated with existing use and acceptance of BDI. Use of an existing reasoning architecture also made available a number of validated agent frameworks (such as *Jason*) suitable for experimentation; a further risk of a custom architecture was that experimentation would be evaluating the rational reasoning *in general*, rather than the relative merits of proactive versus reactive plan robustness.

We used the generic BDI reasoning cycle (Rao and Georgeff [1992], Rao and Georgeff [1995]) to describe invocation of *maintain*, including assumptions concerning the relative timing of messaging and activity execution. The *maintain* function employed information within Beliefs (which included the CAMP-BDI supporting architecture) to reason over a specified (to be maintained) intention, but was not concerned with the reasoning for *generating* the B, D or I (or indeed D) mental state sets. Agents situated within realistic environments may considered to *inherently* require mental state components equivalent to BDI – Georgeff *et al.* [1999] argue the "*basic components of a system designed for a dynamic, uncertain world should in-*

clude some representation of Beliefs, Desires, Intentions and Plans, or what has come to be called a BDI agent" – meaning our assumptions, and the base concept of Beliefs and Intentions, are potentially generic enough for CAMP-BDI applicability to extend *beyond* BDI (discussed in Section 14.2.3).

BDI lacks an explicit model of *social ability* (i.e. a characteristic Wooldridge [1999] requires for intelligent agency), although extensions such as *BOID* (Broersen *et al.* [2001a]) and *B-DOING* (Dignum *et al.* [2002]) do model the motivation and constraining role of *obligations* within agent reasoning. Distributed maintenance required specification of assumptions regarding the use of contracts and the temporal context of messaging when designing *distributed* maintenance, as these elements can be viewed as left implicit within generic BDI reasoning. These assumptions do entail that our approach has requirements beyond the minimum of 'generic' BDI reasoning, which implicitly allows ad-hoc delegation and execution (i.e. without advance contract formation or update messaging being required).

We assumed intentions could be modelled as *goal*:*plan* pairs, with this information either directly represented (i.e. as our design employed employed for a simplifying assumption) or inferable from the I set. This attempted to address ambiguity in the specific *meaning* of an intention within existing work, and an additional lack of an explicit representation of *selected goals* within BDI (Section 3.2.1). We argue it is reasonable to assume agents would record, or could infer, relationships between activities (i.e. to identify plan members) and selected desires (goals). This particularly applies for environments – such as our motivating domain – where the risk of activity failure would likely require such information to be available for *reactive* recovery (regardless of CAMP-BDI's requirements).

It is worthwhile to consider how BDI reasoning may evolve in future, and the potential impact upon CAMP-BDI. One criticism of BDI has been a lack of support for learning – arguably stemming from a traditional use of plan libraries for computational efficiency, and which may be addressed through (increasingly viable) use of runtime planning. Our composite capability definition primarily used plan library contents to specify (from selection conditions) capability preconditions and for confidence estimation; Section 8.2.3.2 discusses the extension of composite capabilities to represent the ability achieve goals through runtime planning, noting that manual specification of

preconditions and confidence functions would likely be required. Changes to composite capabilities would potentially impact reasoning over as-yet unselected or unrefined goals; selected goals would likely have associated intended plans, where most activities to be considered by maintenance would likely be atomic and mappable to primitive or external capability knowledge.

Increased runtime planning could lead to BDI rationality focusing upon *goal* determination over activity selection, as the latter can be delegated to automated planning components once a goal is selected. This may lead BDI implementations to further record the reasoning behind goal selection, perhaps through recording QOC-style[3] rationale (MacLean *et al.* [1991], Polyak and Tate [1998]) to support retrospective evaluation of agent decision making. Rationale capture would potentially require CAMP-BDI to record information regarding maintenance decisions (i.e. which tasks were identified, which were selected, and why a given maintenance plan was accepted or rejected). Supporting architecture knowledge could also be recorded *if* desire and intention selection were to be extended to employ capability information.

Research upon agents reasoning behaviour when acting within a multiagent system or team context will be important in enabling the use of BDI within distributed activity contexts. Existing work such as BOID and B-DOING, as noted previously, extend BDI to represent the role of *Obligations* to other agents upon agent motivation and behaviour. *Norms*, represented within B-DOING or Normative Agent Architecture (NoA) (Kollingbaum and Norman [2003]), similarly present societal constraints or requirements upon activity – for example, to prohibit certain activities or the establishment of certain states. In NoA, norms include the concept of *obligation* (Kollingbaum *et al.* [2006]) – i.e. define a requirement to perform some activity or ensure some state holds – which can be regarded as similar to the Obligation concept of B-DOING and BOID[4].

Whilst CAMP-BDI's core *maintain* function is intended to be executed following

[3]Questions, Options, Criteria records decision rationale through in three elements; *Questions* or issues to be addressed, *Options* for answering those questions, and *Criteria* for selecting between Options.

[4]BOID treats Obligations as implicit Norms. B-DOING distinguishes Norms from Obligations, with former being stable, abstract and inherent to operating within the agent group or society, and the latter entered into by choice as a result of agent activity. We note in all these cases Obligations refer to external motivational requirements for activity; this differs slightly from our supporting architecture, which models the specific contract contents as well as the existence of agreements between agents.

I, and ergo arguably insulated from changes to the *generation* of intentions, such constraints would need to be respected; we opted to assume the maintenance planning component (Section 9.4.2) would include any reasoning required to account for system norms. There remains an additional possibility that norms, particularly if modified during runtime, could impact the specification of maintenance planning problems (i.e. by introducing constraints or requirements on top of the planning problem generated using capability precondition or effects specification). Consequently, work may be required to adapt CAMP-BDI for use in particular norm-aware BDI frameworks depending upon the norm semantics and any reasoning specifics relevant to maintenance planning.

Finally, we can suggest the potential extension of BDI reasoning to consider *uncertainty* over beliefs – i.e. associating state atoms in *B* with probability values. This would recognize the *B* mental component as *believed* rather than certain information, and that the agent may have sensory or otherwise limitations impacting the certainty of environmental knowledge (Section 14.4.4 also suggests similar potential extension of CAMP-BDI to use state probability information, to improve estimation of future activities' execution context). If so, work would be required to integrate such state probability information within CAMP-BDI maintenance reasoning. Whilst an obvious, intuitive solution is to determinize *B* for use by *maintain* (employing some minimal probability threshold for 'true' states), the significance and relevance of given state atoms may differ in the terms of their impact upon robustness. Whilst a straight threshold-based determinization of *B* remains intuitively viable, future investigation could better consider the relative importance of state atoms using capability knowledge (i.e. with reference to preconditions, and states contributing to confidence) during determinization – i.e. to recognize which atoms are particularly irrelevant in the context of successful execution, and which would have severe enough consequences to mandate a greater degree of caution and ergo a lower 'truth' probability threshold.

14.2.3 Requirements and Potential Generalization

The requirements of our approach can be characterized in terms of the information required by the *maintain* function, the temporal context of invocation, and infrastructure requirements for facilitating distributed maintenance:

- In **Information** terms, we defined $maintain(i) \rightarrow i'$; where i is $\{goal_i, plan_i\}$

(with $goal_i$ and $plan_i$ treated as respectively immutable and mutable). This assumes implicit access to agent beliefs B (i.e. $B = \{W, C, MP, Ob, Dp\}$, giving world state W, and with the supporting architecture of Capabilities C, Maintenance Policies MP, Obligation contracts Ob and Dependency contracts Dp); meaning we can more generally define *maintain* as:

$$maintain(goal_i, plan_i, W, C, MP, Ob, Dp) \rightarrow plan_i'.$$

- In **Temporal** terms, *maintain* must be executed *after* (in response to, or to propagate) receipt of post-maintenance messaging from obligants and *before* performing messaging dependants with obligation changes, due to maintenance or by propagated (received) sub-obligant changes. These are not required to exist within the agent reasoning approach, but rather require an ability to modify agent reasoning to *support* these messaging requirements.

- In **Infrastructure** terms, CAMP-BDI agents require provision of services to share (advertise) the Capability and Maintenance Policy information required for distributed maintenance. We also assume the delegation of activities leads to acyclic structures, including the decomposition of delegated tasks by obligants, to void the risk of infinite maintenance 'loops' from cyclical relationships[5].

Whilst we initially sought to investigate robustness within a BDI context, our approach may be suitable as a robustness approach for agents employing deterministic plans *in general*. This is particularly as our algorithms do not *require* BDI rational behaviour, but rather reason over the *outcome* of it – i.e. it is not important *how* beliefs are updated, or goals and plans selected, but rather that they are available for use as parameters to *maintain*. Whilst we extend the BDI reasoning cycle in our design, this primarily serves to provide a temporal context and conditions for invoking *maintain* invocation with regard to message receipt (i.e. defining contract messaging assumptions for the distributed case), plan selection (and execution).

Although our design refers to BDI mental state concepts of *Belief* and *Intention*, such concepts need not be restricted to BDI rationality – as our generalization of *maintain*'s arguments suggests, our pre-emptive plan modification logic can be applied in any agent which employs a plan, associates that plan with a goal or task, and has some

[5]Although our concern is with the decomposition of *tasks*; agents may hold indirect self-dependencies, provided the distributed plan eventually resolves to an acyclic graph of activities.

form of holding world state knowledge to allow implementation of our supporting architecture (including infrastructure for capability and maintenance policy advertisement). This raises the possibility of applying our approach within *other* plan-executing agent approaches (including, as we do not strictly require *autonomous* goal and plan formation, non-intelligent ones); such generalization represents an important area for future investigation and for enhancing the applicability of our contribution.

14.3 Related Work

In our literature review, we did not identify any robustness approaches specifically equivalent to CAMP-BDI; i.e. those for proactively avoiding the failure of activities within intended plans, or for doing so within a distributed team context. We did, however, identify alternate approaches towards plan robustness, or which are proactive but concerned with different aspects of agent or MAS fault tolerance.

Replication or redundancy approaches can prevent failure from agent loss, but do not target behavioural robustness (i.e. are responsible for whether BDI agent instances *exist*, but not their behavioural correctness). Our capability model may be useful for targeted replication (e.g. de Luna Almeida *et al.* [2007]), as it can allow consideration of agent capabilities as part of determining relative criticality. *Role-filling* approaches, such as OMACS by DeLoach [2009] or that by Preisler and Renz [2012] can use utility functions similar to capability confidence estimation, but are concerned with agent role assignment as part of maintaining a fixed, predefined system organization rather than providing robust behaviour *by* agents. While role-filling approaches are concerned with maintaining predefined organizations, CAMP-BDI effectively allows *modification* of the meta-organizations arising from task delegation as a consequence of maintenance changes to plans. If a fixed organizational structure *does* exist, this can be represented in CAMP-BDI by appropriate constraints upon capability advertisement, as capability awareness controls the delegation relationships an agent is able to (knows it can) form.

Sentinel monitoring, *Exception* propagation and *Failure Diagnosis* approaches (surveyed in Chapter 4) are considered outside our plan execution focus and not directly impacting (or conflicting with) CAMP-BDI. Sentinels (Hägg [1997]) and failure diagnosis approaches (Kaminka and Tambe [1998], Roos and Witteveen [2005]), observe

agent behaviour to detect and (reactively) reconcile belief inconsistencies; this would only impact CAMP-BDI in terms of taking action to ensure more accurate agent beliefs (i.e. as used for reasoning). The approach used by our design for adoption of responsibility during distributed maintenance mimics exception propagation (Klein and Dellarocas [1999], Souchon *et al.* [2004]), in terms of escalation and aggregation of maintenance responsibility up a hierarchical team.

One approach for improving plan robustness is to form plans which account for environmental uncertainty. Conformant planning (Smith and Weld [1998]) attempts to form plans guaranteed to succeed *regardless of world state*, covering uncertainty over outcome or world state; MDP (Markov Decision Process) are solved to form a policy that defines the optimal (maximum reward) activity for any given world state. However, both are often intractable for realistically complex environments (such as shown in Schut *et al.* [2002]), especially when uncertainty includes unpredictable exogenous change as well as probabilistic activity effects (especially for POMDPS, which introduce additional levels of complexity by reasoning over *possible* observations). Domain abstraction can improve tractability for both conformant planning (Palacios and Geffner [2006]) and MDP solution (Boutilier and Dearden [1994]), but this reduced precision can reduce optimality – and introduce a risk of failure as a result. Capability estimation can be equated to definition of transition probabilities in MDP models, although the former only requires an indicative, scalar estimation of quality rather than an exact probability.

Continual planning defers planning decisions until during execution, assuming more accurate knowledge would be held at that point than at planning time (desJardins *et al.* [1999]). Where this involves decomposition of abstract activities at execution, CAMP-BDI allows maintenance through composite capabilities, which allow determination of whether an undecomposed activity has a selectable plan *and* can estimate confidence by anticipating the plan most likely to be selected. Where planning can include sensing activities, this can be supported by definition of knowledge-requirement preconditions and knowledge-attainment effects (similar to Petrick and Bacchus [2002] or Brenner and Nebel [2009]) within corresponding capabilities.

Conditional or Contingent planning handles uncertainty using conditional branching. This can risk exponential plan growth in realistic domains due to their complexity

(Albore *et al.* [2007]); making some form of intelligent branch placement necessary – for example, Dearden *et al.* [2002] probabilistically identify likely failure points. It is unlikely contingent plans can entirely avoid failure in realistic domains, particularly where exogenous changes can occur at any point during plan execution (and ergo require handling branches at *any* activity point). CAMP-BDI *can* maintain conditional plans where linearized, inferring branches likely to be followed using current beliefs.

CAMP-BDI's approach to *handling* threats adopts a plan repair style approach, due to the stability benefits offered over replanning (Fox *et al.* [2006]). Our literature review found only limited current work on *multiagent* plan repair. One approach, by Boella and Damiano [2002], defined a *reactive* plan repair algorithm aimed at BDI agents. Like CAMP-BDI, their approach was agent-centric and employed a utility function (equatable to our *confidence* function) to determine whether repair was required; however, unlike CAMP-BDI, their approach did not extend to the multiagent case, or define *how* utility function information was provided for agent reasoning.

Komenda *et al.* [2012] compared multiple approaches for repairing a multiagent plan following activity failure: *back on track repair* inserted activities to re-establish 'missing' post-effects; *lazy repair* inserted a new suffix to the end of the plan upon each failure to achieve missing effects, to be executed after any remaining viable activities in the original plan; *repeated lazy repair* avoided concatenation of lazy-repair suffixes following multiple failures by (re)forming a suffix after every failure, removing any added by prior repair. Both *back on track* repair and CAMP-BDI preconditions maintenance may be triggered by loss of causal links and entail addition of repair plans as prefixes – however, as CAMP-BDI is a *proactive* approach it may insert 'prefix' plans as suffixes to the preceding activity. CAMP-BDI effects maintenance bears similarities to repeated lazy repair in terms of the scope of plan changes, as this maintenance type *may* replace the maintained activity and the remainder of the (sub)plan containing it.

We employed an HTN re-refinement repair approach towards distributed maintenance, regarding the hierarchical decomposition performed by an agent team as structurally equivalent to HTN task refinement. However, CAMP-BDI performs such behaviour in a proactive context rather than in response to activity failure – requiring CAMP-BDI agents hold additional knowledge and algorithms to anticipate possible failure, rather than respond to post-hoc detection. A key distinction of CAMP-BDI

is our focus upon knowledge requirements for enabling proactive use of plan repair, rather than defining particular semantics for *forming* repair plans.

Braubach *et al.* [2005] define two types of goals driving agent proactivity – those to achieve a state, and those to maintain it while state or temporal conditions hold. Duff *et al.* [2006] further distinguish proactive and reactive types of maintenance goal. Reactive maintenance goals require re-establishment of violated 'protected' states when violated – driving adoption of (re)achievement goals (where resultant intentions could be maintained by CAMP-BDI) – whilst proactive goals constrain intention formation (both goals and plans) to prevent violation of protected states. CAMP-BDI arguably produces a similar outcome to proactive maintenance goals, as the maintenance agenda formation algorithm effectively acts to preserve precondition required states. Effects maintenance proactively responds to state violations – similar to a proactive maintenance goal to ensure the *inverse* or absence of confidence lowering states – but the consequent behaviour can allow insertion of an alternate equivalent activity sequence rather than *requiring* effective removal of confidence lowering states. We also assume planning mechanisms used by CAMP-BDI respect proactive maintenance goals, in the same manner as forming (or selecting) intended plans.

Hindriks and van Riemsdijk [2008] used (limited) lookahead to ensure proactive maintenance goals are respected; a goal plan tree was formed and used to anticipate the future effects of adopted intentions, to identify potential violations of maintenance goal states. Such violation was suggested as best addressed by goal relaxation (which, we note, may not be always be a viable option) – they viewed plans as pre-defined and immutable, whilst CAMP-BDI treats intended plans as mutable but *goals* met by them as effectively immutable. Duff *et al.* [2006] suggest another predictive approach, again using a goal-plan tree to filter goal adoption and avoid plans with effects that would violate proactive maintenance goals. CAMP-BDI varies from both these approaches by recognizing and reacting to the effects of exogenous change, rather than assuming state violations only arise from the selection of new plans or goals.

CAMP-BDI agenda formation behaviour serves a similar purpose to *Plan Execution Monitoring* by responding to activity failure or unexpected effects. PEM approaches, such as SIPE (Wilkins [1983]) or IPEM (Ambros-Ingerson and Steel [1988]), detect and react to divergence between expected (at plan formation) and ac-

tual execution context, such as by invoking replanning or plan repair – this arguably blurs boundaries between proactive and reactive behaviour where divergence can arise from exogenous change rather than activity failure. FF-Replan (Yoon *et al.* [2007]), for example, determinizes probabilistic domains to take advantage of classical planning optimizations – using PEM to perform replanning if *actual* effects differ from those stated in the *determinized* (single, most likely outcome) domain.

Like FF-Replan, CAMP-BDI responds to unexpected outcomes, but attempts to minimize resultant changes rather than entirely replan; our focus is also more upon exogenous change impacting future activities, rather than occurrence of known but less-likely (by determinization) activity effects. Similarities also exist between CAMP-BDI and the use of *protection monitors* as repair triggers in O-Plan (Drabble *et al.* [1997]), as both employ causal link information to determine whether future activity risks violated preconditions. A more general difference with PEM behaviour is that our approach is agent orientated – we directly consider provision and communication of (planning operator equivalent) capability knowledge, including between members of a MAS. We also use confidence estimation to anticipate *risk* of failure, whilst PEM approaches typically consider deterministic operator models; PEM approaches also respond to existing (i.e. occurred) failure, whilst CAMP-BDI examines plans for threats on every reasoning cycle to identify the impact of belief changes from any source.

