

Fall 2012

# Search-based system architecture development using a holistic modeling approach

Renzhong Wang

Follow this and additional works at: [http://scholarsmine.mst.edu/doctoral\\_dissertations](http://scholarsmine.mst.edu/doctoral_dissertations)

 Part of the [Systems Engineering Commons](#)

**Department: Engineering Management and Systems Engineering**

## Recommended Citation

Wang, Renzhong, "Search-based system architecture development using a holistic modeling approach" (2012). *Doctoral Dissertations*. 2256.

[http://scholarsmine.mst.edu/doctoral\\_dissertations/2256](http://scholarsmine.mst.edu/doctoral_dissertations/2256)

This Dissertation - Open Access is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact [scholarsmine@mst.edu](mailto:scholarsmine@mst.edu).



SEARCH-BASED SYSTEM ARCHITECTURE DEVELOPMENT  
USING A HOLISTIC MODELING APPROACH

by

RENZHONG WANG

A DISSERTATION

Presented to the Faculty of the Graduate School of the  
MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

In

SYSTEMS ENGINEERING

2012

Approved by  
Cihan H. Dagli, Advisor  
Venkat Allada  
Steven M. Corns  
Ivan G. Guardiola  
Sanjay Madria

© 2012

Renzhong Wang

All Rights Reserved

## ABSTRACT

This dissertation presents an innovative approach to system architecting where search algorithms are used to explore design trade space for good architecture alternatives. Such an approach is achieved by integrating certain model construction, alternative generation, simulation, and assessment processes into a coherent and automated framework. This framework is facilitated by a holistic modeling approach that combines the capabilities of Object Process Methodology (OPM), Colored Petri Net (CPN), and feature model. The resultant holistic model can not only capture the structural, behavioral, and dynamic aspects of a system, allowing simulation and strong analysis methods to be applied, it can also specify the architectural design space. Both object-oriented analysis and design (OOA/D) and domain engineering were exploited to capture design variables and their domains and define architecture generation operations. A fully realized framework (with genetic algorithms as the search algorithm) was developed. Both the proposed framework and its suggested implementation, including the proposed holistic modeling approach and architecture alternative generation operations, are generic. They are targeted at systems that can be specified using object-oriented or process-oriented paradigm. The broad applicability of the proposed approach is demonstrated on two examples. One is the configuration of reconfigurable manufacturing systems (RMSs) under multi-objective optimization and the other is the architecture design of a manned lunar landing system for the Apollo program. The test results show that the proposed approach can cover a huge number of architecture alternatives and support the assessment of several performance measures. A set of quality results was obtained after running the optimization algorithm following the proposed framework.

## ACKNOWLEDGMENTS

I own my gratitude to all those people who made this study possible. First, I would like to gratefully and sincerely thank my advisor, Dr. Cihan H. Dagli, for his guidance, support, understanding, and patience in this research and my entire studies at the Missouri University of Science and Technology. His guidance has made this a thoughtful and rewarding journey. I would also like to extended sincere appreciations to my advisory committee members, Dr. Venkat Allada, Dr. Steven M. Corns, Dr. Ivan G. Guardiola, and Dr. Sanjay Madria, for their time and insightful criticisms that help me to have this research well-structured.

Special thanks to the Engineering Management and Systems Engineering Department for funded my studies. Thanks also extend to the department staff who helped me, especially Ms. Turner who has always been very responsive and helpful. I am also thankful to Ms. Hudgins, in Graduate office, who is very professional and always quick to my requests. Additionally, I am very grateful for the friendship of my colleagues in the Smart Engineering Systems Lab who have always encouraged me and helped me.

I am deeply indebted to my parents, Enming Wang and Xiuyun Hong, for their faith in me and allowing me to be as ambitious as I wanted. As such, they have been sacrificed too much for me throughout my life. I would not haven been where I am today without their constant source of love, concern, support and strength. I am grateful to my sister Yang Wang who has always been supporting me and caring about me. She has been taking my responsibility for taking care of my parents since I was absent most of the time. I would also like to thank all my relatives, friends, and past colleagues, who have encouraged me, helped me, supported me and cared about me.

Finally and most importantly, I would like to give my special thanks to my lovely wife, Beibei Cheng, for the joy she brought to me, and her endless love, support, tolerance, and understanding during the course of my studies.

## TABLE OF CONTENTS

	Page
ABSTRACT .....	iii
ACKNOWLEDGMENTS .....	iv
LIST OF ILLUSTRATIONS .....	viii
LIST OF TABLES .....	x
NOMENCLATURE .....	xi
SECTION	
1. INTRODUCTION .....	1
1.1. NEEDS .....	1
1.2. AIMS AND APPROACHES .....	1
1.3. DISSERTATION SYNOPSIS .....	2
2. LITERATURE REVIEW .....	4
3. OVERVIEW OF RELATED FIELDS AND TECHNOLOGIES .....	10
3.1. OBJECT-ORIENTED MODELING AND DOMAIN ANALYSIS .....	10
3.1.1. Object-Oriented Modeling (OOM) .....	10
3.1.2. Feature Models .....	11
3.2. MODELING LANGUAGES FOR ARCHITECTING .....	13
3.2.1. UML and SysML .....	13
3.2.2. OPM .....	14
3.2.3. Petri Nets .....	15
3.3. RECONFIGURABLE MANUFACTURING SYSTEMS .....	17
4. SEARCH-BASED ARCHITECTURE DEVELOPMENT APPROACH .....	22
4.1. SEARCH-BASED ARCHITECTURE DEVELOPMENT FRAMEWORK ...	22
4.1.1. Requirements Analysis and Design Formulation .....	22
4.1.2. Search-based Architecture Development Process .....	24
4.2. ARCHITECTURE MODELING .....	28
4.2.1. System Design Set .....	28
4.2.2. Object-Oriented Abstraction and Metamodel .....	31
4.2.3. Modeling Process .....	33