Foss *et al.* [2007] describe an approach which, like CAMP-BDI, aims to avoid irrecoverable failure. Their approach is inspired by FF-Replan, determinizing a Probabilistic PDDL specification, but employing pseudo-probabilistic planning similar to PACPlan (Jiménez *et al.* [2006a]). Their planner can both insert precautionary repair steps, and perform limited conformant or contingent planning where a potential outcome would cause irrecoverable failure. Whilst sharing our motivation, they only consider effects defined in PPDDL operator specifications; CAMP-BDI does not perform the same type of precautionary repair or *advance* contingency planning, as it only modifies plans in response to known *current* threats, it *can* account for exogenous change as well as unexpected (low probability, known *or* unknown) effects.

14.4 Further Work

CAMP-BDI presents an initial approach; our design and experimentation suggests a number of opportunities for further research and development, which we will detail here to conclude this thesis. Future work to expand our experimentation to consider different domains or domain properties is also desirable, as is investigation into the use of CAMP-BDI to complement reactive robustness methods.

14.4.1 Asynchronous Maintenance

Our capability model currently defines preconditions and effects for activities, similar to within a STRIPS (Fikes and Nilsson [1971]) operator. Although we have viewed activities in state transition terms, they may involve transient or fluent states that persist for some period *during* execution. There may be benefits from modelling such durative states within internal and external (whether advertised or within contract *EC* fields) capabilities, such as to support asynchronous performance of maintenance alongside standard BDI reasoning.

Planning is likely to represent the primary temporal cost in maintenance, which risks imposing a delay between completion of an activity and execution of it's successor. If activities typically take longer to execute than maintenance planning takes to complete, maintenance could be performed *in parallel* – under the assumption agents are *logically* idle while executing a non-instant activity. Durative state knowledge could facilitate maintenance during this 'idle' period by allowing estimation of the post-execution state following the current activity – meaning the current $plan_i$ could be maintained whilst that activity is executing (under an assumption of success). This would require the CAMP-BDI reasoning cycle employ multi-threaded reasoning – i.e. one thread forming desires, intentions and plans whilst another continuously evaluates the currently executing $plan_i$. Whilst reactive approaches can only invoke planning operations after execution has completed (and failed) and post-failure state is known, this approach towards maintenance could mitigate planning costs through parallelization.

Multi-threaded maintenance behaviour could be further extended to perform 'speculative' reasoning; using current beliefs, capability knowledge and potential exogenous change types to identify alternative *possible* future execution contexts for planned ac-

tivities. Multiple threads of maintenance could consider these potential beliefs, pre-emptively forming maintenance plans that could be cached for future use, generalized to form reactive plan libraries, or inserted as conditional (contingency) plans (similar to Foss *et al.* [2007]). This could, however, risk high computational cost and would require balancing against resources available to that agent – possibly to the extent of only being viable for certain logical 'controller' type agents.

14.4.2 Heterogeneous Planning

CAMP-BDI algorithms were designed as planner agnostic; this allows for the possibility of *heterogenous* planning. For example, physical agents with reduced computational resources or tighter time constraints could use HTN approaches or libraries for maintenance planning – trading flexibility for greater reactive speed. Conversely, agents with greater computational resource – such as broker or controlling logical agents – could employ more flexible classical planning, or even pseudo-probabilistic methods such as used by *CAMP-BDI.Quality* (Chapter 11).

Planner heterogeneity could also be employed *within* an individual CAMP-BDI agent, using different types of planning depending upon the particular activity under maintenance, the associated $goal_i$, or even the current extent of maintenance scope (i.e. increasing computational expenditure where larger parts of the plan would be impacted). Maintenance plans, once formed, could also be generalized and stored (providing generic types can be defined for maintenance tasks, so as to be associated with generic plans) – allowing more reactive behaviour for frequent and common case issues. Plan and task type generalization could also enable the *sharing* of maintenance plans across agents, effectively defining standard operating procedures.

14.4.3 Communications Optimizations

The primary drawback observed for CAMP-BDI was the communications cost of frequent contract update messages. Potential optimization may consider two factors; reducing the volume of messages sent, and reducing individual cost (size) for messages. As the latter is specific to the particular agent framework implementation, we will focus on the former. However, one possible size optimization is to compress communicated information to only contain the information required by recipients to *locally* modify contracts. This would not effect worst case complexity – where all contract fields are

updated – but could improve the average case.

If multiagent planning employs a specific private/public action approach (Brafman and Domshlak [2008]), frequency could be reduced by only considering whether *public* atoms have changed – i.e. if changes to the Casual Link (*CL*) or External Capability (*EC*) fields (Section 8.4) only concern private atoms, these can be identified as only meaningful to the obligant and that any communication to update the dependant would be unnecessary. The set of private/public atoms information could potentially be inferred using external capabilities – although this must consider that restrictions on advertisement might prevent agents identifying all public atoms, due to lacking total awareness of other agent capabilities. Again, this would not improve worst case cost, but may improve the general case (but requiring use of a private/public model).

The frequency of dependency contract updates could also be linked to immediacy – or to even omit updates entirely if execution of the dependency is sufficiently distant. This would trade off the accuracy of information available to obligants against the cost of communication. Such an approach might also require modification to the agenda formation algorithm to consider increased uncertainty stemming from less frequent contract updates for distant dependencies. It may also be advantageous to form a method to determine if contract changes are significant – and therefore only communicate updates where changes are likely to *require* maintenance changes by recipients. This would involve reconsideration of the information shared between agents, with particular regard to establishing the roles delegated activities play in providing causal links or establishing goal states within the dependent plan.

14.4.4 Execution Context Prediction

CAMP-BDI predicts the future execution context of an activity by combining current beliefs with the (capability defined) ordered effects of preceding plan activities. Although exogenous events are unpredictable, we assume they are not *chaotic*; i.e. that they may be inferred, or associated with a set probability of occurring at *some* point in the future. Investigation of more accurate predictive approaches could help avoid both false positive and false negative maintenance task generation – improving failure avoidance and reducing the cost of unnecessary planning or change.

One option is to associate probabilities with predicted state atoms, denoting how likely they are to hold. For example, a *flooded* road will gradually dry out if not currently being rained upon; conversely, a rained-upon road would be expected to become slippery and flooded for future activities. Where a particular facet of the world is represented by multiple atoms (e.g. *flooded*/*slippery*/*dry*, or *dangerZone* / ¬*dangerZone*), each possibility would have an associated probability – with maintenance employing the *most likely* beliefs for an activities execution context. This would entail additional temporal modelling and domain knowledge to perform probabilistic predictions and inference.

Improved execution context prediction could also support *preventative* activity. Where some negative (confidence lowering) state exceeds a certain probability threshold (potentially defined in maintenance policies), agents could employ behaviour to insert pre-emptive mitigatory activities – such as a *Truck* loading a spare tyre if there is a sufficiently high likelihood of a flat tyre *at some point* in the future. Adopting a state probability method for triggering such behaviour would allow balancing the costs of potentially unnecessary preventative activity, against the benefits where those activities *are* needed to avoid failure. This would raise further issues for consideration, such as accounting for uncertainty when predicting future states – including appropriate scaling with increasing time – and consideration of the risk of combinatorial blow-up from probabilistic reasoning.

Appendices

Appendix A

Cargoworld Simulator Screenshots

This appendix presents screenshots of our Cargoworld simulator, taken during experimentation, to illustrate the differences in initial world state for both geographies. The same seed values were used for exogenous event generation.

Yellow circles represent junctions, interconnected by tarmac (gray if dry, light blue if slippery, and dark blue if flooded) or mud (light brown if dry, brown if slippery, dark brown if flooded) roads; blocked roads are coloured red and toxic roads (not shown here) as purple. The airplane symbols in yellow boxes indicate airports; red polygons denote dangerzones, with junction circles and airplane symbols similarly coloured red. Finally, vehicles are shown as green circles, labelled with their identifying names; as the simulation has not yet started in these screenshots, all vehicles are in their initial starting locations at junctions and no cargo objects or requests have been generated.

A.1 World A

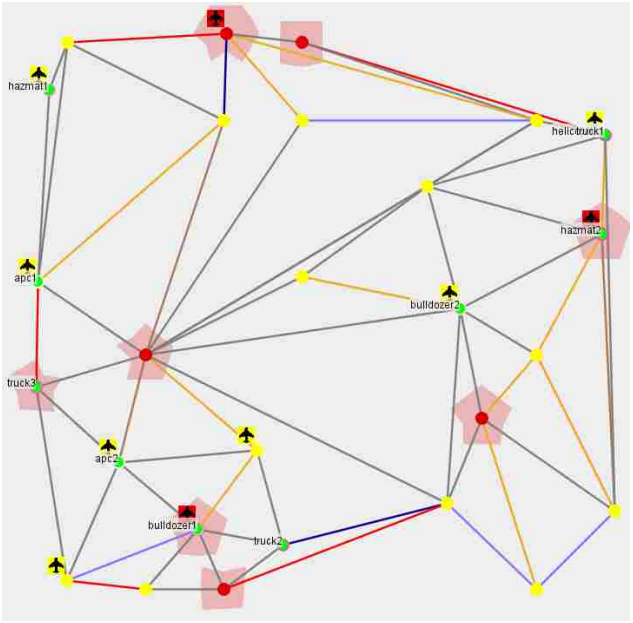


Figure A.1: Screenshot of Cargoworld simulator showing the initial state of **World A** for $n_{exo=1}$.

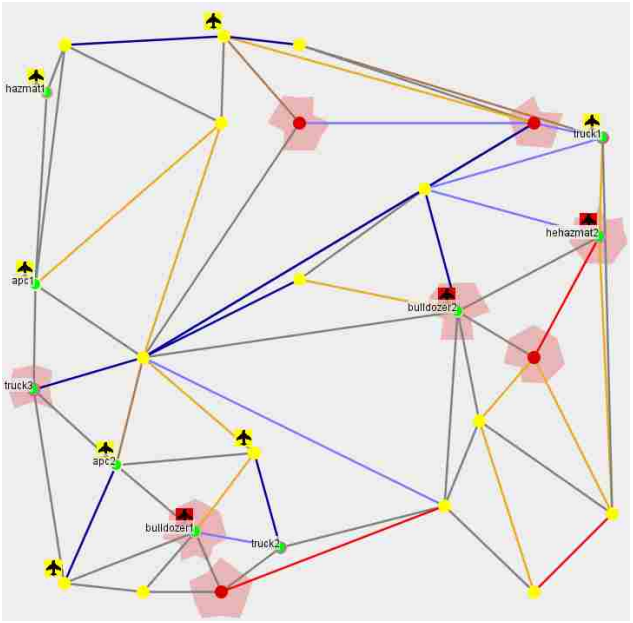


Figure A.2: Initial state of **World A** for $n_{exo=2}$.

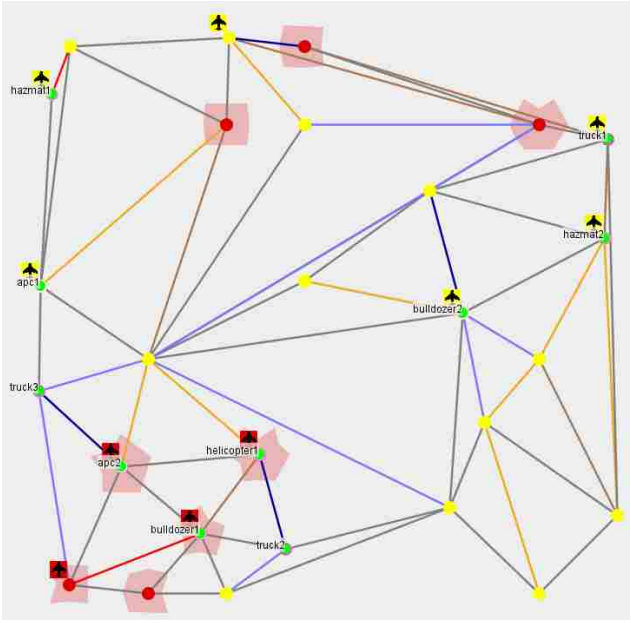


Figure A.3: Initial state of **World A** for $n_{exo=3}$.

A.2 World B

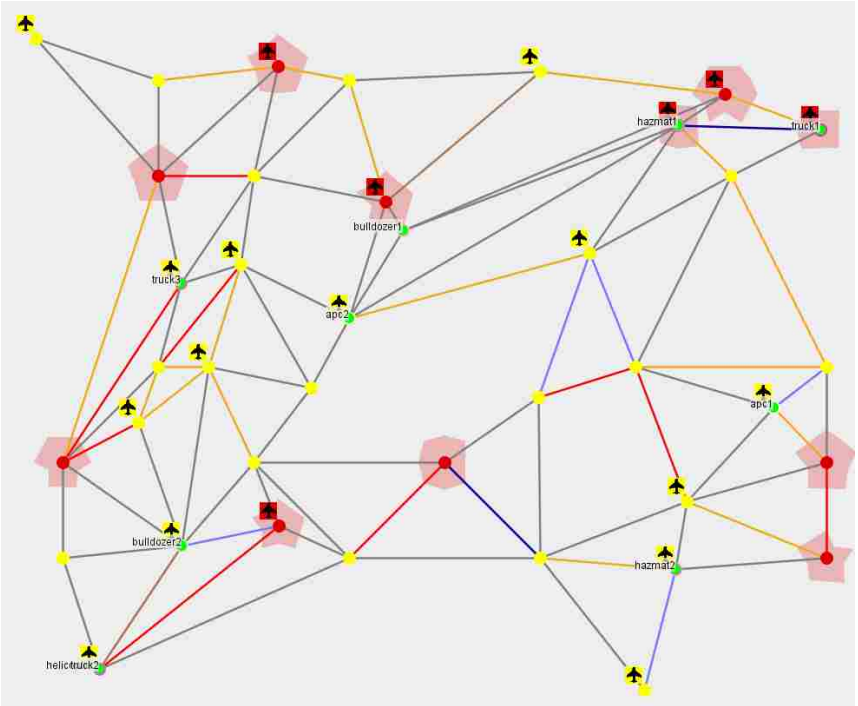


Figure A.4: Screenshot of Cargoworld simulator showing the initial state of **World B** for $n_{exo=1}$.

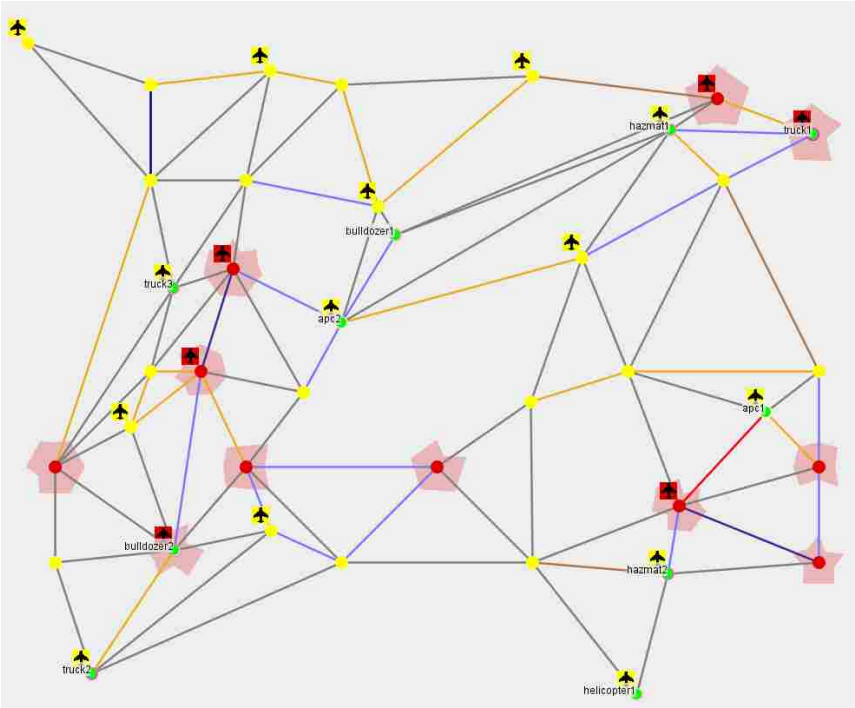


Figure A.5: Initial state of **World B** for $n_{exo=2}$.

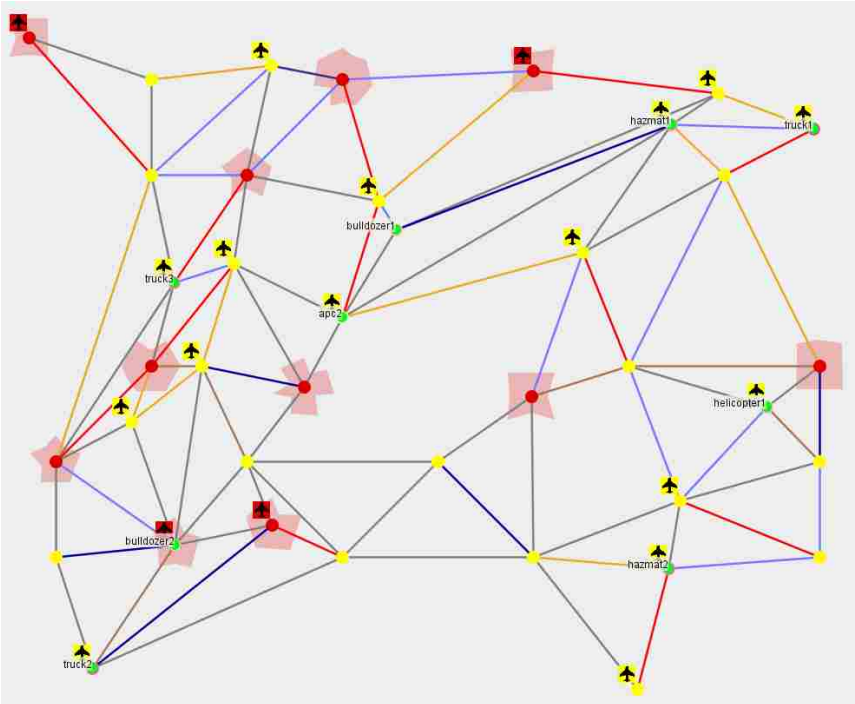


Figure A.6: Initial state of **World B** for $n_{exo=3}$.

Appendix B

Experimental Results

This appendix provides the detailed numerical results, as used to form the result graphs present in Chapter 12. We provide further comparative values (and p values; we deemed $p < 0.05$ as being statistically significant) employed for evaluation of performance (activity, planning and messaging related) metrics; these detail differences *between* approaches for every experimental configuration (World, n_{exo} and n_{risk} value). Finally, we also provide tables detailing the specific changes (or otherwise) in CAMP-BDI performance for each performance metrics as n_{risk} values were progressively increased for experimentation.