4.3. ARCHITECTURE ASSESSMENT.....	38
4.3.1. Architecture Analysis .....	38
4.3.1.1 Evaluation-based approaches .....	39
4.3.1.2 Emulation-based approaches and reasoning about system interactions .....	40
4.3.2. Architecture Selection .....	43
4.3.2.1 A priori approaches.....	45
4.3.2.2 A posteriori approaches .....	46
4.3.2.3 Interactive methods.....	46
4.3.3. Optimization .....	48
5. HOLISTIC MODELING APPROACH .....	54
5.1. DEVELOPING A HOLISTIC MODELING APPROACH .....	54
5.1.1. Strengths and Weaknesses of Some Existing Modeling Languages .....	54
5.1.2. Characteristics of an Ideal Holistic Modeling Language .....	61
5.1.3. Combining UML/SysML, OPM, Petri Nets, and Feature Models.....	61
5.1.3.1 Formal definition of the extended OPM .....	63
5.1.3.2Extend OPM with feature model concepts to capture design space.....	66
5.1.3.3 Supplementing execution semantics of OPM with CPN .....	71
5.1.4. The Roles of CPN in Architecture Modeling and Analyses .....	86
5.2. ARCHITECTURE GENERATION .....	91
5.2.1. Architecture Alternative Generation Operations .....	91
5.2.1.1 Generate element instances.....	92
5.2.1.2 Generate structural variants.....	92
5.2.1.2.1 Add/remove/modify links – operation 1 .....	92
5.2.1.2.2 Add/remove/modify entities - operation 2.....	93
5.2.1.2.3 Side effects handling.....	95
5.2.1.2.4 Advanced operations.....	95
5.2.1.3 Generate full architecture alternative.....	96
5.2.2. Automatic Generation of All Architecture Alternatives .....	97
6. GENERIC IMPLEMENTATION .....	100



6.1. PACKAGE ARCHITECTURE .....	100
6.2. MODULES .....	104
7. APPLICATION DEMONSTRATIONS .....	109
7.1. RECONFIGURABLE MANUFACTURING SYSTEM.....	109
7.1.1. Problem Definition.....	110
7.1.2. Building a Holistic System Model for the RMS .....	113
7.1.3. Building Analysis Models .....	124
7.1.4. Building Optimization Models.....	125
7.1.5. Development of Problem-Specific Modules in Python.....	125
7.1.6. Results and Discussion .....	127
7.2. THE APOLLO PROGRAM (RETROSPECTIVE).....	135
7.2.1. Problem Definition and Analysis.....	135
7.2.2. Architecture Modeling.....	137
7.2.3. Design Space Analysis .....	138
8. CONCLUSION AND FUTURE WORK.....	146
8.1. DISCUSSION .....	146
8.1.1. Comparisons with other Approaches for Solving Similar Problems.....	146
8.1.2. Strengths and Weaknesses.....	147
8.1.3. Scalability of the Proposed Approaches.....	150
8.2. CONCLUSIONS.....	152
8.3. FUTURE WORK.....	153
APPENDICES	
A. MACHINE PROCESSING INFORMATION FOR THE RMS DESIGN EXAMPLE .....	157
B. SELECTED RESULTS OF THE RMS DESIGN EXAMPLE .....	163
C. PYTHON CODE, OUTPUT ARCHIVE FILES, AND OPM AND CPN MODELS ON CD-ROM.....	167
BIBLIOGRAPHY.....	171
VITA .....	188

## LIST OF ILLUSTRATIONS

Figure	Page
3.1. Various Interpretations of the Petri Net Semantics.....	16
3.2. Illustration of a Reconfigurable Manufacturing System [52].....	17
4.1. Framework of the Search-Based Architecture Development Process .....	23
4.2. Elaboration on the Architecture Assessment Process.....	26
4.3. System Design Set ([13]).....	29
4.4. An Example of the Four-Layer Metamodel Hierarchy [27].....	32
4.5. A Simple Taxonomy of Optimization Algorithms ([13]).....	49
5.1. Combining Existing Modeling Languages to Achieve Holistic Modeling.....	62
5.2. A Sample Feature Model ([30]).....	70
5.3. An OPM Model (Created by OPCAT) Extended with Feature Model Concepts to Capture Design Space.....	70
6.1. Components of the Software Implementation .....	101
6.2. Workflow of the Implementation of the Search-Based Architecture Development Framework.....	103
6.3. Illustration of the GUI of the ABCD Simulator.....	106
7.1. Example of a Selected RMS Configuration ([56]) .....	111
7.2. Part to be Produced by the RMS ([57]) .....	113
7.3. OPM/H Model for a RMS - Overview .....	114
7.4. OPM/H Model for a RMS – Zoom-in into Manufacturing Process .....	114
7.5. CPN Model for the RMS .....	115
7.6. CPN Model for the RMS Specified in the ABCD Language .....	116
7.7. Examples of Information Set at the Property Sheet of OPCAT .....	119
7.8. An Alternative Way to Model the RMS - Un-fold RMS.....	122
7.9. An Alternative Way to Model the RMS - Zoom-in into Manufacturing Process..	122
7.10. An Alternative Way to Model the RMS - Zoom-in into OS1 Process .....	123
7.11. Chromosome Encoding of the Design Variables for Solving the RMS Problem ([56]) .....	126
7.12. String Representation of a Solution ([56]).....	126
7.13. The Pareto Front of the Solutions Found Using GA for the RMS Problem .....	129
7.14. Near-Optimal Solutions in the Pareto Front .....	129

7.15. Illustration of One of the Near-Optimal Solution .....	130
7.16. Convergence Curve of the NSGA-II in Solving the RMS Configuration .....	130
7.17. Discrete-Space Representation of the Trajectory of the Manned Lunar Landing System ([1]) .....	136
7.18. OPM/H Class Model Representing the Architecture of the Manned Lunar Landing System Represented.....	139
7.19. CPN Model Used for the Design Space Exploration.....	143
7.20. An Instance Model Representing an Architecture Alternative (LOR System Configuration).....	145

## LIST OF TABLES

Table	Page
5.1. Comparison of UML/SysML, OPM, and Petri Nets.....	56
5.2. Properties of OPM Links .....	65
5.3. Syntax and Semantics of OPM and its Mapping to CPN .....	76
7.1. Attributes of the Machine Object in the OPM/H Model.....	117
7.2. Attributes of the Part Object in the OPM/H Model .....	118
7.3. Parameters Used in the GA.....	128
7.4. Impact of the Number of Part Tokens Used in the CPN Model on the Computation of the Unit Production Time and the Production Rate.....	132
7.5. Statistics from 10 CPN Simulations .....	132
7.6. Final Marking on the Place $M\_Idle$ Obtained from One Simulation Run of the CPN Model for the RMS .....	133
7.7. Final Marking on the Place $P\_Arrived$ Obtained from One Simulation Run of the CPN Model for the RMS .....	134
7.8. Major Modes of the Manned Lunar Landing System and the Corresponding Spacecraft Configuration .....	141
7.9. Dimensions of the Design Space of the Manned Lunar Landing System .....	142
7.10. Summary of Token Values at the Place $A\_Earthlans$ Representing the Architecture Alternatives Discovered.....	143