B.1 Average Goal Achievement

B.1.1 World A – Average Goal Achievement

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	57 (6.765)	41.6 (11.542)	29.08 (15.293)	30.8 (12.691)
Replanning	98.15 (1.376)	89.02 (3.708)	73.77 (6.847)	60.82 (9.058)
Continual	98.37 (1.57)	91.05 (2.883)	87.65 (4.254)	83.18 (4.928)
CAMP-BDI.Spd	98.25 (3.952)	98.57 (1.309)	98.73 (1.25)	98.6 (1.332)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	31.9 (6.461)	24.9 (7.487)	21.63 (8.744)	18.97 (5.828)
Replanning	98 (1.761)	77.87 (4.349)	53.9 (7.926)	37.37 (8.67)
Continual	97.45 (2.003)	83.18 (5.038)	72.2 (7.328)	61.52 (8.945)
CAMP-BDI.Spd	97.55 (2.132)	97.62 (1.518)	97.85 (1.579)	98.08 (1.282)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	24.033 (5.003)	16.4 (6.481)	13.17 (6.13)	12.67 (5.641)
Replanning	97.9 (1.446)	67.98 (6.217)	41.78 (6.844)	26.12 (6.086)
Continual	97.27 (1.914)	78.83 (4.36)	58.53 (7.288)	47.98 (6.305)
CAMP-BDI.Spd	91 (4.651)	90.95 (3.304)	92.18 (3.447)	92.98 (3.212)
CAMP-BDI.Qty	90.97 (3.924)	91.38 (4.115)	90.83 (3.6)	91.75 (3.118)

Figure B.1: Average goal achievement (%) for approaches in World A for every n_{exo} and n_{risk} combination, with standard deviation in brackets. Each table corresponds to an n_{exo} configuration; each cell defines the percentage of goals met for that (row-defined) approach, under the (column-defined) level of n_{risk}.

B.1.2 World B – Average Goal Achievement

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	32.267 (5.332)	22.82 (9.509)	22.72 (8.798)	21.5 (6.652)
Replanning	96.5 (2.195)	76.57 (5.497)	54.78 (7.28)	38.93 (8.205)
Continual	95.68 (2.52)	85.65 (5.108)	78.87 (4.602)	71.32 (6.474)
CAMP-BDI.Spd	95.517 (2.029)	95.83 (1.951)	95.52 (2.164)	95.9 (1.578)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	18.383 (4.491)	14.417 (4.076)	12.57 (5.172)	12.63 (4.266)
Replanning	95.17 (2.192)	58.47 (6.977)	33.57 (5.655)	22.35 (4.857)
Continual	94.7 (2.452)	72.55 (4.068)	54.37 (5.562)	45.92 (6.977)
CAMP-BDI.Spd	92.62 (3.656)	93 (2.387)	93.52 (2.878)	92.95 (2.872)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	11.167 (3.309)	8.9 (3.375)	8.8 (3.572)	8.97 (3.381)
Replanning	93.83 (3.431)	45.22 (6.248)	27.78 (5.067)	15.25 (4.085)
Continual	94.3 (3.303)	57.07 (6.969)	38.35 (5.913)	29.52 (5.832)
CAMP-BDI.Spd	65.47 (6.349)	66.48 (6.328)	66.85 (6.18)	65.62 (6.998)
CAMP-BDI.Qty	66.92 (6.905)	67.82 (7.484)	66.47 (7.338)	65.77 (8.16)

Figure B.2: Average goal achievement (%) for approaches in World B for all n_{exo} and n_{risk} configurations, with standard deviation in brackets.

B.2 Average Activity Success Rate

B.2.1 World A – Average Activity Success Rate

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	92.872 (1.524)	91.3 (3.269)	89.15 (4.766)	89.51 (5.924)
Replanning	93.77 (1.098)	93.19 (1.165)	90.99 (1.576)	89.54 (2.148)
Continual	96.51 (1.149)	96.47 (0.862)	96.5 (0.984)	96.06 (1.028)
CAMP-BDI.Spd	99.9 (0.117)	99.9 (0.093)	99.91 (0.083)	99.91 (0.099)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	88.963 (1.821)	85.17 (4.418)	83.1 (5.962)	85.68 (4.405)
Replanning	89.66 (1.326)	88.49 (1.29)	85.49 (2.061)	82.94 (3.169)
Continual	93.89 (1.051)	93.59 (0.988)	92.85 (1.476)	92.16 (1.715)
CAMP-BDI.Spd	99.86 (0.134)	99.86 (0.09)	99.88 (0.094)	99.89 (0.08)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	86.09 (2.212)	80.62 (5.809)	78.42 (5.745)	80.84 (6.683)
Replanning	86.67 (1.831)	84.84 (1.968)	81.68 (2.194)	79.29 (2.771)
Continual	91.16 (1.588)	90.84 (1.174)	89.53 (1.658)	89.21 (1.26)
CAMP-BDI.Spd	99.6 (0.249)	99.59 (0.157)	99.65 (0.166)	99.69 (0.161)
CAMP-BDI.Qty	99.57 (0.211)	99.59 (0.203)	99.57 (0.175)	99.62 (0.138)

Figure B.3: Average activity success rate (%) for approaches in World A for every n_{exo} and n_{risk} combination, with standard deviation in brackets. Each table corresponds to an n_{exo} configuration; each cell defines the percentage of successful activity executions for that (row-defined) approach, under the (column-defined) level of n_{risk}.

B.2.2 World B – Average Activity Success Rate

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	91.101 (1.563)	87.91 (5.133)	87.96 (4.695)	88.44 (3.248)
Replanning	92.59 (1.163)	90.72 (1.029)	88.42 (1.917)	86.78 (2.06)
Continual	96.56 (0.83)	96.19 (0.842)	95.74 (0.751)	95.69 (0.857)
CAMP-BDI.Spd	99.781 (0.106)	99.8 (0.1)	99.76 (0.092)	99.81 (0.383)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	86.874 (2.026)	83.457 (3.99)	81.46 (5.211)	83.27 (4.67)
Replanning	88.37 (1.065)	85.31 (1.351)	82.85 (1.928)	80.65 (2.383)
Continual	93.42 (0.765)	92.92 (0.94)	91.38 (1.048)	91.5 (1.478)
CAMP-BDI.Spd	99.76 (0.117)	99.77 (0.097)	99.79 (0.098)	99.77 (0.096)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	82.814 (2.707)	78.55 (4.67)	77.91 (5.269)	78.36 (6.216)
Replanning	85.07 (1.08)	81.7 (7.401)	78.63 (2.186)	77.49 (2.267)
Continual	90.82 (1.303)	88.89 (1.619)	87.92 (1.433)	89.21 (1.658)
CAMP-BDI.Spd	99.23 (0.207)	99.29 (0.185)	99.32 (0.165)	99.33 (0.162)
CAMP-BDI.Qty	99.15 (0.23)	99.16 (0.361)	99.17 (0.247)	99.18 (0.186)

Figure B.4: Average activity success rate (%) for approaches in World B for every n_{exo} and n_{risk} combination, with standard deviation in brackets.

B.2.3 World A – Differences between CAMP-BDI.Speed and other Approaches

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+7.03 ($1.83x10^{-41}$)	+8.604 ($3.27x10^{-28}$)	+9.246 ($1.96x10^{-24}$)	+10.4 ($1.26x10^{-19}$)
Replanning	+6.123 ($2.91x10^{-46}$)	+6.714 ($3.82x10^{-47}$)	+8.915 ($2.15x10^{-46}$)	+10.367 ($1.22x10^{-42}$)
Continual Replanning	+3.396 ($5.59x10^{-31}$)	+3.435 ($5.8x10^{-38}$)	3.415 ($2.75x10^{-34}$)	+3.849 ($8.71x10^{-36}$)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+10.893 ($4.23x10^{-48}$)	+14.699 ($1.46x10^{-33}$)	+16.777 ($9.55x10^{-30}$)	+14.211 ($6.88x10^{-33}$)
Replanning	+10.194 ($4.58x10^{-54}$)	+11.376 ($5.57x10^{-58}$)	+14.392 ($8.65x10^{-52}$)	+16.949 ($4.69x10^{-45}$)
Continual Replanning	+5.963 ($2.13x10^{-46}$)	+6.275 ($3.41x10^{-49}$)	+7.028 ($4.55x10^{-42}$)	+7.734 ($7.01x10^{-41}$)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+13.512 ($9.06x10^{-48}$)	+18.968 ($3.71x10^{-33}$)	+21.232 ($3.95x10^{-36}$)	+18.848 ($1.04x10^{-29}$)
Replanning	+12.937 ($1.51x10^{-51}$)	+14.752 ($1.73x10^{-53}$)	+17.975 ($3.99x10^{-56}$)	+20.397 ($4.41x10^{-53}$)
Continual Replanning	+8.444 ($2.32x10^{-45}$)	+8.752 ($1.79x10^{-53}$)	+10.122 ($1.27x10^{-47}$)	+10.472 ($5.6x10^{-56}$)
CAMP-BDI Quality	+0.035 (0.376)	+0.002 (0.954)	+0.079 (0.012)	+0.066 (0.013)

Figure B.5: Differences (p in brackets) in average activity success rate (% of activities that completed successfully) between *CAMP-BDI.Speed* and other approaches in World A. Each table corresponds to an n_{exo} configuration, cells give the difference in percentage activity success between CAMP-BDI and the (row-defined) other mitigation approach under that (column defined) level of n_{risk}. Positive values show a greater percentage of activities successfully executed for *CAMP-BDI.Speed* agents than those using any other approach.

B.2.4 World B – Differences between CAMP-BDI.Speed and other Approaches

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+7.193 ($5.81x10^{-46}$)	+9.077 ($2.089x10^{-25}$)	+11.37 ($3.58x10^{-27}$)	+13.038 ($1.02x10^{-34}$)
Replanning	+7.193 (1.66^{-48})	+9.077 ($8.32x10^{-58}$)	+11.37 ($8.64x10^{-48}$)	+13.0378 ($1.95x10^{-49}$)
Continual Replanning	+3.221 ($8.78x10^{-37}$)	+3.615 ($3.89x10^{-39}$)	+4.052 ($6.71x10^{-45}$)	+4.124 ($2.15x10^{-42}$)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+8.68 ($1.64x10^{-49}$)	+11.894 ($1.05x10^{-38}$)	+11.832 ($6.7x10^{-35}$)	+11.37 ($5.81x10^{-35}$)
Replanning	+11.398 ($2.12x10^{-62}$)	+14.454 ($1.12x10^{-62}$)	+16.935 ($1.31x10^{-57}$)	+19.125 ($4.97x10^{-55}$)
Continual Replanning	+6.341 ($2.21x10^{-55}$)	+6.844 ($2.9x10^{-53}$)	+8.404 ($3.06x10^{-55}$)	+8.27 ($1.35x10^{-52}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+12.889 ($1.11x10^{-47}$)	+16.311 ($6.8x10^{-40}$)	+18.329 ($7.63x10^{-38}$)	+16.504 ($2.03x10^{-33}$)
Replanning	+14.157 ($3.26x10^{-67}$)	+17.585 ($3.87x10^{-66}$)	+20.697 ($2.39x10^{-59}$)	+21.838 ($1.66x10^{-59}$)
Continual Replanning	+8.406 ($6.93x10^{-49}$)	+10.4 ($1.65x10^{-49}$)	+11.407 ($3.13x10^{-55}$)	+12.177 ($2.28x10^{-53}$)
CAMP-BDI Quality	+0.081 (0.034)	+0.127 (0.017)	+0.157 ($3.8x10^{-5}$)	+0.143 ($3.47x10^{-5}$)

Figure B.6: Differences (p in brackets) in average activity success rate (% of activities that completed successfully) between *CAMP-BDI.Speed* and other approaches in World B, for all n_{exo} and n_{risk} configurations. Positive values show a greater percentage of activities successfully executed for *CAMP-BDI.Speed* agents than those using any other approach.

B.3 Average Delivery Cost

B.3.1 World A – Average Delivery Cost

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	15.417 (1.384)	15.57 (1.683)	17.98 (5.38)	17.74 (4.64)
Replanning	13.58 (0.549)	14.58 (0.796)	17.04 (1.643)	18.76 (2.383)
Continual	12.91 (0.594)	13.54 (0.605)	14.29 (0.984)	14.92 (1.04)
CAMP-BDI.Spd	13.19 (0.409)	13.17 (0.403)	13.2 (0.447)	13.17 (0.429)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	19.938 (2.81)	22.45 (4.819)	23.85 (6.718)	22.66 (4.631)
Replanning	15.99 (0.685)	18.44 (1.233)	23.68 (2.939)	29.01 (7.75)
Continual	15.08 (0.631)	16.44 (1.083)	18.58 (1.865)	20.52 (2.417)
CAMP-BDI.Spd	16.74 (0.886)	16.83 (0.845)	16.56 (0.709)	16.7 (0.721)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	23.538 (3.709)	30.82 (10.963)	33.94 (11.164)	32.13 (10.66)
Replanning	17.99 (0.999)	22.47 (2.243)	29.54 (4.401)	37.62 (8.158)
Continual	17.28 (0.881)	19.68 (1.433)	23.41 (2.077)	25.73 (2.95)
CAMP-BDI.Spd	23.24 (1.262)	23.03 (1.346)	22.92 (1.106)	22.87 (1.135)
CAMP-BDI.Qty	22.32 (1.01)	22.18 (0.849)	22.47 (1.078)	22.48 (1.097)

Figure B.7: Average delivery cost (activities per goal achieved) in World A, with standard deviation in brackets, for all n_{exo} and n_{risk}. Each cell defines how many activities were executed per goal achieved, for that (row-defined) approach at the given (column-defined) level of n_{risk}

B.3.2 World B – Average Delivery Cost

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	24.279 (5.332)	27.89 (4.866)	28.92 (8.236)	28.85 (11.186)
Replanning	20.01 (0.812)	23.75 (1.79)	28.68 (2.967)	33.87 (5.221)
Continual	19.45 (0.749)	20.77 (1.153)	23.31 (1.4)	24.15 (1.97)
CAMP-BDI.Spd	19.884 (0.732)	20.08 (0.794)	20.05 (0.871)	19.92 (0.749)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	35.68 (5.61)	38.581 (11.057)	40.77 (10.397)	35.69 (9.858)
Replanning	24.64 (0.957)	33.66 (3.875)	44.13 (6.773)	49.95 (8.612)
Continual	24.03 (1.033)	27.6 (1.728)	33.68 (3.058)	35.77 (5.367)
CAMP-BDI.Spd	29.22 (1.538)	29.2 (1.391)	29.4 (1.395)	29.5 (1.32)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	50.767 (15.624)	55.3 (22.258)	53.8 (26.846)	51.02 (36.012)
Replanning	30.85 (1.769)	45.31 (5.317)	61.09 (15.844)	66.19 (15.619)
Continual	29.23 (1.718)	37.96 (3.991)	42.26 (4.357)	49.04 (7.75)
CAMP-BDI.Spd	64.53 (6.088)	64.97 (5.77)	66.5 (6.567)	66.6 (5.961)
CAMP-BDI.Qty	52.75 (3.804)	52.45 (3.351)	53.15 (4.48)	53.05 (4.898)

Figure B.8: Average delivery cost (activities per goal achieved) in World B, with standard deviation in brackets, for all n_{exo} and n_{risk}.

B.3.3 World A – Differences between CAMP-BDI.Speed and other Approaches

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-2.224 ($9.16x10^{-17}$)	-2.396 ($2.42x10^{-15}$)	-1.645 ($6.64x10^{-9}$)	-4.563 $1.99x10^{-10}$)
Replanning	-0.392 (0.0001)	-1.408 ($1.23x10^{-17}$)	-3.271 ($4.33x10^{-7}$)	-5.591 ($6.95x10^{-25}$)
Continual Replanning	+0.278 (0.006)	-0.371 (0.0004)	-1.088 ($1.3x10^{-10}$)	-1.746 ($1.22x10^{-16}$)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-3.197 ($2.93x10^{-11}$)	-5.614 ($8.29x10^{-13}$)	-7.29 ($1.69x10^{-11}$)	-5.964 ($1.79x10^{-13}$)
Replanning	+0.754 ($1.17x10^{-7}$)	-1.609 ($2.29x10^{-11}$)	-7.116 ($2.68x10^{-25}$)	-12.309 ($8.09x10^{-18}$)
Continual Replanning	+1.66 ($3.15x10^{-17}$)	+0.391 (0.043)	-2.019 ($1.64x10^{-10}$)	-3.819 ($9.23x10^{-18}$)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-0.301 (0.525)	-7.782 ($1.12x10^{-6}$)	-11.015 ($3.79x10^{-10}$)	-9.257 ($1.4x10^{-8}$)
Replanning	+5.248 ($2.87x10^{-33}$)	+0.567 (0.107)	-6.623 ($4.53x10^{-16}$)	-15.042 ($1.56x10^{-20}$)
Continual Replanning	+5.962 ($2.31x10^{-39}$)	+3.358 ($7.64x10^{-19}$)	-0.487 (0.144)	-2.859 ($6.16x10^{-9}$)
CAMP-BDI Quality	+0.92 ($9.43x10^{-6}$)	+0.85 (0.0001)	+0.456 (0.025)	+0.392 (0.055)

Figure B.9: Differences (p in brackets) in average goal cost between *CAMP-BDI.Speed* and other approaches in World A. Each table corresponds to a n_{exo} configuration; each cell gives the difference between *CAMP-BDI.Speed* and some (row-defined) approach, under a given (column defined) n_{risk}. Negative values indicate *CAMP-BDI.Speed* executed, on average, less activities for each achieved goal then the compared approach – i.e. was more efficient in activity terms.

B.3.4 World B – Differences between CAMP-BDI.Speed and other Approaches

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-4.395 ($1.22x10^{-17}$)	-7.807 ($4.34x10^{-17}$)	-8.87 ($1.91x10^{-11}$)	-8.927 ($8.31x10^{-8}$)
Replanning	-0.125 (0.361)	-3.669 ($4.19x10^{-20}$)	-8.623 ($2.14x10^{-28}$)	-13.948 ($2.25x10^{-28}$)
Continual Replanning	+0.427 (0.001)	-0.686 (0.0004)	-3.255 ($6.3x10^{-21}$)	-4.227 ($7.28x10^{-22}$)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-5.879 ($1.23x10^{-10}$)	-9.383 ($2.05x10^{-8}$)	-11.371 ($1.55x10^{-11}$)	-6.187 ($1.46x10^{-5}$)
Replanning	+4.578 ($1.12x10^{-28}$)	-4.463 ($9.91x10^{-12}$)	-14.732 ($8.84x10^{-24}$)	-20.449 ($2.58x10^{-25}$)
Continual Replanning	+5.186 ($4.33x10^{-29}$)	+1.603 ($3.56x10^{-7}$)	-4.282 ($5.83x10^{-14}$)	-6.273 ($3.02x10^{-12}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+13.761 ($2.01x10^{-08}$)	+9.668 (0.003)	+12.7 (0.0005)	+15.583 (0.002)
Replanning	+33.676 ($1.5x10^{-45}$)	+19.656 ($3.08x10^{-27}$)	+5.414 (0.017)	+0.406 (0.851)
Continual Replanning	+35.299 ($1.39x10^{-46}$)	+27.0111 ($5.23x10^{-37}$)	+21.248 ($3.4x10^{-27}$)	+17.56 ($1.01x10^{-20}$)
CAMP-BDI Quality	+11.777 ($5.03x10^{-20}$)	+12.525 ($6.16x10^{-19}$)	+13.353 ($7.77x10^{-19}$)	+13.55 ($6.72x10^{-21}$)

Figure B.10: Differences (p in brackets) in average goal cost between *CAMP-BDI.Speed* and other approaches in World A, for all n_{exo} and n_{risk} configurations. Negative values indicate *CAMP-BDI.Speed* executed, on average, less activities for each achieved goal then the compared approach – i.e. was more efficient in activity terms.

B.3.5 Differences between CAMP-BDI.Quality and other Approaches

World A	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+4.328 ($1.25x10^{-31}$)	-0.283 (0.339)	-7.079 ($2.44x10^{-18}$)	-15.434 ($5.68x10^{-21}$)
Continual Replanning	+5.042 ($5.58x10^{-37}$)	+2.508 ($1.35x10^{-19}$)	-0.943 (0.003)	-3.251 ($8.17x10^{-12}$)

World B	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+21.899 ($5.36x10^{46}$)	+7.131 ($2.69x10^{-12}$)	-7.939 (0.0003)	-13.144 ($1.13x10^{-7}$)
Continual Replanning	+23.522 ($4.8x10^{-47}$)	+14.486 ($4.38x10^{-28}$)	+7.895 ($3.26x10^{-15}$)	+4.01 (0.001)

Figure B.11: Differences (p in brackets) in average goal cost between *CAMP-BDI.Quality* and replanning approaches when evaluated in $n_{exo} = 3$; each table corresponds to a different world geography. Negative values indicate *CAMP-BDI.Quality* executed, on average, less activities for each achieved goal then the compared approach – i.e. was more efficient in activity terms.

B.4 Planning Operations Per Goal

B.4.1 World A – Average Planning Operations Per Goal

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	0.9 (0.187)	1.27 (0.34)	2.58 (0.906)	4.2 (1.807)
Continual	6.55 (0.393)	7.12 (0.487)	7.67 (0.771)	8.29 (0.892)
CAMP-BDI Speed	1.71 (0.365)	1.71 (0.255)	1.65 (0.285)	1.653 (0.242)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	1.77 (0.321)	2.99 (1.356)	6.36 (2.169)	11.21 (5.678)
Continual	7.85 (0.417)	9.13 (0.828)	11.11 (1.775)	13.24 (2.74)
CAMP-BDI Speed	3.45 (0.479)	3.51 (0.503)	3.35 (0.43)	3.35 (0.373)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	2.54 (0.484)	4.77 (1.389)	10.35 (3.364)	18.24 (6.759)
Continual	9.29 (0.635)	11.63 (1.148)	15.46 (1.944)	18.28 (2.969)
CAMP-BDI Speed	8.21 (0.959)	8.18 (0.943)	7.85 (0.838)	7.88 (0.756)
CAMP-BDI Quality	7.47 (0.734)	7.33 (0.672)	7.56 (0.788)	7.55 (0.752)

Figure B.12: Average planning cost (planner operations per goal achieved) in World A, with standard deviation in brackets. Each table provides results for an n_{exo} configuration; each cell provides the average planner operations per goal achieved for that (row-defined) approach in a given (column-defined) n_{risk}.

B.4.2 World B – Average Planning Operations Per Goal

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	1.57 (0.321)	3.03 (1.085)	6.09 (2.104)	10.29 (3.704)
Continual	11.39 (0.549)	12.69 (0.931)	14.86 (2.679)	15.87 (1.77)
CAMP-BDI Speed	3.272 (0.44)	3.28 (0.392)	3.29 (0.406)	3.22 (0.383)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	3.02 (0.372)	7.04 (1.734)	15.09 (4.95)	22.98 (7.6)
Continual	14.47 (0.711)	18.07 (1.372)	24.43 (3.343)	28.07 (5.776)
CAMP-BDI Speed	7.8 (0.868)	7.71 (0.726)	7.81 (0.782)	7.77 (0.76)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	5.21 (2.754)	12.67 (3.045)	27.69 (12.096)	38.7 (13.191)
Continual	17.95 (1.21)	27.35 (3.756)	37.27 (5.852)	43.77 (9.558)
CAMP-BDI Speed	33.28 (3.958)	33.58 (3.733)	34.47 (4.579)	34.7 (3.933)
CAMP-BDI Quality	33.42 (2.873)	33.4 (3.851)	33.55 (3.977)	33.72 (4.646)

Figure B.13: Average planning cost (planner operations per goal achieved) in World B, with standard deviation in brackets, for all n_{exo} and n_{risk} configurations.

B.4.3 World A – Differences between CAMP-BDI.Speed and other Approaches

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+0.812 ($6.22x10^{-30}$)	+0.433 ($1.99x10^{-10}$)	-0.925 ($1.75x10^{-9}$)	-2.551 ($4.3x10^{-15}$)
Continual Replanning	-4.839 ($9.17x10^{-61}$)	-5.411 ($2.24x10^{-61}$)	-6.016 ($3.17x10^{-53}$)	-6.635 ($4.21x10^{-52}$)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+1.687 ($5.3x10^{-31}$)	+0.521 (0.007)	-3.01 ($1.14x10^{-14}$)	-7.863 ($2.54x10^{-15}$)
Continual Replanning	-4.394 ($5.58x10^{-52}$)	-5.625 ($5.78x10^{-46}$)	-7.774 ($9.79x10^{-39}$)	-9.889 ($2.003x10^{-35}$)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+5.67 ($3.29x10^{-45}$)	+3.41 ($1.47x10^{-22}$)	-2.501 ($7.23x10^{-7}$)	-10.361 ($8.75x10^{-17}$)
Continual Replanning	-1.082 ($1.1x10^{-10}$)	-3.45 ($1.42x10^{-25}$)	-7.61 ($4.38x10^{-34}$)	-10.405 ($9.08x10^{-34}$)
CAMP-BDI Quality	+0.736 ($1.21x10^{-6}$)	+0.845 ($4.84x10^{-7}$)	+0.287 (0.08)	+0.329 (0.026)

Figure B.14: Differences (p in brackets) in average planning calls per goal between *CAMP-BDI.Speed* and other approaches in World A. Each table corresponds to a n_{exo} configuration; each cell gives the difference between *CAMP-BDI.Speed* and some (row-defined) approach, under a given (column defined) n_{risk}. Negative values show where *CAMP-BDI.Speed* performed, on average, less planning operations for each goal than the compared approach; i.e. was more efficient in planning terms.

B.4.4 World B – Differences between CAMP-BDI.Speed and other Approaches

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+1.701 ($5.78x10^{-34}$)	+90.252 (0.099)	-2.8 ($3x10^{-14}$)	-7.076 ($4.93x10^{-21}$)
Continual Replanning	-8.113 ($3.81x10^{-64}$)	-9.413 ($3.41x10^{-58}$)	-11.564 ($9.81x10^{-8}$)	-12.654 ($5.565x10^{-52}$)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+4.782 ($6.04x10^{-45}$)	+0.665 (0.006)	-7.279 ($1.25x10^{-16}$)	-15.21 ($1.1x10^{-21}$)
Continual Replanning	-6.668 ($2.81x10^{-47}$)	-10.364 ($1.45x10^{-50}$)	-16.622 ($3.11x10^{-43}$)	-20.308 ($1.01x10^{-34}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+28.072 ($8.31x10^{-50}$)	+20.914 ($2.95x10^{-40}$)	+6.776 (0.0002)	-4.004 (0.032)
Continual Replanning	+15.338 ($9.05x10^{-37}$)	+6.237 ($4.23x10^{-14}$)	-2.803 (0.007)	-9.078 ($1.85x10^{-9}$)
CAMP-BDI Quality	-0.132 (0.813)	+0.185 (0.795)	+0.919 (0.265)	+0.972 (0.216)

Figure B.15: Differences (p in brackets) in average planning calls per goal between *CAMP-BDI.Speed* and other approaches in World B, for all n_{exo} and n_{risk} configurations.

B.4.5 Differences between CAMP-BDI.Quality and other Approaches

World A	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+4.934 ($1.78x10^{-44}$)	+2.565 ($4.6x10^{-19}$)	-2.788 ($2.07x10^{-8}$)	-10.69 ($1.26x10^{-17}$)
Continual Replanning	+1.818 ($5.37x10^{-21}$)	-4.296 ($1.74x10^{-34}$)	-7.898 ($6.39x10^{-36}$)	-10.734 ($6.25x10^{-36}$)

World B	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+28.072 ($7.74x10^{-51}$)	+20.914 ($6.37x10^{-40}$)	+6.776 (0.001)	-4.004 (0.01)
Continual Replanning	+15.338 ($8.05x10^{-43}$)	+6.237 ($6.7x10^{-13}$)	-2.803 (0.0001)	-9.078 ($2.68x10^{-10}$)

Figure B.16: Differences (p in brackets) in average planner operations per goal between *CAMP-BDI.Quality* and replanning approaches when evaluated in $n_{exo} = 3$; each table corresponds to a different world geography. Negative values indicate *CAMP-BDI.Quality* executed, on average, less activities for each achieved goal then the compared approach – i.e. was more efficient in planning terms.

B.5 Planning Time Costs

B.5.1 World A – Average Planning Operation Time

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	224647561.1 (24834961.55)	231393504.8 (51839423.95)	219686321.7 (43912349.95)	212509413.4 (39701306.45)
Continual Replanning	180906459.1 (38220649.27)	186429542.1 (41262910.49)	190528407.1 (43017092.4)	188631450.3 (41696859.55)
CAMP-BDI Speed	225714151.2 (51714395.39)	222721444.1 (42008312.45)	217317231.4 (41765224.1)	216349491.4 (46390455.04)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	224820496.1 (27880044.72)	234506435.3 (53428217.35)	212802630.2 (31247267.73)	216365734.2 (30338466.71)
Continual Replanning	163301655.4 (40031105.47)	177180479.8 (45677389)	184857896.5 (44600075.83)	187645935.1 (44644254.04)
CAMP-BDI Speed	194938357.6 (38988265.43)	202234396.7 (34298210.62)	209337763.4 (35623259.83)	188257611.8 (32995741.59)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	187544916.8 (39646416.61)	200408781.7 (41435249.08)	201453497.9 (43202521.91)	192446049.6 (37245175.82)
Continual Replanning	167667566.1 (37689237.58)	175638995.3 (44530566.66)	176017368.3 (40467801.11)	195863054 (45003799.53)
CAMP-BDI Speed	167767101.5 (27825980.59)	178558035.3 (33088372.05)	162855158.7 (28191981.05)	172894569.1 (39014909.21)
CAMP-BDI Quality	567396652.6 (21339269.22)	560764379.2 (10474347.97)	557155862.2 (10420689.36)	560792848.8 (9755132.237)

Figure B.17: Average execution time for each planning operation (in *ns*) in World A, with standard deviation in brackets. Each table provides results for an n_{exo} configuration; each cell provides the average time for that (row-defined) approach in a given (column-defined) n_{risk}. Lower values show planning (on average) completed earlier for that approach, in that n_{exo} and n_{risk} configuration.

B.5.2 World B – Average Planning Operation Time

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	233537035 (34597433.41)	245879491.5 (51577245.44)	258967342.1 (53765514.53)	263671465.2 (56529130.8)
Continual Replanning	177460981.8 (41732617.73)	199075477.2 (46638704.94)	224589561.9 (51235629.56)	230837308.1 (51128754.4)
CAMP-BDI Speed	252398535.1 (52024266.36)	245734399.7 (41902656.47)	260020259 (46423262.04)	251592301.7 (45483471.35)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	248201236.5 (25417262.83)	268250535.7 (29684823.88)	283463172.7 (37968775.33)	287340019.4 (34342844.81)
Continual Replanning	191340826 (45727428.53)	212791696.9 (45565753.51)	230996889.2 (49223185.41)	254803959.1 (56703758.56)
CAMP-BDI Speed	246704384.3 (41417674.17)	238970691.2 (38821570.81)	241178003.4 (35579886.96)	255798051.2 (26923196.43)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	250544242.2 (57906555.72)	256531171.5 (46167195.91)	252909841 (52158409.39)	262311486.2 (56345281.12)
Continual Replanning	190520622.4 (40724621)	225016734.3 (48965863.88)	247540680.3 (55818988.33)	258604877 (63919184.98)
CAMP-BDI Speed	195621445.3 (36746249.53)	195242489.2 (32664428.4)	196395650 (34237814.2)	195018019.9 (30964714.99)
CAMP-BDI Quality	610157472.3 (7620685.32)	600253254.3 (20119749.99)	610476304.8 (17165339.89)	611549866.3 (10558331.65)

Figure B.18: Average execution time for each planning operation (in *ns*) in World B, with standard deviation in brackets, for all n_{exo} and n_{risk} configurations.

B.5.3 World A – Differences between CAMP-BDI.Speed and other Approaches

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	$+0.473\%$ (0.888)	-3.894% (0.347)	-1.394% (0.747)	$+1.775\%$ (0.458)
Continual Replanning	$+19.852\%$ ($2.55x10^{-6}$)	$+16.295\%$ (0.0001)	$+15.764\%$ (0.007)	$+12.817\%$ (0.005)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	-15.329% ($3.76x10^{-6}$)	-15.958% ($4.77x10^{-5}$)	-1.655% (0.461)	-14.931% ($2.17x10^{-8}$)
Continual Replanning	$+16.229\%$ (0.001)	$+12.389\%$ (0.005)	$+11.694\%$ (0.003)	$+0.325\%$ (0.935)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	-11.789% (0.0002)	-12.237% ($1x10^{-5}$)	-23.701% ($2.92x10^{-8}$)	-11.308% (0.0002)
Continual Replanning	$+0.059\%$ (0.984)	$+1.635\%$ (0.53)	-8.082% (0.029)	-13.285% (0.001)

Figure B.19: Percentage difference (p in brackets) between *CAMP-BDI.Speed* and other approaches for average planner operation execution time in World A. Each table corresponds to a n_{exo} configuration; each cell provides the difference between that (row-defined) approach and *CAMP-BDI.Speed* under the given (column-defined) level of n_{risk}. Negative values denote planning operations in *CAMP-BDI.Speed* spent less time on average – i.e. terminated faster – that the compared approach.

B.5.4 World B – Differences between CAMP-BDI.Speed and other Approaches

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	$+7.473\%$ (0.001)	-0.059% (0.967)	$+0.405\%$ (0.805)	-4.801% (0.003)
Continual Replanning	$+29.69\%$ ($7.72x10^{-9}$)	$+18.988\%$ ($3.52x10^{-5}$)	$+13.626\%$ (0.004)	$+8.249\%$ (0.079)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	-0.607% (0.79)	-12.252% ($2.84x10^{-6}$)	-17.533% ($2.17x10^{-9}$)	-12.331% ($1.3x10^{-15}$)
Continual Replanning	$+22.441\%$ ($5.63x10^{-8}$)	$+10.955\%$ (0.003)	$+4.221\%$ (0.232)	$+0.389\%$ (0.889)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	-28.076% ($1.28x10^{-7}$)	-23.526% ($3.48x10^{-13}$)	-28.776% ($1.62x10^{-13}$)	-34.506% ($1.98x10^{-14}$)
Continual Replanning	$+2.607\%$ (0.319)	-11.429% ($3.08x10^{-5}$)	-26.042% ($5.32x10^{-11}$)	-32.606% ($3.35x10^{-11}$)

Figure B.20: Percentage difference (p in brackets) between *CAMP-BDI.Speed* and other approaches for average planner operation execution time in World B, for all n_{exo} and n_{risk} configurations.

B.6 Messaging Costs

B.6.1 World A – Average Messaging Costs

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	14.6 (2.5)	16.8 (4.9)	23.6 (10.2)	25.5 (18.0)
Replanning	7.3 (0.5)	8.5 (1.1)	13.2 (3.9)	20.3 (7.7)
Continual	7.8 (0.6)	8.6 (0.7)	9.8 (1.4)	11.5 (2.9)
CAMP-BDI.Spd	36.5 (2.0)	36.3 (1.8)	36.5 (2.1)	36.1 (1.8)
(excluding **updatedContract**)	7.8 (0.6)	7.7 (0.4)	7.8 (0.6)	7.7 (0.5)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	24.6 (5.3)	34.1 (14.6)	42.3 (24.5)	41.2 (14.0)
Replanning	9.3 (0.7)	12.7 (2.2)	25.3 (8.1)	46 (21.9)
Continual	9.8 (0.6)	11.7 (1.5)	16.2 (4.2)	22.2 (7.3)
CAMP-BDI.Spd	53 (4)	53.5 (4.2)	52.4 (3.3)	52.9 (3.3)
(excluding **updatedContract**)	11.6 (1)	11.6 (1.1)	11.4 (0.8)	11.4 (0.8)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	34.1 (9.0)	57.8 (34.8)	72.1 (38.6)	74.6 (40.2)
Replanning	10.8 (0.8)	17.5 (3.6)	38.3 (11.8)	72.4 (26.5)
Continual	11.8 (0.8)	15.2 (1.8)	24.8 (4.5)	34.8 (10.1)
CAMP-BDI.Spd	84.9 (6.0)	84.5 (6.5)	84.1 (5.6)	82.9 (6.0)
(excluding **updatedContract**)	20.3 (1.7)	20.3 (1.9)	19.9 (1.6)	19.7 (1.6)
CAMP-BDI.Qty	80.0 (5.0)	79.3 (4.3)	80.9 (5.4)	80.9 (5.9)
(excluding **updatedContract**)	19.0 (1.5)	18.9 (1.4)	19.4 (1.7)	19.3 (1.7)

Figure B.21: Average messaging cost (messages sent per goal achieved) in World A, with standard deviation in brackets. Each table provides results for an n_{exo} configuration; each cell provides the average message sent per goal achieved for that (row-defined) approach in a given (column-defined) n_{risk}.