## NOMENCLATURE

Abbreviation	Description
ABCD	Asynchronous Box Calculus with Data
ABM	Agent based model
ABMS	Agent Based Modeling and Simulation
AHP	Analytic Hierarchy Process
ALF	Action Language for Foundational UML
ANP	Analytical Network Process
ATAM	Architecture Tradeoff Analysis Method
AVM	Architecture Value Map
BBN	Bayesian Belief Network
BPML	Business Process Modeling Language
CM	Command Module
CPN	Colored Petri Net
DF	Direct Flight
DP	Demand Periods
EA	Evolutionary Algorithm
EOR	Earth Orbit Rendezvous
ELECTRE	ELimination and Choice Expressing REality
FAMA	FeAture Model Analyser
FODA	Feature–Oriented Domain Analysis
FOM	Figures of Merit
FUML	Foundational Subset for Executable UML
GA	Genetic Algorithm
GP	Generative Programming
GUI	Graphical User Interface
HC	Hill Climbing
INRS	Improved Net Rewriting System
JDPM	Joint Probability Distribution Method
KISS	Keep It Simple, Stupid

KPA	Key Performance Attributes
LEM	Lunar Excursion Module
LEV	Lunar Excursion Vehicle
LOR	Lunar Orbit Rendezvous
LTDB	Lunar Touchdown Module
MC	Machine Configuration
MOE	Measure of Effectives
MOEA	Multi-Objective Evolutional Algorithm
MOF	Meta-Object Facility
NMS	Number parallel Machines
NP	Number of Parts
NS	Number of Stages
NSL	Number of Stage Locations
NSGA	Nondominated Sorting Genetic Algorithm
OC	Operation cluster
OCL	Object Constraint Language
OOA	Object-Oriented Analysis
OOA/D	Object-Oriented Analysis and Design
OOD	Object-Oriented Design
OOM	Object-Oriented Modeling
OOP	Object-Oriented Programming
OP	Operation
OPL	Object Process Language
OPN	Object Process Network
OPM	Object Process Methodology
OPM/H	OPM for Holistic modeling
OPM/T	OPM with real-time extension
OS	Operation cluster Setup
PG	Precedence Graph
PROMETHEE	Preference Ranking Organization Method for Enrichment, Evaluation
QFD	Quality Function Decomposition

RMS	Reconfigurable Manufacturing System
RMT	Reconfigurable Machine Tool
SA	Simulated Annealing
SBSE	Search -Based Software Engineering
SM	Service Module
SPFL	Single Product Flow Line
SQL	Structured Query Language
STD	Standard Deviation
SysML	System Modeling Language
TOPSIS	Technique for Ordered Preference based on Similarity to Ideal Solution
UML	Unified Modeling Language
URL	Uniform Resource Locator
XMI	XML Metadata Interchange

# **1. INTRODUCTION**

## **1.1. NEEDS**

Computational technologies applied in design, analysis and optimization have flourished in various domain specific disciplines. Well defined methodologies and sophisticated tools have been developed in a large variety of engineering domains to alleviate humans from tedious tasks while increasing design efficiency and quality, for example the computer aided design and computer aided engineering. However, conceptual design in general and architecture design in particular are poorly supported by automated analysis, design and optimization tools. Such design domain is very challenging because: (1) conceiving and designing such systems requires abstract concept formulation and development, (2) the subjects are characterized by ambiguous, intangible, poorly defined, and uncertainty, (3) available implicit or explicit knowledge and experience about the actual system is scarce and the operating environment is entrenched with high degree of uncertainty [1], (4) such design involves multiple knowledge domains, (5) the design space is vast and is difficult to specify due to ambiguity, and (6) transforming information and knowledge from architecture representation to architecture assessment is a field that has not been fully explored.

Traditional architecture design, analysis and development approaches and the modeling, analysis and simulation tools developed for them usually only focus on a single system model or very limited design alternatives. Trade-off studies, as a separate process, are only conducted on simplified system model using partial system information. On the other hand, architecture design space is usually vast since fewer constraints have been identified in this stage of design. In the meantime, architecture design shapes the final form and function of a system. A significant amount of project cost is usually committed at this stage. Hence, architecture design is crucial to the success of the system. Overlooking potential architecture alternatives means loss.

## **1.2. AIMS AND APPROACHES**

This research is aimed at developing a framework with a set of enabling technologies to achieve optimum architecture development through an effective search



process. As the architecture design space is usually vast, such design approach requires automating certain model construction, alternative generation, simulation, and assessment tasks. These tasks should also be integrated into a coherent framework. In order to support such integration and automation, a holistic system model is needed for capturing all relevant design information and supporting architecture analyses. Particularly, the focuses of this research can be summarized as follows

- Identify the tasks needed in a search-based architecture development process and develop a framework to integrate related tasks
- Develop a holistic modeling approach such that the system of interest can be modeled by a holistic model that captures all structural, behavior and dynamic aspects of the system. Such models should not only capture all the design information and variables but also be able to specify the design space.
- Develop an effective approach to generate all architecture alternatives within the design space specified by that holistic system model. Such alternative generation mechanism should be based on the modeling formalisms proposed.
- Identify applicable architecture assessment techniques that can reach rational decisions regarding the selection of architecture alternatives based on the information provided by the architecture model. Identify the required design information and variables that must be captured by such an architecture model.

With such design approach, vast design space can be explored and evaluated before commitment to more detailed design, thus reducing time, cost, and risks and improving design quality.

### **1.3. DISSERTATION SYNOPSIS**

This dissertation is organized as follow:

Section 1, introduction, briefly introduces the motivation of this research.

Section 2, literature review, discusses the application of search-based algorithms in various architecture related problems.

Section 3, overview of related fields and technologies, provides a brief review of some background knowledge needed to develop the approaches proposed in this research

such as object-oriented paradigm, domain analysis, and some related modeling languages. It also briefly introduces the RMS, which will be used as an example to demonstrate the application of the proposed approaches.

Section 4, search-based architecture development framework, presents the proposed architecture development framework along with the discussions of some enabling technologies for each of its components.

Section 5, holistic modeling approach, presents the development of a holistic modeling approach achieved by integrating three modeling formalisms, i.e., OPM, CPN and feature model. A set of architecture variant generation operations is also defined.

Section 6, programming implementation, presents how the proposed approaches are implemented using Python programming language.

Section 7, application demonstration, applies the proposed approach to the design of reconfigurable manufacturing systems and the manned lunar landing system for the Apollo program (retrospective).

Section 8, conclusion and future work, discusses the scalability, strengths and limitations of the proposed approach before concluding the dissertation. It also provides some insights into possible future expansions of the current work.

## 2. LITERATURE REVIEW

This section focuses on reviewing the application of search-based algorithms to the architecture development and its sub-problems. Discussions of other related topics are presented in related sections throughout this dissertation.

Search algorithms have been used widely in different fields of research, such as engineering, business and financial and economic modeling [2]. However, search-based system architecting as a research domain is far from mature and recently there has been an increasing interest in implementing search algorithms to complex system design including architecture design. This review covers the application of search-based algorithms to architecture related problems from a variety of domains. Although many of such applications are either problem specific or domain specific, when studied at the abstract level, they share a lot in common with the system architecture design in general. Therefore studies of these applications may reveal useful inspiration and insights as to how search algorithms can be used in the field of system architecting in general.