B.6.2 World B – Average Messaging Costs

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	23.5 (4.5)	35.6 (15.9)	39.2 (22.9)	39.4 (18.4)
Replanning	9.2 (0.6)	13.8 (2.7)	25.8 (7.5)	44.1 (16.2)
Continual	11.3 (0.7)	13.5 (1.3)	17.3 (2.6)	20.5 (5.3)
CAMP-BDI.Spd	53.5 (3.1)	54.2 (3.3)	54.2 (3.7)	53.5 (2.9)
(excluding **updatedContract**)	11.0 (0.8)	11.0 (0.8)	11.1 (0.9)	10.9 (0.8)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	46.1 (12.8)	58.6 (22.9)	69.9 (29.8)	66.7 (27.0)
Replanning	12.5 (0.8)	23.4 (5.2)	51.9 (16.1)	87.6 (29.5)
Continual	14.9 (1.0)	19.8 (2.2)	32.6 (7.8)	45.8 (14.8)
CAMP-BDI.Spd	95.3 (7.6)	95.0 (6.0)	95.7 (6.7)	96.7 (5.6)
(excluding **updatedContract**)	20.7 (1.8)	20.6 (1.5)	20.8 (1.8)	21.0 (1.5)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	80.2 (31.9)	103.3 (53.8)	111.5 (71.2)	109.7 (93.9)
Replanning	17.4 (3.0)	38.9 (9.0)	94.4 (39.2)	151.4 (60.1)
Continual	20.4 (1.8)	34.6 (6.6)	61.5 (16.7)	83.6 (29.9)
CAMP-BDI.Spd	271.6 (29.3)	272.9 (28.4)	279.1 (30.4)	278.7 (27.7)
(excluding **updatedContract**)	70.8 (8.4)	71.0 (8.0)	72.4 (8.7)	72.7 (7.9)
CAMP-BDI.Qty	215.8 (17.8)	213.1 (17.4)	216.7 (22.1)	216.9 (24.3)
(excluding **updatedContract**)	53.7 (5.2)	52.8 (5.3)	53.9 (6.4)	54.1 (7.2)

Figure B.22: Average messaging cost (messages sent per goal achieved) in World B, with standard deviation in brackets, for all n_{exo} and n_{risk} configurations.

B.6.3 World A – Absolute Messaging Costs

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	687.2 (36.1)	649.3 (103.6)	568.2 (204)	642.7 (127.5)
Replanning	715.2 (44.7)	755.3 (69.2)	960.2 (235.3)	1176.7 (280.1)
Continual	764.5 (54.8)	781.3 (57.8)	856.9 (91.7)	946.5 (198.7)
CAMP-BDI.Spd	3585 (234.3)	3580 (184.8)	3604.2 (194.5)	3561.7 (175.5)
(excluding **updatedContract**)	767.7 (48.1)	763.6 (42.5)	766.5 (49.4)	757.1 (43.6)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	757.9 (34.7)	750.3 (29.2)	741.9 (33.6)	712.3 (86.6)
Replanning	908.6 (61.8)	982.8 (140.6)	1313.6 (256.1)	1553.2 (328.1)
Continual	950.3 (55.8)	971 (89.5)	1146.8 (208)	1311 (268.3)
CAMP-BDI.Spd	5168 (388.1)	5225.4 (390.2)	5130 (318.1)	5184.1 (328.2)
(excluding **updatedContract**)	1127.7 (92.5)	1135.9 (97.9)	1111.2 (76.1)	1117.5 (78)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	775.4 (32.7)	769.9 (35.3)	760.6 (41.7)	756 (42.6)
Replanning	1058.5 (71.3)	1168.7 (155.8)	1536.4 (285)	1746.5 (300.7)
Continual	1148.2 (70.4)	1162.5 (92.9)	1427.3 (169)	1627.4 (352.2)
CAMP-BDI.Spd	7718.9 (599.1)	7676.6 (570.1)	7744.5 (533)	7707.2 (547.2)
(excluding **updatedContract**)	1840.2 (141.2)	1843.1 (152.5)	1834.5 (132.8)	1829.6 (137)
CAMP-BDI.Qty	7270.2 (459.2)	7239.4 (466.4)	7339 (440.9)	7409.6 (457.9)
(excluding **updatedContract**)	1729.8 (118.6)	1724.7 (119.2)	1755.9 (123.2)	1767.2 (122.6)

Figure B.23: Total messages sent in World A, with standard deviation in brackets. Each table provides results for an n_{exo} configuration; each cell provides the averaged total messages sent for each an experimental run of that (row-defined) approach in a given (column-defined) n_{risk}.

B.6.4 World B – Absolute Messaging Costs

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	736.8 (34.2)	695 (142.8)	738 (145.4)	754.9 (135.8)
Replanning	891.3 (51.3)	1043.4 (144)	1370 (254.7)	1609 (331.2)
Continual	1078.1 (66.7)	1150.7 (85.8)	1318 (147.8)	1445.9 (323)
CAMP-BDI.Spd	5110.9 (266.6)	5188 (298.5)	5177.8 (352.1)	5126.1 (264.8)
(excluding **updatedContract**)	1049.3 (66.3)	1055.8 (67)	1062.7 (75.6)	1043.7 (68.3)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	794.7 (41)	759.7 (31)	743.2 (35.4)	739.2 (37.6)
Replanning	1193.1 (71.9)	1336.3 (141.7)	1665.1 (245.8)	1839.5 (310.7)
Continual	1412.3 (82.2)	1428.2 (137.2)	1737.5 (246.7)	2023.1 (407)
CAMP-BDI.Spd	8800.5 (573.6)	8838.3 (583.6)	8942.3 (574.6)	8978.2 (496.9)
(excluding **updatedContract**)	1913.4 (131.3)	1917.3 (132.5)	1942.4 (150.7)	1953.3 (121.6)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	791.9 (35.8)	776.6 (43.4)	770.6 (36.11)	769.1 (41.8)
Replanning	1627.3 (279.1)	1713(204.6)	1978.2 (309.7)	2135.8 (549.3)
Continual	1920.3(143.8)	1936.6 (216.9)	2293.5 (447)	2332.6 (439.4)
CAMP-BDI.Spd	17681.1(1775.4)	18041.6(1701.8)	18539.6(1644)	18184.8(1812.6)
(excluding **updatedContract**)	4602 (438.3)	4687.2 (425.2)	4802.4 (413.7)	4738.2 (440.9)
CAMP-BDI.Qty	14375.6(1429.1)	14402.4(1572.6)	14294.6(1327.4)	14124.8(1378.1)
(excluding **updatedContract**)	3569.6 (296.5)	3559 (346.9)	3546.4 (298.5)	3514.4 (307.7)

Figure B.24: Total messages sent in World B, with standard deviation in brackets, for all n_{exo} and n_{risk} configurations.

B.6.5 World A – Absolute Message count differences with increasing n_{risk}

World A $n_{exo} = 1$	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
None	-36.717 (0.015)	-83.583 (0.013)	+90.733 (0.008)
Replanning	+43.633 (0.0003)	+216.667 ($3.77x10^{-8}$)	+228.95($8.74x10^{-6}$)
Continual	+20.867 (0.042)	+84.933 ($1.79x10^{-}7$)	+97.683 (0.001)
CAMP-BDI.Spd (*excluding* **updatedContract**)	-5 (0.885) -4.033 (0.605)	+24.183 (0.511) +2.85 (0.763)	-42.5 (0.22) -9.35 (0.303)

World A $n_{exo} = 2$	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
None	-5.283 (0.342)	+0.6 (0.933)	-17.617 (0.166)
Replanning	+125.533 (0.024)	+306.267 ($2.12x10^{-6}$)	+244.367 ($1.08x10^{-5}$)
Continual	+30.8 (0.019)	+187.35 ($1.32x10^{-7}$)	+171 (0.001)
CAMP-BDI.Spd (*excluding* **updatedContract**)	+57.35 (0.319) +8.167 (0.596)	-95.367 (0.186) -24.7 (0.175)	+54.117 (0.37) +6.267 (0.666)

World A $n_{exo} = 3$	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
None	-6.433 (0.344)	-0.733 (0.921)	+6.683 (0.382)
Replanning	+124.8 ($2.34x10^{-6}$)	+381.567 ($2.87x10^{-13}$)	+214.844 ($6.46x10^{-6}$)
Continual	+21.5 (0.166)	+282.85 ($2.92x10^{-15}$)	+210.967 ($8.62x10^{-5}$)
CAMP-BDI.Spd (*excluding* **updatedContractd**)	-30.833 (0.721) -5.067 (0.821)	+99.55 (0.206) +31.183 (0.155)	+70.6 (0.374) +11.25 (0.607)
CAMP-BDI.Qty (*excluding* **updatedContract**)	-42.233 (0.652) +2.9 (0.897)	+67.867 (0.496) -8.533 (0.732)	-37.333 (0.707) -4.95 (0.838)

Figure B.25: Differences in the *total* messages sent, on average, in World A by each approach over increasing n_{risk} (with p in brackets). Each table corresponds to a given n_{exo}; cells define the change in total messages sent, on average, for that (row-defined) approach over the given (column-defined) increase in n_{risk} – positive values show more messages were sent at the higher n_{risk}.

B.6.6 World B – Absolute Message count differences with increasing n_{risk}

World B $n_{exo} = 1$	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
None	-77.167(0.0002)	+43.05(0.082)	+16.833(0.534)
Replanning	+89.983($4.35x10^{-5}$)	+326.583($2.95x10^{-11}$)	+238.967(0.0001)
Continual	+9.283 (0.458)	+167.3($5.38x10^{-13}$)	+127.933 (0.005)
CAMP-BDI.Spd (*excluding* **updatedContract**)	+77.133 (0.115) +6.433 (0.585)	-10.233 (0.854) +6.95 (0.567)	-51.667 (0.251) -18.983 (0.083)

World B $n_{exo} = 2$	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
None	-32.987 ($1.51x10^{-5}$)	-0.633 (0.93)	+27.617 (0.001)
Replanning	+151.428 ($5.4x10^{-9}$)	+343.167 ($8.41x10^{-14}$)	+174.467 (0.002)
Continual	+20.045 (0.301)	+324.85 ($8.37x10^{-14}$)	+297.667 ($5.91x10^{-5}$)
CAMP-BDI.Spd (*excluding* **updatedContract**)	+37.826 (0.618) +3.877 (0.737)	+103.967 (0.351) +25.15 (0.344)	+35.967 (0.709) +10.867 (0.657)

World B $n_{exo} = 3$	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
None	+12.317 (0.094)	-8.233 (0.313)	-14.4 (0.087)
Replanning	-11.45 (0.932)	+278.15 ($1.93x10^{-6}$)	+160.75 (0.055)
Continual	+14.6 (0.675)	+380.867($4.48x10^{-8}$)	+51.35 (0.556)
CAMP-BDI.Spd (*excluding* **updatedContract**)	+360.517 (0.295) +85.217 (0.305)	+497.917 (0.108) +115.217 (0.147)	-354.783 (0.283) -64.267 (0.424)
CAMP-BDI.Qty (*excluding* **updatedContract**)	+26.783 (0.928) -10.617 (0.871)	-107.733 (0.697) -12.583 (0.836)	-169.833 (0.482) -31.95 (0.564)

Figure B.26: Differences in the *total* messages sent, on average, in World B by each approach over increasing n_{risk} (with p in brackets) in all n_{exo} – positive values show more messages were sent at the higher n_{risk}.

B.6.7 Messaging Costs *including* updatedContract

B.6.7.1 World A – Differences between CAMP-BDI.Speed and other Approaches

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+21.916 ($1.29x10^{-51}$)	+19.557 ($1.72x10^{-36}$)	+12.896 ($4.68x10^{-13}$)	+10.597 ($2.53x10^{-5}$)
Replanning	+29.207 ($5.04x10^{-69}$)	+27.795 ($1.84x10^{-69}$)	+23.27 ($2.19x10^{-42}$)	+15.846 ($8.27x10^{-22}$)
Continual Replanning	+28.721 ($5.5x10^{-69}$)	+27.723 ($2.21x10^{-70}$)	+26.69 ($1.61x10^{-62}$)	+24.643 ($2.04x10^{-54}$)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+28.369 ($2.22x10^{-39}$)	+19.48 ($1.76x10^{-14}$)	+10.143 (0.002)	+11.657 ($9.35x10^{-8}$)
Replanning	+43.715 ($7.55x10^{-64}$)	+40.85 ($6.36x10^{-57}$)	+27.104 ($3.97x10^{-31}$)	+6.881 (0.019)
Continual Replanning	+43.236 ($1.33x10^{-62}$)	+41.809 ($2.46x10^{-59}$)	+36.218 ($9.04x10^{-51}$)	+30.687 ($1.68x10^{-37}$)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+50.821 ($5.19x10^{-43}$)	+26.705 ($2.3x10^{-7}$)	+11.949 (0.023)	+8.349 (0.122)
Replanning	+74.072 ($3.43x10^{-66}$)	+67.024 ($9.88x10^{-60}$)	+45.7190 ($8.28x10^{-34}$)	+10.518 (0.005)
Continual Replanning	+73.082 ($2.86x10^{-66}$)	+69.279 ($1.36x10^{-61}$)	+59.281 ($9.37x10^{-55}$)	+48.104 ($7.52x10^{-14}$)
CAMP-BDI Quality	+4.898 ($1.1x10^{-6}$)	+5.234 ($1.37x10^{-6}$)	+3.165 (0.003)	+2.08 ($7.52x10^{-14}$)

Figure B.27: Difference (p in brackets) between *CAMP-BDI.Speed* and other approaches for average messages per goal in World A (including **updatedContract** messages); each table corresponds to a particular n_{exo} configuration, with cells defining the difference between the given (row-defined) approach and CAMP-BDI under that (column-defined) n_{risk} – positive values indicate *CAMP-BDI.Speed* sent *more* messages on average.

B.6.7.2 World B – Differences between CAMP-BDI.Speed and other Approaches

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+29.99 ($1.14x10^{-43}$)	+18.549 ($1.07x10^{-11}$)	+15.006 ($5.12x10^{-6}$)	+14.03 ($2.12x10^{-7}$)
Replanning	+44.294 ($7.7x10^{-70}$)	+40.387 ($5.5x10^{-61}$)	+28.406 ($1.31x10^{-32}$)	+9.379 ($7.69x10^{-5}$)
Continual Replanning	+42.267 ($3.09x10^{-68}$)	+40.673 ($1.17x10^{-64}$)	+36.973 ($4.63x10^{-54}$)	+32.943 ($2.24x10^{-46}$)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+49.19 ($2.93x10^{-34}$)	+36.441 ($5.89x10^{-17}$)	+25.855 ($1.08x10^{-8}$)	+29.972 ($1.14x10^{-11}$)
Replanning	+82.712 ($7.3x10^{-62}$)	+71.662 ($1.99x10^{-57}$)	+43.838 ($1.81x10^{-28}$)	+9.003 (0.028)
Continual Replanning	+80.327 ($2.37x10^{-60}$)	+75.298 ($2.29x10^{-64}$)	+63.089 ($1.77x10^{-49}$)	+50.813 ($5.83x10^{-34}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	+191.337 ($1.7x10^{-41}$)	+169.565 ($1.64x10^{-28}$)	+167.542 ($1.24x10^{-24}$)	+168.98 ($1.48x10^{-19}$)
Replanning	+254.195 ($1.09x10^{-57}$)	+233.912 ($1.19x10^{-55}$)	+184.672 ($1.76x10^{-36}$)	+127.321 ($1.6x10^{-21}$)
Continual Replanning	+251.164 ($5.99x10^{-57}$)	+238.258 ($8.86x10^{-57}$)	+217.567 ($1.34x10^{-49}$)	+195.149 ($1.44x10^{-42}$)
CAMP-BDI Quality	+55.797 ($1.4x10^{-19}$)	+59.758 ($1.06x10^{-17}$)	+62.418 ($1.59x10^{-18}$)	+61.83 ($3.83x10^{-20}$)

Figure B.28: Difference (p in brackets) between *CAMP-BDI.Speed* and other approaches for average messages per goal in World B (including **updatedContract** messages), for all n_{exo} and n_{risk} configurations – positive values indicate *CAMP-BDI.Speed* sent *more* messages on average.

B.6.7.3 Differences between CAMP-BDI.Quality and other Approaches

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+69.174 ($7.61x10^{-69}$)	+61.79 ($4.1x10^{-65}$)	+42.554 ($4.88x10^{-36}$)	+8.438 (0.023)
Continual Replanning	+68.184 ($9.65x10^{-69}$)	+64.045 ($1.09x10^{-71}$)	+56.116 ($3.52x10^{-54}$)	+46.024 ($4.73x10^{-40}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+198.397 ($2.16x10^{-63}$)	+174.153 ($3.35x10^{-58}$)	+122.254 ($4.82x10^{-31}$)	+65.491 ($3.97x10^{-10}$)
Continual Replanning	+195.367 ($3.16x10^{-63}$)	+178.5 ($7.81x10^{-59}$)	+155.149 ($1.23x10^{-45}$)	+133.319 ($2.24x10^{-35}$)

Figure B.29: Difference (p in brackets) between *CAMP-BDI.Quality* and other approaches for average messages per goal in World A and B for $n_{exo} = 3$ (including **updatedContract** messages); each cell define the difference between the given (row-defined) approach and CAMP-BDI under that (column-defined) n_{risk} – positive values indicate *CAMP-BDI.Quality* sent *more* messages on average.

B.6.7.4 Messaging Cost differences with increasing n_{risk}

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-0.178 (0.585)	+0.198 (0.604)	-0.391 (0.294)
$n_{exo} = 2$	+0.556 (0.376)	-1.112 (0.152)	+0.42 (0.501)
$n_{exo} = 3$			
CAMP-BDI.Spd	-0.404 (0.67)	-0.425 (0.718)	-1.119 (0.252)
CAMP-BDI.Qty	-0.74 (0.382)	+1.644 (0.043)	-0.034 (0.974)

World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+0.618 (0.258)	+0.068 (0.91)	-0.754 (0.137)
$n_{exo} = 2$	-0.204 (0.969)	+0.659 (0.596)	+0.9444 (0.407)
$n_{exo} = 3$			
CAMP-BDI.Spd	+1.294 (0.812)	+6.215 (0.29)	-0.357 (0.947)
CAMP-BDI.Qty	-2.667 (0.461)	+3.556 (0.312)	+0.2 (0.958)

Figure B.30: Messaging cost differences (p in brackets) for *CAMP-BDI* in World A and B *including* **updatedContract**, over increasing n_{risk} for all n_{exo}. Each table corresponds to an experimental geography; cells define the difference between average messages between the (column defined) lower and higher value of n_{risk} for the (row-defined) n_{exo}; positive values denote more messages were sent per goal when n_{risk} increased.