A lot of research has been conducted on applying search-base algorithms to software system architecture designs. A software development paradigm known as Generative Programming (GP) is first proposed in the dissertation of Dr. Dipl.-Inf. Krzysztof Czarnecki [3] and later become an active research topic in software engineering [4]. GP is defined in [3] as follows:

Generative Programming (GP) is about designing and implementing software modules which can be combined to generate specialized and highly optimized systems fulfilling specific requirements. The goals are to (a) decrease the conceptual gap between program code and domain concepts (known as achieving high intentionality), (b) achieve high reusability and adaptability, (c) simplify managing many variants of a component, and (d) increase efficiency(both in space and execution time).

GP builds on system-family engineering (also referred to as product-line engineering). It concerns with designing and implementing reusable software for generating specific systems rather than developing each of the specific systems from scratch [3]. It covers a broad range of reusable workproducts (or reusable assets), which include reusable components, requirements, analysis and design models, architectures, patterns, generators, domain-specific languages, frameworks. Particular, it identifies

feature modeling and domain analysis as the main means for specifying design space. Using such an approach, given a system specification, a concrete system can be automatically generated based on a set of reusable components. However, GP focuses on a class of systems within a domain not necessarily exploring all possible variants. Its major application is software systems.

Extensive research has been conducted on a new field emerged in software engineering domain, i.e., the so-called Search-Based Software Engineering (SBSE) [5–8]. SBSE is a collection of a variety of approaches to software engineering in which search-based optimization algorithms are used to address problems in software engineering. The work presented in [6] divides areas where search algorithms are used into four major categories: analysis, design, implementation, and testing. Examples include classifying software production data, project scheduling, static task scheduling related to parallel computing, allocating modules to subsystems, N-version programming, test data generation and generating an integration test order [6]. A more refined classification of software engineering areas to which SBSE has been applied and the various applications within each category are discussed in [8]. Such areas include network protocols, requirements/specifications, design tools and techniques, coding tools and techniques, software/program verification, testing and debugging, distribution, maintenance and enhancement, management, distributed artificial intelligence, and security and protection [8].

Another related study in the software engineering field is the generic programming. Generic programming is a programming style and a set of language mechanisms to achieve program reuse by implementing type-safe polymorphic containers [9]. Generic programming centers around the idea of abstracting from concrete, efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software [10]. Generic programming depends on the decomposition of programs into components which may be developed separately and combined arbitrarily, subject only to well-defined interfaces [11]. However, as summarized in [3] generic programming limits code generation to substituting concrete types for generic type parameters and welding together pre-existing fragments of code in a fixed pattern.

Search-based approaches developed for architecture development in systems engineering field are relatively rare. A Smart Systems Architecting framework is proposed in [12]. It highlights the tasks of applying computational intelligence into architecture trade-off space exploration but provides few implementation details. A generic framework for constructing an evolutionary design model for design of complex systems is presented in [13]. This framework identifies the architecture modeling tasks for various design states and a set of existing technologies applicable to each design task. The resultant design model is described as an evolutionary model that moves a system's design from simple abstract states to more complex and detailed states. However, it presents the framework only. No implementation is developed.

A meta-language for systems architecting called object-process network (OPN) was developed by Koo in [1]. It is a Petri net like executable language that utilizes a small set of linguistic primitives, i.e., objects and processes that transform them. The aim of the language is to support system architects' modeling process by automating certain mechanical communication and computational tasks in architectural reasoning. Koo [1] suggested three usage of OPN in architectural modeling: (1) as a declarative language to specify the space of architectural options, (2) as an imperative language to create architectural option instances and to compute the performance metrics for those instances, and (3) as a simulation language. The rationale behind usage (1) and (2) is an analogue of defining classes and creating instances. Therefore, its variability generation mechanism, like that in OOA/D, is limited to the intra-application variability (i.e., creating object variants only) as pointed out by [3]. It still lacks an explicit mechanism to model both the variations and the related constraints like the one provided by feature models and the domain engineering [3], [14]. Thus, although OPN is effective in creating element instances, it still lacks an effective way to automatically generate the entire architecture as alternatives. Nevertheless, Koo demonstrated that tokens can be used to record the execution trace in a simulation of an OPN model in [1]. Such traces can represent the architecture alternatives discovered. The execution semantics of OPN is based on the function-algebraic model, which supports discrete, continuous, and probabilistic events simulation. Furthermore, the emphasis of the modeling language is for creating computational model. The language is not intuitive to represent static

relationships between system entities. A software environment is developed for the proposed meta-language in [1].

The evolutionary algorithms and other metaheuristic based algorithms such as simulated annealing and tabu search have been broadly applied to many architecture related designs [2]. Most of such applications use no explicit system models or use very simple system description to contain related information. Instead, the idea is to develop problem specific chromosome representations and crossover/mutation operators. For example, the Genetic Algorithm (GA) is applied to software architecture design in [15]. In this work, a complicated chromosome representation is used. Such chromosome is comprised of a list of supergenes following the supergene idea given by [16]. Each of the supergene corresponds to one responsibility in the system. Each responsibility is described by a set of attributes and has a set of responsibilities depend on it. Each responsibility is also associated with a class which implements an interface, belongs to a super class, and communicates with a set of responsibilities through a dispatcher. Accordingly sophisticated mutation operator is defined based on the structure of supergene. The crossover operator is a simple one point crossover that is applied at a random selected supergene. Such type of chromosome encodes the complete information of an architecture model into a chromosome representation. Therefore no extra architectural model is needed. Such chromosome encoding scheme also eliminates the needs to develop additional alternative generation mechanism because mutation and crossover operators can be used to generate alternatives directly. However, the disadvantage of this approach is that its chromosome encoding is rather rigid and cannot generalize well for use in non-software systems. Such approach also assumes a fixed set of responsibilities which may not be the case in other types of systems.

Another problem-specific application of GA in architecture related problem is presented in [17], where GA has been applied to dynamic and multiple criteria web-site optimizations. The purpose is to find the best-possible arrangements (in terms of both combinations and sequences) of a given set of web-objects, such as banners, images, splash screens, leased spots, sounds, and other multimedia objects, based on simultaneous optimization of multiple criteria: (1) download time; (2) visualization; and (3) product association level [17]. Again, no system model is used. The system can be simply

described by a look-up table containing a set of candidate web-objects, each of which is described by a set of attributes such as product name, download time, visualization score, and likelihood that the product will be sold in combination with other products or services [17]. The chromosome representation is simply a sequence of web-objects. The mutation is achieved by swapping two random web-objects within the chromosome. The crossover operator works as follows: select the first  $k$  members of parent 1 as the first  $k$  members of the offspring, where  $k$  is a random number between 0 and the number of web-objects in the chromosome. The remaining members of the offspring come from parent 1 but following the order in which they appear in the parent 2 sequence. The results achieved by Asllani and Lari [16] show that the algorithm provides dynamic and timely solutions independent of the number of objects to be arranged.

System architecting is a broad field comprised of many sub-problems. Studies on solutions to the sub-problems also contribute to the overall body of knowledge of architecture design in general. Rähkä [15] studied many search-based algorithms applied to problems that constitute to sub-problems of software architecture design. These solutions also provide useful insights into application of search-based algorithms in system architecture design in general. These sub-problems studied in [15] include search algorithms in clustering, systems integration, system refactoring, and program transformation. Clustering is a classical problem that is often studied in system architecting as a means to achieve modularity, particularly in software engineering [2], [18], [19]. Systems integration in software engineering [2], [20] is in a way quite similar to module clustering, only now the modules are known, and the order in which they are incorporated to the system is what needs to be decided [15]. Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure [21]. Refactoring is basically a variant of restructuring [22] used in object-oriented system. The key idea here is to redistribute classes, variables, and methods across the class hierarchy in order to facilitate future adaptations and extensions [23–25]. Program transformation enables programming at a higher-level of abstraction, thus increasing maintainability and re-usability [26]. All approaches to transformation share the common principle that they alter the program's syntax without affecting its semantics [2].

This dissertation presents a search-based architecture development framework that integrates architecture modeling, alternative generation, and architecture assessment into a coherence process. A holistic modeling approach is developed to facilitate the implementation of such framework. This modeling approach is achieved by combining the capabilities of OPM, CPN and feature modeling into one holistic representation. The resultant holistic model not only can capture the structure, behavior, and dynamic aspects of a system but can also support simulation and formal model analysis. This holistic modeling approach not only supports the generation of instance models that contain all information needed for architecture specification and analysis but also support the development of a class model that captures the specification of design space (or constraints). An architecture generation mechanism based on the proposed modeling formalism is also developed to support the generation of all architecture alternatives that cover the entire design space. The proposed approaches are implemented using Python with the support of some open source libraries for implementing the CPN and evolutionary algorithms. Two sample projects, the design of RMSs and the architecture design of a manned lunar landing system for the Apollo program (retrospective), are used to demonstrate how to apply the proposed approaches.



### 3. OVERVIEW OF RELATED FIELDS AND TECHNOLOGIES

This section presents a brief review of some technologies related to developing the proposed approaches as well as some background knowledge of the sample problem to be used for demonstrating the application of the proposed approach. The aim is to reach a common understanding of related terminologies and to provide the background and foundation for further discussions in later sections.

#### 3.1. OBJECT-ORIENTED MODELING AND DOMAIN ANALYSIS

**3.1.1. Object-Oriented Modeling (OOM).** OOM is a modeling paradigm originating from computer science, known as object-oriented programming (OOP). OOP uses “objects” as the primary constituents to build a system. An object contains encapsulated data fields and procedures, together with interface, to represent an entity. An object-oriented program is described by the interaction of these objects. Closely related to OOM, are the concepts of Object-Oriented Design (OOD) and Object-Oriented Analysis (OOA). OOD is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis (OOA). There are two major approaches to object-oriented design, class-based approach, where objects are obtained by instantiating classes, and prototype-based approach, where objects are typically obtained by cloning other (prototype) objects. Only the class-based approach is discussed in this dissertation. The basic object-oriented concepts are briefly introduced as follows (biased toward software engineering) [27]:

*An Object* is an entity that has state, attributes and services.

A *Class* describes a set of objects that share the same specifications of features, constraints, and semantics [28].

*Attributes* together represent an object’s static features and state.

*Relationships* include “is\_a” classification relations, “part\_of” assembly relationships, and any “associations” between classes.

*Methods* (services, functions) are the operations that all objects in a class can do.

An *Interface* defines how objects interact with each other. In software engineering, it defines the functions or methods signatures without implementing them.

Such kind of abstractions is so universal that OOM is claimed to be more “natural”. Some key features of OOD include:

*Object/Class*: A *class* defines common properties of a set of objects in terms of what it is and what it can do. A *class* is used to create instances of itself, referred to as *class instances*, or simply *objects*.

*Inheritance*: Inheritance is a process of sharing properties of the higher level object or class [28]. Part of the subclass can be derived (inherited) from the superclass. The subclass can “specialize” the parent class by adding additional attributes and methods or by replacing an inherited attribute or method with another. Multiple inheritance (i.e., multiple different superclasses) is also possible. Inheritance facilitates reuse (part) of class definition by allowing building new class or objects from the base class or super class [28].

*Polymorphism*: Polymorphism allows a name to denote instances of many different classes as long as they are related by some common superclass [29]. Any object denoted by this name is thus able to respond to some common set of operations in different ways [29].

**3.1.2. Feature Models.** Feature models [14], [30] are widely used in software product line engineering. The term feature model first appeared in the Feature–Oriented Domain Analysis (FODA) report [31] and has been an active research topic in software product lines since then.

A feature model represents the information of all possible products of a software product line in terms of features and relationships among them [14]. A feature model defines a hierarchical structure over the set of features of a domain using: (1) relationships between a parent (or compound) feature and its child features (or subfeatures); (2) cross–tree constraints [14]. The root of a feature tree always represents the domain whose features are modeled. A child feature can only appear in a product if its parent feature does. A basic feature model has the following relationships among features:

- *Mandatory*: Mandatory relations connect mandatory features to their parent feature. Mandatory features are always part of the system if their parent feature is part of the system.

- *Optional*: Optional relations connect optional features to their parent feature. Optional features can be optionally included in the system if their parent feature is already in the system.

- *Alternative*: Alternative relations are *exclusive or* relations connecting optional features to their parent feature. Exactly one feature out of a set is part of the system if the parent feature is part of the system.

- *Or*: one or more of children can be included in the system in which its parent feature appears.

Cross-tree constraints between features typically include:

- *Requires*. If a feature A requires a feature B, the inclusion of A in a system implies the inclusion of B in such system.

- *Excludes*. Only one out of a set of features can be part of the system.

The basic feature model has difficulty to express complex concepts. Hence various extensions have been proposed. For example, the cardinality-based feature models [32] extend FODA feature model with multiplicity concepts like the ones used in Unified Modeling Language (UML). Particularly, two types of cardinality exist: feature cardinality and group cardinality as summarized in [14]. Feature cardinality (denoted by  $[n..m]$  with  $n$  and  $m$  as the lower and upper bound respectively) determines the number of instances of the feature that can be part of a product and is a generalization of the original mandatory ( $[1, 1]$ ) and optional feature ( $[0, 1]$ ) [14]. Group cardinality (denoted by  $\langle n..m \rangle$  with  $n$  and  $m$  as the lower and upper bound respectively) determines the number of child features that can be part of a product when its parent feature is selected [14]. More advanced extensions to basic feature models can also be found in literature. Such extensions include adding feature attributes (, which usually contain at least a name, a domain and a value) and complex constraints among attributes and features as summarized in in [14].