B.6.8 Messaging Costs *excluding* updatedContract

B.6.8.1 World A – Differences between CAMP-BDI.Speed and other Approaches

World A $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-6.751 ($4.9x10^{-28}$)	-9.013 ($2.02x10^{-20}$)	-15.853 ($3.21x10^{-17}$)	-17.848 ($2.38x10^{-10}$)
Replanning	+0.54 ($8.25x10^{-6}$)	-0.774 ($2.94x10^{-6}$)	-5.479 ($6.84x10^{-15}$)	-12.598 ($3.28x10^{-18}$)
Continual Replanning	+0.054 (0.636)	-0.846 ($1.71x10^{-10}$)	-2.059 ($1.13x10^{-15}$)	-3.802 ($1.14x10^{-14}$)

World A $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-13.055 ($2.85x10^{-26}$)	-22.424 ($3.79x10^{-17}$)	-30.932 ($7.63x10^{-14}$)	-29.804 ($1.72x10^{-23}$)
Replanning	+2.292 ($4.66x10^{-23}$)	-1.054 (0.002)	-13.97 ($4.49x10^{-19}$)	-34.579 ($1.14x10^{-17}$)
Continual Replanning	+1.812 ($2.68x10^{-17}$)	-0.095 (0.696)	-4.856 ($6.04x10^{-12}$)	-10.774 ($1.97x10^{-16}$)

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-13.814 ($4.29x10^{-17}$)	-37.485 ($1.69x10^{-11}$)	-52.183 ($6.36x10^{-15}$)	-54.893 ($4.47x10^{-15}$)
Replanning	+9.438 ($4.67x10^{-45}$)	+2.833 ($4.63x10^{-7}$)	-18.412 ($4.79x10^{-17}$)	-52.725 ($9.58x10^{-22}$)
Continual Replanning	+8.447 ($2.45x10^{-43}$)	+5.088 ($1.4x10^{-22}$)	-4.85 ($3.67x10^{-10}$)	-15.138 ($3.04x10^{-16}$)
CAMP-BDI Quality	+1.21 ($1.5x10^{-5}$)	+1.398 ($6.95x10^{-6}$)	+0.556 (0.087)	+0.404 (0.218)

Figure B.31: Difference (p in brackets) between *CAMP-BDI.Speed* and other approaches for average messages per goal in World A (*excluding* **updatedContract** messages); each table corresponds to a particular n_{exo} configuration, with cells defining the difference between the given (row-defined) approach and CAMP-BDI under that (column-defined) n_{risk} – negative values indicate where *CAMP-BDI.Speed* sent *less* messages on average.

B.6.8.2 World B – Differences between CAMP-BDI.Speed and other ApproachesB

World B $n_{exo} = 1$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-12.55 ($2.77x10^{-28}$)	-24.582 ($4.11x10^{-17}$)	-28.083 ($2.38x10^{-13}$)	-28.55 ($1.97x10^{-17}$)
Replanning	+1.754 ($7.06x10^{-20}$)	-2.744 ($2.72x10^{-10}$)	-14.684 ($2.49x10^{-21}$)	-33.201 ($1.434x10^{-22}$)
Continual Replanning	-0.273 (0.052)	-2.458 ($3.99x10^{-18}$)	-6.117 ($1.05x10^{-23}$)	-9.637 ($3.1x10^{-20}$)

World B $n_{exo} = 2$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-11.039 ($1.3x10^{-22}$)	-37.983 ($1.7x10^{-18}$)	-49.053 ($1.56x10^{-18}$)	-45.644 ($5.916x10^{-19}$)
Replanning	+1.015 ($8.42x10^{-40}$)	-2.762 (0.0003)	-31.069 ($9.05x10^{-22}$)	-66.613 ($1.15x10^{-24}$)
Continual Replanning	+0.798 ($2.51x10^{-29}$)	+0.874 (0.0163)	-11.819 ($8.7x10^{-17}$)	-24.803 ($7.13x10^{-19}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
None	-9.436 (0.029)	-32.312 ($3.25x10^{-5}$)	-39.134 ($8.08x10^{-5}$)	-36.992 (0.004)
Replanning	+53.422 ($2.57x10^{-50}$)	+32.034 ($1.34x10^{-29}$)	-22.005 ($8.89x10^{-5}$)	-78.651 ($2.67x10^{-14}$)
Continual Replanning	+50.391 ($9.13x10^{-48}$)	+36.38 ($4.06x10^{-37}$)	+10.89 ($2.35x10^{-5}$)	-10.823 (0.009)
CAMP-BDI Quality	+17.06 ($1.99x10^{-21}$)	+18.156 ($9x10^{-19}$)	+18.509 ($2.69x10^{-19}$)	+18.596 ($1.06x10^{-20}$)

Figure B.32: Difference (p in brackets) between *CAMP-BDI.Speed* and other approaches for average messages per goal in World A (*excluding* **updatedContract** messages), for all n_{exo} and n_{risk} configurations – negative values indicate where *CAMP-BDI.Speed* sent *less* messages on average.

B.6.8.3 Differences between CAMP-BDI.Quality and other Approaches

World A $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+8.228 ($2x10^{-42}$)	+1.435 (0.004)	-18.968 ($2.92x10^{-18}$)	-53.128 ($5.38x10^{-22}$)
Continual Replanning	+7.237 ($9.41x10^{-41}$)	+3.69 ($1.26x10^{-20}$)	-5.406 ($6.37x10^{-12}$)	-15.542 ($1.71x10^{-17}$)

World B $n_{exo} = 3$	$n_{risk} = 0$	$n_{risk} = 0.25$	$n_{risk} = 0.5$	$n_{risk} = 0.75$
Replanning	+36.362 ($1.76x10^{-48}$)	+13.878 ($1.33x10^{-15}$)	-40.514 ($4.09x10^{-11}$)	-97.248 ($9.27x10^{-18}$)
Continual Replanning	+33.332 ($1.85x10^{-48}$)	+18.224 ($2.33x10^{-23}$)	-7.619 (0.002)	-29.429 ($3.86x10^{-10}$)

Figure B.33: Difference (p in brackets) between *CAMP-BDI.Quality* and other approaches for average messages per goal in World A and B for $n_{exo} = 3$ (excluding **updatedContract** messages); each cell define the difference between the given (row-defined) approach and CAMP-BDI under that (column-defined) n_{risk} – positive values indicate *CAMP-BDI.Quality* sent *more* messages on average.

B.6.8.4 Differences with increasing n_{risk}

World A	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	-0.081 (0.424)	+0.019 (0.851)	-0.086 (0.39)
$n_{exo} = 2$	+0.076 (0.666)	-0.283 (0.158)	+0.034 (0.825)
$n_{exo} = 3$			
CAMP-BDI.Spd	+0.04 (0.879)	-0.366 (0.271)	+0.23 (0.418)
CAMP-BDI.Qty	-0.148 (0.583)	+0.476 (0.09)	0.078 (0.804)

World B	$n_{risk} = 0 \rightarrow\ 0.25$	$n_{risk} = 0.25 \rightarrow\ 0.5$	$n_{risk} = 0.5 \rightarrow\ 0.75$
$n_{exo} = 1$	+0.027 (0.846)	+0.109 (0.444)	-0.244 (0.066)
$n_{exo} = 2$	-0.098 (0.854)	+0.175 (0.584)	+0.236 (0.447)
$n_{exo} = 3$			
CAMP-BDI.Spd	+0.19 (0.9)	+1.416 (0.404)	+0.347 (0.821)
CAMP-BDI.Qty	+0.906 (0.396)	+1.063 (0.299)	+0.26 (0.834)

Figure B.34: Messaging cost differences (p in brackets) for *CAMP-BDI* in World A and B *excluding* **updatedContract**, over increasing n_{risk} for all n_{exo}. Each table corresponds to an experimental geography; cells define the difference between average messages between the (column defined) lower and higher value of n_{risk} for the (row-defined) n_{exo}; positive values denote more messages were sent per goal when n_{risk} increased.

Appendix C

Publications

Work based upon earlier design and experimental results was presented at the 18th Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2015):

Alan White, Austin Tate, Michael Rovatsos. **CAMP-BDI: A Pre-emptive Approach for Plan Execution Robustness in Multiagent Systems**. In Qingliang Chen, Paolo Torroni, Serena Villata, Jane Yung-jen Hsu, Andrea Omicini, editors, *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*. Volume 9387 of *Lecture Notes in Computer Science*, pages 65-84, Springer, 2015.

`http://www.aiai.ed.ac.uk/project/ix/documents/2015/2015-prima-white-camp-bdi.pdf`

Bibliography

Alexandre Albore, Héctor Palacios, and Héctor Geffner. Fast and Informed Action Selection for Planning with Sensing. In Daniel Borrajo, Luis A. Castillo, and Juan M. Corchado, editors, *Current Topics in Artificial Intelligence, 12th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2007, Salamanca, Spain, November 12-16, 2007. Selected Papers*, volume 4788 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2007.

David N. Allsopp, Patrick Beautement, Jeffrey M. Bradshaw, Edmund H. Durfee, Michael Kirton, Craig A. Knoblock, Niranjan Suri, Austin Tate, and Craig W. Thompson. Coalition agents experiment: Multiagent cooperation in international coalitions. *IEEE Intelligent Systems*, 17(3):26–35, 2002.

Jose A. Ambros-Ingerson and Sam Steel. Integrating Planning, Execution and Monitoring. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence. St. Paul, MN, August 21-26, 1988.*, pages 83–88. AAAI Press / The MIT Press, 1988.

Chitta Baral and Tran Cao Son. Approximate Reasoning about Actions in Presence of Sensing and Incomplete Information. In Jan Maluszynski, editor, *ILPS*, pages 387–401. MIT Press, 1997.

Jeremy W. Baxter and Graham S. Horn. *Controlling Teams of Uninhabited Air Vehicles*, pages 97–112. Birkhäuser Basel, Basel, 2008.

Guido Boella and Rossana Damiano. A Replanning Algorithm for a Reactive Agent Architecture. In Donia Scott, editor, *Artificial Intelligence: Methodology, Systems, and Applications, 10th International Conference, AIMSA 2002, Varna, Bulgaria, September 4-6, 2002, Proceedings*, volume 2443 of *Lecture Notes in Computer Science*, pages 183–192. Springer, 2002.

Blai Bonet and Héctor Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. In Steve A. Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pages 52–61. AAAI, 2000.

Blai Bonet and Héctor Geffner. Planning and Control in Artificial Intelligence: A Unifying Perspective. *Applied Intelligence*, 14(3):237–252, 2001.

Blai Bonet and Héctor Geffner. Planning As Heuristic Search. *Artificial Intelligence*, 129(1-2):5–33, June 2001.

Rafael H. Bordini and Jomi F. Hübner. *Computational Logic in Multi-Agent Systems: 6th International Workshop, CLIMA VI, London, UK, June 27-29, 2005, Revised Selected and Invited Papers*, chapter BDI Agent Programming in AgentSpeak Using Jason, pages 143–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

Craig Boutilier and Richard Dearden. Using Abstractions for Decision Theoretic Planning with Time Constraints. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1016–1022. San Francisco, CA: Morgan Kaufmann, 1994.

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK96)*, pages 195–210, 1996.

Jeffrey M Bradshaw, Stewart Dutfield, Pete Benoit, and John D Woolley. Kaos: toward an industrial-strength open agent architecture. In *Software agents*, pages 375–418. MIT Press, 1997.

Ronen I. Brafman and Carmel Domshlak. Factored Planning: How, When, and When Not. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 809–814, 2006.

Ronen I. Brafman and Carmel Domshlak. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Procs. ICAPS 2008*, pages 28–35. AAAI Press, 2008.

Michael E. Bratman. *Intention, Plans, and Practical Reason*. Cambridge University Press, March 1999.

Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for bdi agent systems. In Rafael H. Bordini, Mehdi Dastani, Jrgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3346 of *Lecture Notes in Computer Science*, pages 44–65. Springer Berlin Heidelberg, 2005.

Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Extending the Capability Concept for Flexible BDI Agent Modularization. In Rafael H. Bordini, MehdiM. Dastani, Jrgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *Lecture Notes in Computer Science*, pages 139–155. Springer Berlin Heidelberg, 2006.

Michael Brenner and Bernhard Nebel. Continual Planning and Acting in Dynamic Multiagent Environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, December 2009.

Jan Broersen, Mehdi Dastani, Joris Hulstijn, Zhisheng Huang, and Leendert W. N. van der Torre. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In *Agents*, pages 9–16, 2001.

Jan Broersen, Mehdi Dastani, and Leendert Van Der Torre. Resolving conflicts between beliefs, obligations, intentions, and desires. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 568–579. Springer, 2001.

Jan Broersen, Mehdi Dastani, Joris Hulstijn, and Leendert van der Torre. Goal generation in the BOID architecture. *Cognitive Science Quarterly*, 2(3-4):428–447, 2002.

Alan Burns and A. J. Wellings. *Real-time Systems and Their Programming Languages*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

Paolo Busetta, Nicholas Howden, Ralph Rönnquist, and Andrew Hodgson. Structuring BDI Agents in Functional Clusters. In *6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL)*, ATAL '99, pages 277–289, London, UK, UK, 2000. Springer-Verlag.

Stefan Bussmann and Klaus Schild. An agent-based approach to the control of flexible production systems. In *ETFA 2001. 8th International Conference on Emerging*

Technologies and Factory Automation. Proceedings (Cat. No.01TH8597), volume 2, pages 481–488 vol.2, Oct 2001.

Tom Bylander. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69:165–204, 1994.

Ibrahim Cakirlar, Erdem Eser Ekinci, and Oguz Dikenelli. *Exception Handling in Goal-Oriented Multi-Agent Systems*, volume 5485 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 2008.

Victoria M. Catterson, Euan M. Davidson, and Stephen D.J. McArthur. Practical applications of multi-agent systems in electric power systems. *European Transactions on Electrical Power*, 22(2):235–252, 2012.

Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM (JACM)*, 43(2):225–267, March 1996.

Edwin K. P. Chong, Robert L. Givan, and Hyeong Soo Chang. A Framework for Simulation-based Network Control via Hindsight Optimization. In *39th IEEE Conf. on Decision and Control*, pages 1433–1438, 2000.

Kum Wah Choy, Adrian A. Hopgood, Lars Nolle, and Brian C. O'Neill. Implementation of a Tileworld Testbed on a Distributed Blackboard System. In *Proceedings of The 18th European Simulation Multiconference*, pages 129–135, 2004.

Bradley J Clement and Anthony C Barrett. Continual coordination through shared activities. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 57–64. ACM, 2003.

Bradley J. Clement and Edmund H. Durfee. Top-down Search for Coordinating the Hierarchical Plans of Multiple Agents. In *Proceedings of the Third Annual Conference on Autonomous Agents*, AGENTS '99, pages 252–259, New York, NY, USA, 1999. ACM.

Armando Walter Colombo, Ronald Schoop, and Ralf Neubert. An agent-based intelligent control platform for industrial holonic manufacturing systems. *IEEE Transactions on Industrial Electronics*, 53(1):322–337, Feb 2005.

Vincent Conitzer. Comparing multiagent systems research in combinatorial auctions and voting. *Annals of Mathematics and Artificial Intelligence*, 58(3-4):239–259, April 2010.

Arie A. Covrigaru and Robert K. Lindsay. Deterministic Autonomous Systems. *AI Magazine*, 12(3):110–117, September 1991.

Jeffrey S. Cox, Edmund H. Durfee, and Thomas Bartold. A Distributed Framework for Solving the Multiagent Plan Coordination Problem. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 821–827, New York, NY, USA, 2005. ACM.

Matthew Crosby, Anders Jonsson, and Michael Rovatsos. A Single-Agent Approach to Multiagent Planning. In *21st European Conf. on Artificial Intelligence (ECAI'14)*, Prague, Czech Republic, 2014.

Jeff Dalton, Jeffrey M. Bradshaw, Austin Tate, and Andrzej Uszok. Coalition Search and Rescue - Task Support: Intelligent Task Achieving Agents on the Semantic Web. Interim Technical Report (Final DAML Program Technical Report), January 2003-December 2004. Technical Report AFRL-IF-RS-TR-2006-91, Air Force Research Laboratory, July 2006. AIAI Contract No. F-30602-03-2-0014 (DARPA Order No. P105/00), IHMC Contract No. F-30602-00-2-0577.

Sylvain Damiani, Gérard Verfaillie, and Marie-Claire Charmeau. An Earth Watching Satellite Constellation: How to Manage a Team of Watching Agents with Limited Communications. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 455–462, New York, NY, USA, 2005. ACM.

Mehdi Dastani, M. Birna van Riemsdijk, and Michael Winikoff. Rich Goal Types in Agent Programming. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 405–412, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

Alessandro de Luna Almeida, Samir Aknine, Jean-Pierre Briot, and Jacques Malenfant. Predictive fault tolerance in multiagent systems: a plan-based replication approach. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*, page 139. IFAAMAS, 2007.

Lavindra de Silva and Lin Padgham. A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning. In Geoffrey I. Webb and Xinghuo Yu, editors,

Australian Conference on Artificial Intelligence, volume 3339 of *Lecture Notes in Computer Science*, pages 1167–1173. Springer, 2004.

Mathijs de Weerdt and Brad Clement. Introduction to Planning in Multiagent Systems. *Multiagent Grid Syst.*, 5(4):345–355, December 2009.

Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David Smith, and Rich Washington. Contingency planning for planetary rovers. October 2002.

Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(02):319–346, 1992.

Keith S. Decker and Victor R. Lesser. Quantitative Modeling of Complex Environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour.*, 2:215–234, January 1993.

Scott A. DeLoach, Walamitien H. Oyenan, and Eric T. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, 2007.

Scott A DeLoach. OMACS: a Framework for Adaptive, Complex Systems. *Complex Systems. In V. Dignum, editor, Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 76–104, 2009.

Marie E. desJardins, Edmund H. Durfee, Charles L. Ortiz, and Michael J. Wolverton. A Survey of Research in Distributed, Continual Planning. *AI Magazine*, 20(4), 1999.

Ralph Deters. Scalable Multi-agent Systems. In *Proceedings of the 2001 Joint ACM-ISCOPE Conference on Java Grande*, JGI '01, pages 182–, New York, NY, USA, 2001. ACM.

Frank Dignum, David Kinny, and Liz Sonenberg. From desires, obligations and norms to goals. *Cognitive Science Quarterly* ., 2(3-4):407–427, 2002.

Yannis Dimopoulos, Alfonso Gerevini, Patrik Haslum, and Alessandro Saetti. The Benchmark Domains of the Deterministic Part of IPC-5. In *Special Booklet for the Planning Competition in the Working Notes of the 16th Int'l Conf. on Automated Planning and Scheduling (ICAPS-06)*, Monterey, CA, USA, June 2006.

Mark dInverno, Koen Hindriks, and Michael Luck. A Formal Architecture for the 3APL Agent Programming Language. In *ZB 2000: Formal Specification and Development in Z and B*, volume 1878 of *Lecture Notes in Computer Science*, pages 168–187. Springer Berlin Heidelberg, 2000.

Klaus Dorer and Monique Calisti. An Adaptive Solution to Dynamic Transport Optimization. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 45–51, New York, NY, USA, 2005. ACM.

Brian Drabble, Jeff Dalton, and Austin Tate. Repairing Plans on the Fly. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, Oxnard, CA, USA, October 1997.

Simon Duff, James Harland, and John Thangarajah. On Proactivity and Maintenance Goals. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1033–1040, New York, NY, USA, 2006. ACM.