In addition, a variety of operations of analysis, tools, paradigms and algorithms have been developed to support automated analysis of feature models. David et al provides an extensive review of the operations developed for automated analysis of feature model in [14].

In order to implement these operators the usual graphical notations of features are mapped to various computational languages such as Propositional Logic, Constraint Programming, Description Logic and other ad-hoc solutions [14]. Once a feature model is transformed into a suitable representation, various off-the-shelf solvers can be applied to analyze a feature model automatically. Such solvers include Constraint Satisfaction Problem solver, Boolean Satisfiability Problem solver, and Binary Decision Diagrams solver, etc.

### **3.2. MODELING LANGUAGES FOR ARCHITECTING**

This section provides a brief review some existing modeling languages that support system specification and/or system analysis. Here the discussion is focused on three languages UML, OPM, and Petri nets. Each of these languages has distinct language design goal and capabilities, along with its own merit. This section briefly review their language features only. A detailed comparison of their strengths and weaknesses in the context of search-based architecture development will be further discussed in Section 5.1.1.

**3.2.1. UML and SysML.** UML [28], [33] is comprehensive language family served as a general-purpose, standardized modeling language for object-oriented analysis and design. It uses a set of diagrams to model a system from multiple views such as requirements view (by use case diagrams), structure view (by class, package diagrams, composite structure, component diagrams etc.), behavior view (by state machine, activity, interaction diagrams, etc.), and implementation view (by deployment diagrams) [34]. An additional textual language, the Object Constraint Language (OCL), is also provided with UML for expressing static consistency constraints on sets of objects and their interrelations. Although UML was initially designed for software developers, its usage has been expanded to many non-software systems due to its popularity and comprehensiveness.

Currently, the semantics of UML language constructs is only defined in a textual, informal way [35]. The syntax of UML is defined by UML metamodel, which is itself a UML class diagram together with OCL-constraints and it defines the context-free as well as context-sensitive syntax of all UML diagram types [35].

Among other capabilities, UML models are often used to serve three purposes: presentation, specification, and documentation. Presentation is the activity of using diagrams for communicating the design ideas with other engineers or stakeholders. Specification involves using UML's prescriptive power to precisely define the system to be built. Documentation involves using UML models as a means to archive designs, requirements or knowledge throughout the development process.

SysML (Systems Modeling Language) [36] is an extension of UML through the profile mechanism of UML. SysML is intended to be a general-purpose modeling language for systems engineering [36]. SysML supports the specification, analysis, design, verification, and validation of a broad range of complex systems [36]. In a manner similar to how UML unified the modeling languages used in the software industry, SysML is intended to unify the diverse modeling languages currently used by systems engineers [36]. It is particularly effective in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis [36]. The language is intended to support multiple processes and methods such as structured, object-oriented, and others, but each methodology may impose additional constraints on how a construct or diagram kind may be used [36].

SysML is a smaller language, compared to UML, in terms of both diagram types and total constructs, as it removes many of UML's software-centric constructs. SysML reuses a subset of UML 2 and provides additional extensions. Seven out of nine diagram types of SysML come from UML. The remaining two, requirements diagrams and parametric diagrams, are achieved through the extension mechanisms of UML.

**3.2.2. OPM.** OPM developed by Dori [37] is a general-purpose modeling language with a single model formalism and a small set of symbols consists of objects, processes and a variety type of relational links connecting them. OPM can be used to specify both the structural and behavioral aspects of a system [38].

The building blocks of OPM are entities (things and states) and links. A thing is a generalization of an object and a process. Objects are things that exist and they may have states. States are lower level entities since they reside in objects. At any particular point in time, an object can be exactly in one state, and object states are changed through processes [39]. Processes are things that transform objects. Links can be structural or

procedural. Structural links express static (persistent, long-term relations) relations between pairs of objects or process [40]. Procedural links, on the other hand, connect entities to describe the behavior of a system [40]. The behavior of a system is manifested in three major ways: (1) processes transform (generating, consuming or affecting) objects; (2) objects can enable process without being transformed by them; and (3) things can trigger events that invoke processes [41].

OPM manages system complexity through three refinement/abstraction mechanism: (1) in-zooming/out-zooming exposes/hide the inner details of a thing within its frame; (2) unfolding/folding is used for refining/abstracting the structural hierarchy of a thing; and (3) state expressing/suppressing expose/hides the state of an object [42]. These mechanisms enable OPM to recursively specify a system to any desired level of detail without losing legibility and comprehension of the resulting specification [40].

OPM has bimodal representation. One is graphic and the other is textual. Both are semantically equivalent. The graphical representation, known as Object-Process Diagram (OPD), uses graphical syntax with each OPM element being denoted by a symbol. The textual representation, known as Object-Process Language (OPL), specifies the same OPM model in a subset of English, enabling direct mapping between the graphic and the textual representations [13]. OPL is a dual-purpose language, oriented towards both humans and machines [41].

The known tools that support OPM model development are OPCAT [43] and Systematica. Features of OPCAT include: animated simulation of the model, automatic generation of OPL from OPD or the reverse, code generation (Java, SQL), UML diagram generation, and automatic document generation

**3.2.3. Petri Nets.** A Petri net [44], [45] is a mathematical modeling language for discrete event system modeling and simulation. A Petri net is a directed bi-partite graph consists of places and transitions and directed arcs that connect a place to a transition or vice versa. A place can represent the state of an object in the system being modeled. Place can store tokens which represent objects in the system. The distribution of tokens over the places collectively marks the state of the system. With the use of tokens to mark the state of a system, Petri nets can captures the dynamic aspects of a system. Transitions represent the actions of a system. When certain conditions hold, a transition will fire,

causing a change in the placement of tokens and thus the change of system states. The firing of transition is nondeterministic, i.e., when multiple transitions are enabled, anyone (and only one) of them may fire. Furthermore multiple tokens may be present anywhere at in the net at the same time. Therefore Petri nets are well suited for modeling the concurrent behavior of distributed systems.

A Petri net can be viewed from two levels. In macro view A Petri net can be interpreted as a state machine. With the movements of tokens from places to places, the system undergoes a series of state transitions. This is the perspective to understand UML/SysML State Machine. In micro view, a Petri net can be seen as a condition/event graph, where places are conditions (availability of certain object or an object being at certain state) and transitions are events. A transition is fired means an event occurs. It can only occur if all conditions for the event hold. Such perspective is usually used in behavior analysis. Such condition/event/effects semantics can also be interpreted input/process/output according to Carlsen [46], who classifies Petri net as a transformational model language. These interpretations of the Petri net semantics are summarized in Figure 3.1.

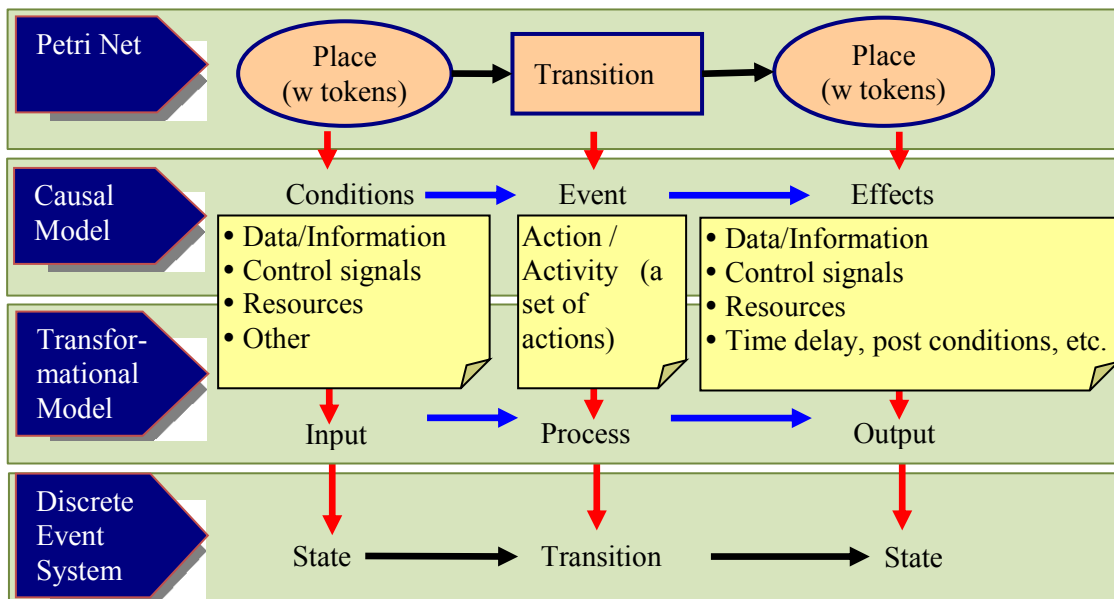


Figure 3.1. Various Interpretations of the Petri Net Semantics

The Petri net is named after Carl A. Petri and was first introduced in his Ph.D. dissertation [47]. It then has since be extensively studied and extended. Through 50 years development, there are several variants of Petri net being developed, for example CPNs, which allow tokens to be typed, timed Petri nets, which introduce time concepts into transition, stochastic Petri nets, which add nondeterministic time through adjustable randomness of the transitions, and Object-oriented Petri nets, which support object-oriented modeling, to name a few. The Petri net and its many variants have been applied to a wide range of applications, such as workflow management, concurrent programming, distributed computing systems, manufacturing system design, and many others [48], [49].

### 3.3. RECONFIGURABLE MANUFACTURING SYSTEMS

A reconfigurable manufacturing system (RMS) is one designed at the outset for rapid change in its structure, as well as its hardware and software components, in order to quickly adjust its production capacity and functionality within a part family in response to sudden market changes or intrinsic system change [50]. A schematic diagram [51] of a RMS is shown in Figure 3.2.

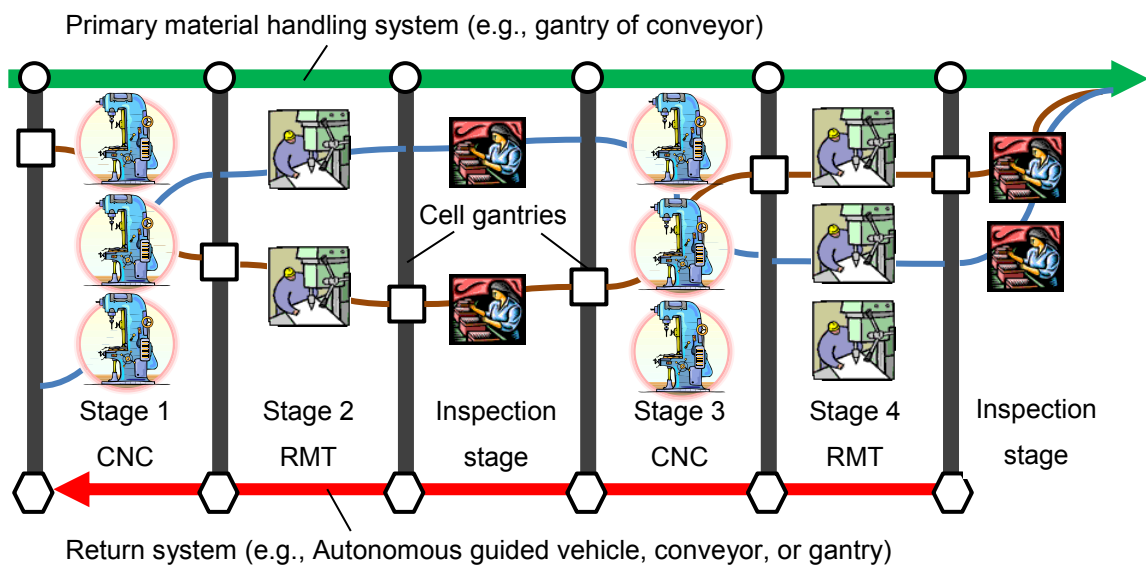


Figure 3.2. Illustration of a Reconfigurable Manufacturing System [51]



RMS is a new manufacturing paradigm that attempts to combine the high throughput of dedicated manufacturing lines with the flexibility of flexible manufacturing systems and react to changes quickly and efficiently [52]. Instead of providing a general flexibility through the use of equipment with built-in high functionality, as in flexible manufacturing systems, RMSs provide customized flexibility through scalability and reconfiguration as needed when needed to meet market requirements [53].

RMS is marked by six core reconfigurable characteristics as summarized in [54]: customization (flexibility limited to part family), convertibility (design for functionality changes), scalability (design for capacity changes), modularity (components are modular), integrability (interfaces for rapid integration), and diagnosability (design for easy diagnostics)

There are many aspects of a RMS configuration. Roughly speaking, a RMS configuration includes system level configuration (such as arrangement of machines and facilities) and machine level configuration (such as machine setup, programming, and machine tool configurations). This dissertation is concerned with the system-level configurations of RMS in a metal-cutting industry.