Edmund H. Durfee and Victor R. Lesser. Using Partial Global Plans to Coordinate Distributed Problem Solvers. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'87, pages 875–883, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

Edmund Durfee and Victor R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:1167–1183, 1991.

Edmund H. Durfee. Multi-agents Systems and Applications. chapter Distributed Problem Solving and Planning, pages 118–149. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

Stefan Edelkamp, Roman Englert, Jörg Hoffmann, Frederico dos S. Liporace, Sylvie Thiébaux, and Sebastian Trüg. Engineering Benchmarks for Planning: the Domains Used in the Deterministic Part of IPC-4. *CoRR*, abs/1110.1016, 2011.

Eithan Ephrati and Jeffrey S. Rosenschein. A Heuristic Technique for Multiagent Planning. *Annals of Mathematics and Artificial Intelligence*, 20, 1997.

Eithan Ephrati, Martha E. Pollack, and Sigalit Ur T. Deriving multi-agent coordination through filtering strategies. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1995.

Kutluhan Erol, James Handler, and Dana S. Nau. Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9, University Of Maryland, 1994.

Tara Estlin, Rebecca Castano, Robert Anderson, Daniel Gaines, Forest Fisher, and Michele Judd. Learning and Planning for Mars Rover Science. In *Proceedings of IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating*. Morgan Kaufmann Publishers, 2003.

Oren Etzioni, Keith Golden, and Daniel Weld. Tractable Closed World Reasoning with Updates (Extended Abstract). In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning, KR-94*, pages 178–189, 1994.

Alan Fedoruk and Ralph Deters. Improving Fault-tolerance by Replicating Agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, AAMAS '02, pages 737–744, New York, NY, USA, 2002. ACM.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

Avgoustinos Filippoupolitis, George Loukas, Stelios Timotheou, Nikolaos Dimakis, and Erol Gelenbe. Emergency response systems for disaster management in buildings, May 2009.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374–382, April 1985.

Klaus Fischer, Michael Schillo, and Jörg Siekmann. Holonic multiagent systems: A foundation for the organisation of multiagent systems. In *Holonic and Multi-Agent Systems for Manufacturing*, pages 71–80. Springer Berlin Heidelberg, 2003.

Janae Foss, Nilufer Onder, and D Smith. Preventing unrecoverable failures through precautionary planning. In *ICAPS07 Workshop on Moving Planning and Scheduling Systems into the Real World*, 2007.

Maria Fox and Derek Long. PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *In Proc. ICAPS*, pages 212–221. AAAI Press, 2006.

Christian Fritz and Sheila McIlraith. Monitoring Plan Optimality during Execution: Theory and Implementation. In *The 18th International Workshop on Principles of Diagnosis (DX-07)*, Nashville, TN, USA, May 29–31 2007. An extended version of this paper appeared at ICAPS07.

Michael P. Georgeff and Francois Felix Ingrand. Decision-making in an Embedded Reasoning System. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, pages 972–978, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael J. Wooldridge. *Intelligent Agents V: Agents Theories, Architectures, and Languages: 5th International Workshop, ATAL'98 Paris, France, July 4–7, 1998 Proceedings*, chapter The Belief-Desire-Intention Model of Agency, pages 1–10. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

Alfonso Gerevini and Ivan Serina. LPG: A planner based on local search for planning graphs. In *"In Proc. of 6th Int. Conf. on AI Planning Systems (AIPS02)"*. AAAI Press, 2002.

Peter Gregory and Alan Lindsay. The dimensions of driverlog. In *Proceedings of the UK PlanSIG*, 2007.

Barbara J. Grosz, Luke Hunsberger, and Sarit Kraus. Planning and Acting Together. *AI Magazine*, 20(4):23–34, 1999.

Rachid Guerraoui and André Schiper. Software-Based Replication for Fault Tolerance. *Computer*, 30(4):68–74, April 1997.

Zahia Guessoum, Nora Faci, and Jean-Pierre Briot. *Adaptive Replication of Large-Scale Multi-Agent Systems - Towards a Fault-Tolerant Multi-Agent Platform*, pages 238–253. Springer Verlag, February 2005.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent Planning with Factored MDPs. In *In NIPS-14*, pages 1523–1530. The MIT Press, 2001.

Staffan Hägg. A Sentinel Approach to Fault Handling in Multi-Agent Systems. In *Revised Papers from the Second Australian Workshop on Distributed Artificial Intelligence: Multi-Agent Systems: Methodologies and Applications*, pages 181–195, London, UK, UK, 1997. Springer-Verlag.

Christian Hahn, Bettina Fley, and Michael Schillo. Optimisation of multiagent organisation for robustness. *Sozionik Aktuell*, 3(1), 2003.

Karen Zita Haigh and Manuela M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 120–127, Pittsburgh, PA, June 1998.

Liangxiu Han, Stephen Potter, George Beckett, Gavin Pringle, Stephen Welch, Sung-Han Koo, Gerhard Wickler, Asif Usmani, José L. Torero, and Austin Tate. FireGrid: An e-infrastructure for next-generation emergency response support. *Journal of Parallel and Distributed Computing*, 70(11):1128–1141, 2010.

Steve Hanks, Dat Nguyen, and Chris Thomas. A Beginner's Guide to the Truckworld Simulator. Technical report, Dept. of Computer Science and Engineering UW-CSE-TR 93-06-09, University of Washington, 1993.

Steve Hanks, Martha E. Pollack, and Paul R. Cohen. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine*, 14(4):17–42, 1993.

Linli He and Thomas R. Ioerger. A Quantitative Model of Capabilities in Multi-Agent Systems. In *Proceedings of the International Conference on Artificial Intelligence, IC-AI '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 2*, pages 730–736, 2003.

Malte Helmert. *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*. Springer-Verlag, Berlin, Heidelberg, 2008.

James Hendler, Austin Tate, and Mark Drummond. AI Planning: Systems and Techniques. *AI Magazine*, 11(2):61–77, April 1990.

Koen V. Hindriks and M. Birna van Riemsdijk. Satisfying maintenance goals. In Matteo Baldoni, Tran Cao Son, M. Birna van Riemsdijk, and Michael Winikoff, editors, *Declarative Agent Languages and Technologies V: 5th International Workshop, DALT 2007, Honolulu, HI, USA, May 14, 2007, Revised Selected and Invited Papers*, pages 86–103, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Michael W. Hofbaur and Brian C. Williams. *Mode Estimation of Probabilistic Hybrid Systems*, pages 253–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

Jörg Hoffmann and Stefan Edelkamp. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research (JAIR)*, 24(1):519–579, October 2005.

Jörg Hoffmann. FF: The Fast-Forward Planning System. *AI magazine*, 22:57–62, 2001.

Jörg Hoffmann. The metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research (JAIR)*, 20(1):291–341, December 2003.

Mark Hoogendoorn, Catholijn M. Jonker, Martijn C. Schut, and Jan Treur. Modeling Adaptive Multi-agent Organizations for Naval Missions. In *Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, AIKED'06, pages 470–478, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS).

Luke Hunsberger and Barbara J. Grosz. A combinatorial auction for collaborative planning. *Multi-Agent Systems, International Conference on*, 00(undefined):0151, 2000.

Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert: Intelligent Systems and Their Applications*, 7(6):34–44, December 1992.

Manish Jain, Bo An, and Milind Tambe. An Overview of Recent Application Trends at the AAMAS conference: Security, Sustainability and Safety. *AI Magazine*, 33(3):14, 2012.

Yosr Jarraya, Sujoy Ray, Andrei Soeanu, Mourad Debbabi, Mohamad Allouche, and Jean Berger. Towards a Distributed Plan Execution Monitoring Framework. In Elhadi M. Shakshuki, Karim Djouani, Michael Sheng, Mohamed Younis, Eduardo Vaz, and Wayne Groszko, editors, *Proceedings of the 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013), Halifax, Nova Scotia, Canada, June 25-28, 2013*, volume 19 of *Procedia Computer Science*, pages 1034–1039. Elsevier, 2013.

Nicholas R. Jennings. Towards a Cooperation Knowledge Level for Collaborative Problem Solving. In *10th European Conf. on Artificial Intelligence (ECAI-92)*, pages 224–228, Vienna, Austria, 1992.

Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

Sergio Jiménez, Andrew Coles, and Amanda Smith. Planning in Probabilistic Domains Using a Deterministic Numeric Planner. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, December 2006.

Sergio Jiménez, Fernando Fernández, and Daniel Borrajo. Inducing non-deterministic actions behaviour to plan robustly in probabilistic domains. In *ICAPS'06 Workshop on Planning under Uncertainty and Execution Control for Autonomous Systems*, 2006.

Anders Jonsson and Michael Rovatsos. Scaling Up Multiagent Planning: A Best-Response Approach. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*. AAAI, 2011.

Gal A. Kaminka and Milind Tambe. What Is Wrong With Us? Improving Robustness Through Social Diagnosis. In Jack Mostow and Chuck Rich, editors, *AAAI/IAAI*, pages 97–104. AAAI Press / The MIT Press, 1998.

Raveesh Kandiyil and Yang Gao. A Generic Domain Configurable Planner Using HTN For Autonomous Multi-Agent Space System. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, October 2012.

Thomas H. Killion. Decision making and the levels of war. *Military Review*, pages 66–70, November 2000.

David Kinny, Magnus Ljungberg, Anand S. Rao, Liz Sonenberg, Gil Tidhar, and Eric Werner. Planned Team Activity. In *MAAMAW*, volume 830 of *Lecture Notes in Computer Science*, pages 227–256. Springer, 1992.

Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsuhi Shinjou, and Susumu Shimada. RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 739–746. IEEE Computer Society, 1999.

Mark Klein and Chrysanthos Dellarocas. Exception Handling in Agent Systems. In *Agents*, pages 62–68, 1999.

Mark Klein, Juan A. Rodríguez-Aguilar, and Chrysanthos Dellarocas. Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):179–189, 2003.

Martin J Kollingbaum and Timothy J Norman. NoA-a normative agent architecture. In *IJCAI*, pages 1465–1466, 2003.

M Kollingbaum, T Norman, Alun Preece, and Derek Sleeman. Norm refinement: Informing the re-negotiation of contracts. In *ECAI 2006 Workshop on Coordination, Organization, Institutions and Norms in Agent Systems, COIN@ ECAI*, volume 2006, pages 46–51, 2006.

Andrey Kolobov, Mausam, and Daniel S. Weld. ReTrASE: Integrating Paradigms for Approximate Probabilistic Planning. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 1746–1753, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

Antonin Komenda, Jiri Vokrinek, Michal Pěchouček, Gerhard Wickler, Jeff Dalton, and Austin Tate. I-Globe: Distributed Planning and Coordination of Mixed-initiative Activities. In *Proceedings of Knowledge Systems for Coalition Operations (KSCO 2009)*, 2009.

Antonín Komenda, Gerhard Wickler, Jǐrí Vokřínekk, Michal Pěchouček, Jeff Dalton, and Austin Tate. I-Globe: Distributed Planning and Coordination of Team-oriented Activities, 2009.

Antonín Komenda, Peter Novák, and Michal Pěchouček. Decentralized multi-agent plan repair in dynamic environments. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1239–1240, 2012.

Antonín Komenda, Peter Novák, Michal Pěchouček, Raz Nissim, Daniel L Kovacs, and Ronen Brafman. How to repair multi-agent plans: Experimental approach. *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS13)*, 2013.

Roman Van Der Krogt and Mathijs De Weerdt. Plan Repair as an Extension of Planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05*, pages 161–170, 2005.

Sanjeev Kumar and Philip R. Cohen. Towards a Fault-tolerant Multi-agent System Architecture. In *Proceedings of the Fourth International Conference on Autonomous Agents*, AGENTS '00, pages 459–466, New York, NY, USA, 2000. ACM.

Sanjeev Kumar and Philip R. Cohen. Towards a Fault-tolerant Multi-agent System Architecture. In *Proceedings of the Fourth International Conference on Autonomous Agents*, AGENTS '00, pages 459–466, New York, NY, USA, 2000. ACM.

Michael Lees. A history of the Tileworld agent testbed. Technical report, School of Computer Science and Information Technology, University of Nottingham, Nottingham, 2002.

Paulo Leitão. Agent-based Distributed Manufacturing Control: A State-of-the-art Survey. *Engineering Applicatins of Artificial Intelligence*, 22(7):979–991, October 2009.

Victor R. Lesser, Keith S. Decker, Thomas Wagner, Norman Carver, Alan Garvey, Bryan Horling, Daniel Neiman, Rodion Podorozhny, M. V. Nagendra Prasad, Anita Raja, Rgis Vincent, Ping Xuan, and Xiaoqin Zhang. Evolution of the GPGP/-TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

Hector J. Levesque, Philip R. Cohen, and Jos H. T. Nunes. On Acting Together. In Howard E. Shrobe, Thomas G. Dietterich, and William R. Swartout, editors, *AAAI*, pages 94–99. AAAI Press / The MIT Press, 1990.

Iain Little and Sylvie Thibaux. Probabilistic planning vs. replanning. In *In ICAPS Workshop on IPC: Past, Present and Future*, 2007.

Derek Long and Maria Fox. Automatic Synthesis and use of Generic Types in Planning. In *AIPS-00*, pages 196–205. AAAI Press, 2000.

Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. Questions, options, and criteria: Elements of design space analysis. *Hum.-Comput. Interact.*, 6(3):201–250, September 1991.

Janusz Marecki, Nathan Schurr, Milind Tambe, and Paul Scerri. *Safety and Security in Multiagent Systems: Research Results from 2004-2006*, chapter Analyzing Dangers in Multiagent Rescue Using DEFACTO, pages 241–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

Vladimír Mařík, Pavel Vrba, Ken H. Hall, and Francisco P. Maturana. Rockwell Automation Agents for Manufacturing. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 107–113, New York, NY, USA, 2005. ACM.

Stephen D.J. McArthur, Euan M. Davidson, Victoria M. Catterson, Aris L. Dimeas, Nikos D. Hatziargyriou, Ferdinanda Ponci, and Toshihisa Funabashi. Multi-agent systems for power engineering applications - part 1: Concepts, approaches and technical challenges. *IEEE Transactions on Power Systems*, 22(4):1743–1752, 2007.

John McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanisation of Thought Processes*, pages 77–84, 1958.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language – Version 1.2. Technical report, Yale Center for Computational Vision and Control, 1998.

Juan Pablo Mendoza, Manuela M. Veloso, and Reid G. Simmons. Plan execution monitoring through detection of unmet expectations about action outcomes. In *ICRA*, pages 3247–3252. IEEE, 2015.

Felipe Meneguzzi and Michael Luck. *Declarative Agent Languages and Technologies V: 5th International Workshop, DALT 2007, Honolulu, HI, USA, May 14, 2007, Revised Selected and Invited Papers*, chapter Composing High-Level Plans for Declarative Agent Programming, pages 69–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Felipe Meneguzzi, Yuqing Tang, Katia Sycara, and Simon Parsons. On representing planning domains under uncertainty. In *The Fourth Annual Conference of the International Technology Alliance*, London, UK, 2010.

Felipe Meneguzzi, Yuqing Tang, Katia Sycara, and Simon Parsons. An approach to generate MDPs using HTN representations. In *Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities*, Barcelona, Spain, 2011.

Roberto Micalizio and Pietro Torasso. Diagnosis of multi-agent plans under partial observability. In *18th International Workshop on Principles of Diagnosis (DX07), pages*, pages 346–353. Citeseer, 2007.

Roberto Micalizio and Pietro Torasso. *Multiagent System Technologies: 5th German Conference, MATES 2007, Leipzig, Germany, September 24-26, 2007. Proceedings*, chapter Team Cooperation for Plan Recovery in Multi-agent Systems, pages 170–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

Roberto Micalizio and Pietro Torasso. Plan Diagnosis and Agent Diagnosis in Multi-agent Systems. In Roberto Basili and Maria Teresa Pazienza, editors, *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, September 10-13, 2007, Proceedings*, volume 4733 of *Lecture Notes in Computer Science*, pages 434–446. Springer, 2007.

Roberto Micalizio and Pietro Torasso. Plan Execution and Recovery in Multi Agent Systems for Space Applications, 2008.

Roberto Micalizio and Pietro Torasso. Supervision and diagnosis of joint actions in multi-agent plans. In Lin Padgham, David C. Parkes, Jrg P. Mller, and Simon Parsons, editors, *AAMAS (3)*, pages 1375–1378. IFAAMAS, 2008.

Roberto Micalizio and Pietro Torasso. Exploiting agent diagnosis for plan repair in MAS. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*, pages 1231–1232. IFAAMAS, 2009.

Leora Morgenstern. A First Order Theory of Planning, Knowledge, and Action. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, TARK '86, pages 99–114, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

Robert Morris, Tara Estlin, and Brian Williams. Demonstrating Robotic Autonomy in NASA's Intelligent Systems Project. In *Proceedings of the 8th ESA Workshop on Advance Space Technologies for Robotics and Automation*, ASTRA 2004, 2004.

Karen L. Myers. CPEF: A Continuous Planning and Execution Framework. *AI Magazine*, 20(4):63–69, 1999.

Giuseppe Narzisi, Joshua S. Mincer, Silas Smith, and Bud Mishra. Resilience in the Face of Disaster: Accounting for Varying Disaster Magnitudes, Resource Topologies, and (Sub)Population Distributions in the PLAN C Emergency Planning Tool. In Vladimír Marík, Valeriy Vyatkin, and Armando W. Colombo, editors, *Holonic and Multi-Agent Systems for Manufacturing, Third International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2007, Regensburg, Germany, September 3-5, 2007, Proceedings*, volume 4659 of *Lecture Notes in Computer Science*, pages 433–446. Springer, 2007.

Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 968–973, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

Dana Nau, Okhtay Ilghami, Fusun Yaman, Ugur Kuter, H?ctor Munoz-Avila, Dan Wu, J. William Murdock, and Tsz-Chiu Au. Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20:34–41, 2005.

Dana Nau. Current Trends in Automated Planning. *AI Magazine*, 28(4):43, 2007.

Bernhard Nebel and Jana Koehler. Plan Modification versus Plan Generation: A Complexity-Theoretic Perspective. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93*, pages 1436–1441. Morgan Kaufmann, 1992.

Raz Nissim and Ronen I. Brafman. Multi-agent A* for Parallel and Distributed Systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1265–1266, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

Raz Nissim and Ronen I. Brafman. Distributed Heuristic Forward Search for Multi-agent Planning. *Journal of Artificial Intelligence Research (JAIR)*, 51:293–332, 2014.

Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 1323–1330. IFAAMAS, 2010.

Ingrid Nunes. Improving the design and modularity of bdi agents with capability relationships. In Fabiano Dalpiaz, Jrgen Dix, and M.Birna van Riemsdijk, editors, *Engineering Multi-Agent Systems*, volume 8758 of *Lecture Notes in Computer Science*, pages 58–80. Springer International Publishing, 2014.

Oliver Obst. Using a planner for coordination of multiagent team behavior. In Rafael H. Bordini, Mehdi M. Dastani, Jurgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *Lecture Notes in Computer Science*, pages 90–100. Springer Berlin Heidelberg, 2006.

Jakub Ondráček, Ondřej Vaněk, and Michal Pěchouček. Solving Infrastructure Monitoring Problems with Multiple Heterogeneous Unmanned Aerial Vehicles. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multi-*

agent Systems, AAMAS '15, pages 1597–1605, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

Lin Padgham and Patrick Lambrix. Formalisations of capabilities for BDI-agents. *Autonomous Agents and Multi-Agent Systems*, 10(3):249–271, 2005.

Héctor Palacios and Héctor Geffner. Compiling Uncertainty Away: Solving Conformant Planning Problems using a Classical Planner (Sometimes). In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 900–905, 2006.

Massimo Paolucci, Onn Shehory, Katia Sycara, Dirk Kalp, and Anandeep Pannu. A Planning Component for RETSINA Agents. In NicholasR. Jennings and Yves Lesprance, editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *Lecture Notes in Computer Science*, pages 147–161. Springer Berlin Heidelberg, 2000.

Simon Parsons and Mark Klein. Towards Robust Multi-Agent Systems: Handling Communication Exceptions in Double Auctions. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 1482–1483. IEEE Computer Society, 2004.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17(3):397–431, 2008.

Michal Pěchouček, Simon G. Thompson, Jeremy W. Baxter, Graham S. Horn, Koen Kok, Cor Warmer, Rene Kamphuis, Vladimir Mařík, Pavel Vrba, Kenwood H. Hall, Francisco P. Maturana, Klaus Dorer, and Monique Calisti. Agents in Industry: The Best from the AAMAS 2005 Industry Track. *IEEE Intelligent Systems*, 21(2):86–95, March 2006.

Michal Pěchouček, Martin Rehák, Petr Charvát, Tomas Vlček, and Michal Kolář. Agent-Based Approach to Mass-Oriented Production Planning: Case Study. *IEEE*

Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 37(3):386–395, May 2007.

Damien Pellier, Humbert Fiorino, and Marc Métivier. Planning When Goals Change: A Moving Target Search Approach. In Yves Demazeau, Franco Zambonelli, Juan M. Corchado, and Javier Bajo, editors, *Advances in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection - 12th International Conference, PAAMS 2014, Salamanca, Spain, June 4-6, 2014. Proceedings*, volume 8473 of *Lecture Notes in Computer Science*, pages 231–243. Springer, 2014.

Diego Rodrigues Pereira, Luciano Vargas Gonçalves, Gracaliz Pereira Dimuro, and Antionio Carlos da Rocha Costa. Constructing BDI plans from optimal POMDP policies, with an application to AgentSpeak programming. In *Proc. of Conf. Latinoamerica de Informática, CLEI*, volume 8, pages 240–249, 2008.

Andre H. Pereira, Luis Gustavo Nardin, and Jaime Simao Sichman. Coordination of Agents in the RoboCup Rescue: A Partial Global Approach. *Agent Systems, their Environment and Applications, Workshop and School of*, 0:45–50, 2011.

Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, pages 212–221, Menlo Park, CA, April 2002. AAAI Press.

Ronald P. A. Petrick and Fahiem Bacchus. Reasoning with conditional plans in the presence of incomplete knowledge. In *Proceedings of the ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Information*, pages 96–102, Trento, Italy, June 2003. Università di Trento.

Ronald P. A. Petrick and Fahiem Bacchus. PKS: Knowledge-based planning with incomplete information and sensing. In *Proceedings of the System Demonstration session at ICAPS 2004*, June 2004. http://www-rcf.usc.edu/

Ronald P. A. Petrick and Mary Ellen Foster. Knowledge-level planning for task-based social interaction. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2012)*, Middlesbrough, United Kingdom, December 2012.

Alexander Pokahr, Lars Braubach, Jan Sudeikat, Wolfgang Renz, and Winfried Lamersdorf. Simulation and Implementation of Logistics Systems based on Agent Technology. In *Hamburg International Conference on Logistics (HICL08): Logistics Networks and Nodes*, pages 291–308, 2008.

Martha E. Pollack and Marc Ringuette. Introducing The Tileworld: Exerimentally Evaluating Agent Architectures. Technical Report 489, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, May 1990.

Martha E. Pollack. Plans As Complex Mental Attitudes. In *Intentions in Communication*, pages 77–103. MIT Press, 1990.

John L. Pollock. *Planning Agents*, pages 53–79. Springer Netherlands, Dordrecht, 1999.

Stephen T. Polyak and Austin Tate. Rationale in Planning: Causality, Dependencies, and Decisions. *Knowl. Eng. Rev.*, 13(3):247–262, October 1998.

Thomas Preisler and Wolfgang Renz. Scalability and Robustness Analysis of a Multi-Agent based Self-healing Resource-flow System. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *FedCSIS*, pages 1261–1268, 2012.

Louise Pryor and Gregg Collins. Planning for Contingencies: A Decision-based Approach. *Journal of Artificial Intelligence Research (JAIR)*, 4(1):287–339, May 1996.

Anand S. Rao and Michael P. Georgeff. An Abstract Architecture for Rational Agents. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *3rd International Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pages 439–449, Cambridge, MA, USA, 25-29 October 1992. Morgan Kaufmann. Proceedings.

Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In *Proceedings of The First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

Glen A. Reece and Austin Tate. Synthesizing Protection Monitors from Causal Structure. In Kristian J. Hammond, editor, *AIPS*, pages 146–151. AAAI, 1994.

Glen A. Reece, Austin Tate, David I. Brown, Mark Hoffman, and Rebecca E. Burnard. The PRECiS Environment. In *Proceedings of the National Conference on Artificial*

Intelligence (AAAI-93) ARPA-RL Planning Initiative Workshop, Washington, DC, 1993.

Nico Roos and Cees Witteveen. *KI 2005: Advances in Artificial Intelligence: 28th Annual German Conference on AI, KI 2005, Koblenz, Germany, September 11-14, 2005. Proceedings*, chapter Diagnosis of Plan Execution and the Executing Agent, pages 161–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

Earl David Sacerdoti. *A Structure for Plans and Behavior.* PhD thesis, Stanford, CA, USA, 1975. AAI7605794.

Maria J. Santofimia, Xavier del Toro, Pedro Roncero-Sánchez, Francisco Moya, Miguel A. Martinez, and Juan C. Lopez. A qualitative agent-based approach to power quality monitoring and diagnosis. *Integrated Computer-Aided Engineering*, 17(4):305–319, 2010.

Sebastian Sardina and Lin Padgham. Goals in the Context of BDI Plan Failure and Planning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 7:1–7:8, New York, NY, USA, 2007. ACM.

Sebastian Sardina, Lavindra de Silva, and Lin Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1001–1008, New York, NY, USA, 2006. ACM.

Michael Schillo and Klaus Fischer. Holonic Multiagent Systems. *KI*, 17(4):54, 2003.

Michael Schillo, Hans-Jürgen Bürckert, Klaus Fischer, and Matthias Klusch. Towards a Definition of Robustness for Market-style Open Multi-agent Systems. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, pages 75–76, New York, NY, USA, 2001. ACM.

Marcel Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. In John P. McDermott, editor, *IJCAI*, pages 1039–1046. Morgan Kaufmann, 1987.

Nathan Schurr and Milind Tambe. *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*, chapter Using Multi-Agent Teams to Improve the Training of Incident Commanders, pages 151–166. Birkhäuser Basel, Basel, 2008.

Martijn Schut, Michael J. Wooldridge, and Simon Parsons. *On Partially Observable MDPs and BDI Models*, pages 243–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

Murat Sensoy, Daniele Masato, Timothy J. Norman, Martin J. Kollingbaum, Chris Burnett, Katia P. Sycara, and Jean Oh. Agent Support for Policy-Driven Mission Planning Under Constraints. In *Proactive Assistant Agents, Papers from the 2010 AAAI Fall Symposium, Arlington, Virginia, USA, November 11-13, 2010*, 2010.

Nazaraf Shah, Kuo-Ming Chao, Nick Godwin, Anne James, and Chun-Lung Huang. *Data Engineering Issues in E-Commerce and Services: Second International Workshop, DEECS 2006, San Francisco, CA, USA, June 26, 2006. Proceedings*, chapter A Sentinel Based Exception Diagnosis in Market Based Multi-Agent Systems, pages 258–267. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

Clauirton Siebra and Austin Tate. An Investigation into the Use of Collaborative Concepts for Planning in Disaster Response Coalitions. In *Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, DIS '06, pages 253–258, Washington, DC, USA, 2006. IEEE Computer Society.

Lavindra Silva and Lin Padgham. *AI 2004: Advances in Artificial Intelligence: 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, December 4-6, 2004. Proceedings*, chapter A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning, pages 1167–1173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

Gerardo I. Simari and Simon Parsons. On the Relationship Between MDPs and the BDI Architecture. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1041–1048, New York, NY, USA, 2006. ACM.

Dhirendra Singh, Sebastian Sardia, Lin Padgham, and Stphane Airiau. Learning context conditions for BDI plan selection. In Wiebe van der Hoek, Gal A. Kaminka,

Yves Lesprance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 325–332. IFAAMAS, 2010.

Munindar P. Singh. Know-how. In Anand S. Rao and Michael J. Wooldridge, editors, *Foundations of Rational Agency*, Applied Logic Series, pages 105–132. Kluwer, 1999.

David Šišlák, Martin Rehák, Michal Pěchouček, Milan Rollo, and Dušan Pavlíček. *Software Agent-Based Applications, Platforms and Development Kits*, chapter A-globe: Agent Development Platform with Inaccessibility and Mobility Support, pages 21–46. Birkhäuser Basel, Basel, 2005.

David Šišlák, Premysl Volf, and Michal Pěchouček. Large-scale Agent-based Simulation of Air-traffic. In *Proceedings of the Twentieth European Meeting on Cybernetics and Systems Research*, 2010.

David Sislak, Premysl Volf, Stepan Kopriva, and Michal Pěchouček. AgentFly: Scalable, High-Fidelity Framework for Simulation, Planning and Collision Avoidance of Multiple UAVs, 2012.

Paul R Smart. Holistan revisited: Development of a demonstration scenario for future military coalition operations. March 2008.

David E. Smith and Daniel S. Weld. Conformant Graphplan. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 889–896, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

Ronald D. Snyder, Douglas C. Mackenzie, and Raymond S. Tomlinson. Robustness infrastructure for multi-agent systems. In *In Open Cougaar*, 2004.

Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. HTN Planning with Preferences. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 1790–1797, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

John Sokolowski. Enhanced military decision modeling using a multiagent system approach. In *Proceedings of the Twelfth Conference on Behavior Representation in Modeling and Simulation, Scottsdale, AZ*, pages 179–186. Citeseer, 2003.

Tran Cao Son and Phan Huy Tu. On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 481–491. AAAI Press, 2006.

Tran Cao Son, Phan Huy Tu, Michael Gelfond, and A. Ricardo Morales. Conformant Planning for Domains with Constraints: A New Approach. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, pages 1211–1216. AAAI Press, 2005.

Frédéric Souchon, Christophe Dony, Christelle Urtado, and Sylvain Vauttier. *Software Engineering for Multi-Agent Systems II: Research Issues and Practical Applications*, chapter Improving Exception Handling in Multi-agent Systems, pages 167–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

Katia P. Sycara. Multiagent systems. *AI Magazine*, 19:79–92, 1998.

Kartik Talamadupula, David Smith, William Cushing, and Subbarao Kambhampati. A Theory of Intra-Agent Replanning. *ICAPS 2013 Workshop on Distributed and Multi-Agent Planning (DMAP)*, 2013.

Milind Tambe. Tracking dynamic team activity. In *National Conference on Artificial Intelligence(AAAI96)*, 1996.

Austin Tate, Jeff Dalton, and John Levine. Generation of Multiple Qualitatively Different Plan Options. In Reid G. Simmons, Manuela M. Veloso, and Stephen F. Smith, editors, *AIPS*, pages 27–35. AAAI, 1998.

Austin Tate, John Levine, and Jeff Dalton. Multi-perspective planning – using domain constraints to support the coordinated development of plans. Technical Report AFRL-IF-RS-TR-1999-60, Air Force Research Laboratory, Rome, NY, USA, April 1999.

Austin Tate, John Levine, Peter Jarvis, and Jeff Dalton. Using AI planning technology for army small unit operations. In Steve A. Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pages 379–386. AAAI, 2000.

Austin Tate, Jeff Dalton, Clauirton de Siebra, J. Stuart Aitken, Jeffrey M. Bradshaw, and Andrzej Uszok. Intelligent Agents for Coalition Search and Rescue Task Support. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 1038–1039, 2004.

Austin Tate. Generating Project Networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, pages 888–893, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

Austin Tate. Intelligible AI planning - generating plans represented as a set of constraints. In *Research and Development in Intelligent Systems XVII*, pages 3–13. Springer, 2001.

John Thangarajah, Lin Padgham, and James Harland. Representation and Reasoning for Goals in BDI Agents. In *Proceedings of the Twenty-fifth Australasian Conference on Computer Science - Volume 4*, ACSC '02, pages 259–265, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

John Thangarajah, Sebastian Sardina, and Lin Padgham. Measuring Plan Coverage and Overlap for Agent Reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1049–1056, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 419–437. Springer Berlin Heidelberg, 2003.

Kentaro Toyama and Gregory D. Hager. If at First You Don't Succeed... In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97*, pages 3–9, 1997.

Gabriela Lindemann-Von Trzebiatowski and Ines Miinch. The Role Concept for Agents in MultiAgent Systems. In *Modelling Artificial Societies and Hybrid Orga-*

nizations. Workshop at KI2001, the Joint German/Austrian Conference on Artificial Intelligence, 2001.

Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *In Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA02*, pages 111–124. Springer, 2002.

Adelinde Uhrmacher and William Swartout. Agent-Oriented Simulation. In Mohammad S. Obaidat and Georgios I. Papadimitriou, editors, *Applied System Simulation*, pages 215–239. Springer US, 2003.

Andrzej Uszok, Jeffrey M. Bradshaw, Matthew Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and Stuart Aitken. KAoS Policy Management for Semantic Web Services. *IEEE Intelligent Systems*, 19(4):32–41, July 2004.

Wiebe van der Hoek and Michael Wooldridge. Towards a Logic of Rational Agency. *Logic Journal of the IGPL*, 11(2):135–159, 2003.

Roman van der Krogt and Mathijs de Weerdt. The two faces of plan repair. In *Proceedings of the Sixteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC-04)*, pages 147–154, 2004.

Roman van der Krogt and Mathijs de Weerdt. Coordination through Plan Repair. In *Proceedings of the Fourth Mexican International Conference on Artificial Intelligence (MICAI-05)*, pages 264–274, 2005.

Regis Vincent, Bryan Horling, and Victor Lesser. Experiences in Simulating Multi-Agent Systems Using TAEMS. *The Fourth International Conference on MultiAgent Systems (ICMAS 2000)*, July 2000.

Steven Wark, Andrew Zschorn, Don Perugini, Austin Tate, Patrick Beautement, Jeffrey M Bradshaw, and Niranjan Suri. Dynamic agent systems in the CoAX Binni 2002 experiment. In *Information Fusion, 2003. Proceedings of the Sixth International Conference of*, volume 1, pages 205–212, July 2003.

Max Waters, Lin Padgham, and Sebastian Sardina. Evaluating Coverage Based Intention Selection. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 957–964, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

Gerhard Wickler, Stephen Potter, and Austin Tate. Using I-X Process Panels as Intelligent To-Do Lists for Agent Coordination in Emergency Response. In *International Journal of Intelligent Control and Systems (IJICS), Special Issue on Emergency Management Systems*, 2006.

Gerhard Wickler, Stephen Potter, Austin Tate, Michal Piechouicek, and Eduard Semsch. Planning and Choosing: Augmenting HTN-Based Agents with Mental Attitudes. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Silicon Valley, CA, USA, November 2-5, 2007*, pages 222–228, 2007.

Gerhard Wickler, Michal Pěchouček, Antonin Komenda, Jiri Vokrinek, and Austin Tate. Multi-Agent Planning with Decommitment. In *Proceedings of Knowledge Systems for Coalition Operations (KSCO 2009)*, 2009.

David E. Wilkins. Representation in a Domain-independent Planner. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'83, pages 733–740, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

David E. Wilkins. Recovering From Execution Errors in SIPE. Technical Report 346, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Jan 1985. Supersedes TN 341. SRI Project 8871. Air Force contract F49620-79-C-0188.

David E. Wilkins. Can AI Planners Solve Practical Problems? *Comput. Intell.*, 6(4):232–246, January 1991.

Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative and procedural goals in intelligent agent systems. In *International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufman, 2002.

Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. A Methodology for Agent-oriented Analysis and Design. In *Proceedings of the Third Annual Conference on Autonomous Agents*, AGENTS '99, pages 69–76, New York, NY, USA, 1999. ACM.

Michael J. Wooldridge. Multiagent systems. chapter Intelligent Agents, pages 27–77. MIT Press, Cambridge, MA, USA, 1999.

Michael J. Wooldridge. *Introduction to multiagent systems*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

Shengnan Wu, Larry Shuman, Bopaya Bidanda, Matthew Kelley, Ken Sochats, and Carey Balaban. Agent-based discrete event simulation modeling for disaster responses. In *IIE Annual Conference. Proceedings*, page 1908. Institute of Industrial Engineers-Publisher, 2008.

Haiping Xu, Xiaoqin Zhang, and Rinkesh J. Patel. Developing role-based open multiagent software systems. Technical report, International Journal of Computational Intelligence Theory and Practice (IJCITP, 2007.

Ping Xuan. Techniques for Robust Planning in Degradable Multiagent Systems. In Paul Scerri, Rgis Vincent, and Roger Mailler, editors, *Coordination of Large-Scale Multiagent Systems*, pages 311–340. Springer US, 2006.

Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: A Baseline for Probabilistic Planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, page 352, 2007.

Sung Wook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic Planning via Determinization in Hindsight. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pages 1010–1016. AAAI Press, 2008.

Sung Wook Yoon, Wheeler Ruml, J. Benton, and Minh Binh Do. Improving determinization in hindsight for on-line probabilistic planning. In Ronen I. Brafman, Héctor Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *ICAPS*, pages 209–217. AAAI, 2010.

F.Benjamin Zhan and Xuwei Chen. Agent-Based Modeling and Evacuation Planning. In Daniel Z. Sui, editor, *Geospatial Technologies and Homeland Security*, volume 94 of *The GeoJournal Library*, pages 189–208. Springer Netherlands, 2008.

Reinier Zwitserloot and Maja Pantic. *Agent Frameworks*, volume 1, pages 15–21. 2005.