A huge variety of techniques have been applied to solve the RMS configuration problems. For example, Youssef and H. ElMaraghy [55], [56] developed an approach for optimizing the capital cost of RMS configurations with multiple aspects using GA. This approach can be used to find optimum configuration for a multi-product, flow-line type RMS with identical machines in each production stage. The various aspects of the RMS configurations being considered include arrangement of machines (number of stages and number of parallel machines per stage), equipment selection (machine type and corresponding machine configuration for each stage) and assignment of operations (operation clusters assigned to each stage corresponding to each part type) [55]. A novel, real-coded chromosome representation is proposed. Such chromosome encoding scheme can guarantee the feasibility of the alternatives generated thus making the algorithm efficient. This problem has been adopted as an example and solved using the approach proposed in this dissertation. The details are presented in Section 7.1.

Dou et al. [57] developed a graph theory-based approach to single product flow line (SPFL) optimization problem with small-to-medium size. Such approach is able to

find  $p$  economical and diversified flow-lines which include the optimal and  $p - 1$  near optimal solutions. A machine graph is developed to represent the RMS. The full topological sorting and graph augmentation procedures are developed to derive a combined machine graph from the operation precedence graph of a specific product [57]. In such graph, each node represents a feasible workstation. A directed arc connecting nodes represents the precedence of workstations in accordance with operation assignments. For a given operation sequence, the problem of finding the minimal cost flow-line can be modeled as a shortest path problem on the machine graph associated with the operation sequence. The proposed search algorithm approach is divided into two stages. The first stage is to find the optimal and  $K - 1$  suboptimal configurations by solving a constrained  $K$ -shortest paths problem on a combined machine graph derived from the specified operation precedence graph. The second stage is to find  $p$  distinctive ones out of  $K$  configurations using the algorithms for  $p$ -dispersion problem [57]. The experimental results showed that this approach performs well for small-to-medium size problems of configuration generation. Further development is needed for the approach to scale up to large size problems and to support multi-objective optimizations for multiple Demand Periods (DPs).

Tang et al. [58] develop an approach to RMS configurations that considers the reconfiguration process of a RMS as a network of potential activities and configurations. Then a shortest path graph-searching strategy is applied to find the best configuration. A generic reconfigurable object model is developed to capture necessary information for all levels of objects in the RMS. A reconfigurable object is an object whose structure and state can be modified by a set of actions to realize changes in its performance [58]. Particularly, A reconfigurable object consists of the following elements: member objects (components of a reconfigurable object), states (the current condition of an object, including relationships between its member objects and their conditions), constraints (defines the domain of a state variable), performance metrics (measures for some functionality that an object possesses), set of reconfigurable actions, mapping functions (relationship between the states and the performance of the object), and rules (heuristic knowledge and expertise that assist the derivation of a reconfiguration plan). An Artificial Intelligence-based computer-aided reconfiguration planning framework has

been developed in order to derive reconfiguration plans for a RMS and reconfigurable hardware in the system [58]. The A\* algorithm and a genetic algorithm are employed to perform the search for the reconfiguration plan. Case studies in planning a RMT and a RMS are conducted and the results show that efficient plans are generated in both situations [58].

The Petri net and its variants such as timed Petri nets, stochastic Petri nets, and object-oriented Petri nets have also been applied to RMS configuration [59–62]. The use of Petri net allows using simulation to gain insights into various performance metrics of a RMS. Li et al. [63] developed an approach that uses rapidly reconfiguring Petri net models for RMS design. An improved net rewriting systems (INRS) is developed to achieve such rapid configuration. Such INRS can implement dynamical adjustments to the structure of a Petri net model and maintain its important behavioral properties, i.e., liveness, boundedness (or safeness), and reversibility. Using such approach, changes in a RMS configuration adjusted with production demands can be rapidly formalized into graph rewriting rules of an INRS [63]. Subsequently, by applying these rewriting rules, the existing Petri net model can be reconfigured rapidly into a new one for the RMS with a new configuration [63]. Validity of the resulting Petri net model can be guaranteed naturally throughout the reconfiguration process. The proposed approach is applied to a reconfigurable manufacturing cell. The results showed that the proposed method can generate configuration solution in a rapid and successive manner, without requiring verification [63]. However, such model provides a description of the RMS and valid its configuration only. Little performance metrics can be derived due to the basic Petri net model used. A similar work that uses hierarchical Petri and INRS for supervisory control of reconfigurable manufacturing systems model is presented in [60].

Cai and Yan [59] developed an approach that use timed reconfigurable Petri nets to model RMS. In this work, each machine or equipment in the RMS is modeled with an object-like subnet. In each subnet, a set of states and transitions are used to model the operations of the machine or equipment. For example, the states can be idle, ready, preparing, loading, processing, and unloading. Each subnet also has a number of “message” places to receive or to send information regarding the operation requests or responses. The whole RMS system is composed of a number of such connected subnets

representing machines or equipment. The parts are represented by tokens of the Petri net. Features of the parts to be processed are encoded in the color set of related tokens. With time associated each transition, such Petri net model for the RMS can provide, through simulations, a variety of performance measures such as completion time of a job, average throughput for a part, and resource utilization. Given a new configuration, a new Petri net model will be generated based on the modification of the previous model. There are a number of similar works that use various object-oriented Petri nets to build a similar model for the RMS [62], [61].

Note that these Petri net-based RMS models primarily serve as analysis models only. The purposes are to derive performance measures or to validate the configuration. A dedicated optimization process is still needed if there are a large number of alternatives to be evaluated.

## 4. SEARCH-BASED ARCHITECTURE DEVELOPMENT APPROACH

This section first presents the search-based architecture development framework. Then the guidelines and concerns in implementing two of its components, architecture modeling and assessments, are further discussed. The architecture modeling section discusses what to be modeled, how to use abstraction to extract necessary information and how to systematically develop a system model and define its design space. The architecture assessment component is presented in three sub-sections: architecture analysis, selection and optimization. A set of applicable technologies is also identified, compared, and discussed for each components of the framework.

### 4.1. SEARCH-BASED ARCHITECTURE DEVELOPMENT FRAMEWORK

The four distinctive tasks in search-based architecture development are:

- Developing an architectural model,
- Generating architecture instances,
- Assessing architectural instances,
- Validating design and/or further refining design.

Figure 4.1 depicts these processes using an OPD.

**4.1.1. Requirements Analysis and Design Formulation.** The architecture development cycle is always preceded by a requirements analysis process. Alfariis [13] suggested using the four categories of requirements developed by Buede [64] in system design. Such categories are input/output, technology and system-wide, tradeoff, and test. These types of requirements are adapted and expanded to encompass a set of tasks together called design formulation in this dissertation. A design formulation includes detailed design concepts, constraints, and plans to guide the architecture development process. More specifically, the design formulation contains the following components: input/output, context and boundary, system function breakdown, constraints, performance metrics, tradeoff, and plan. The details of each are described in the following sections:

- *Input/Output.* Input/output include inputs, outputs, and interfaces of the system with its external environment.

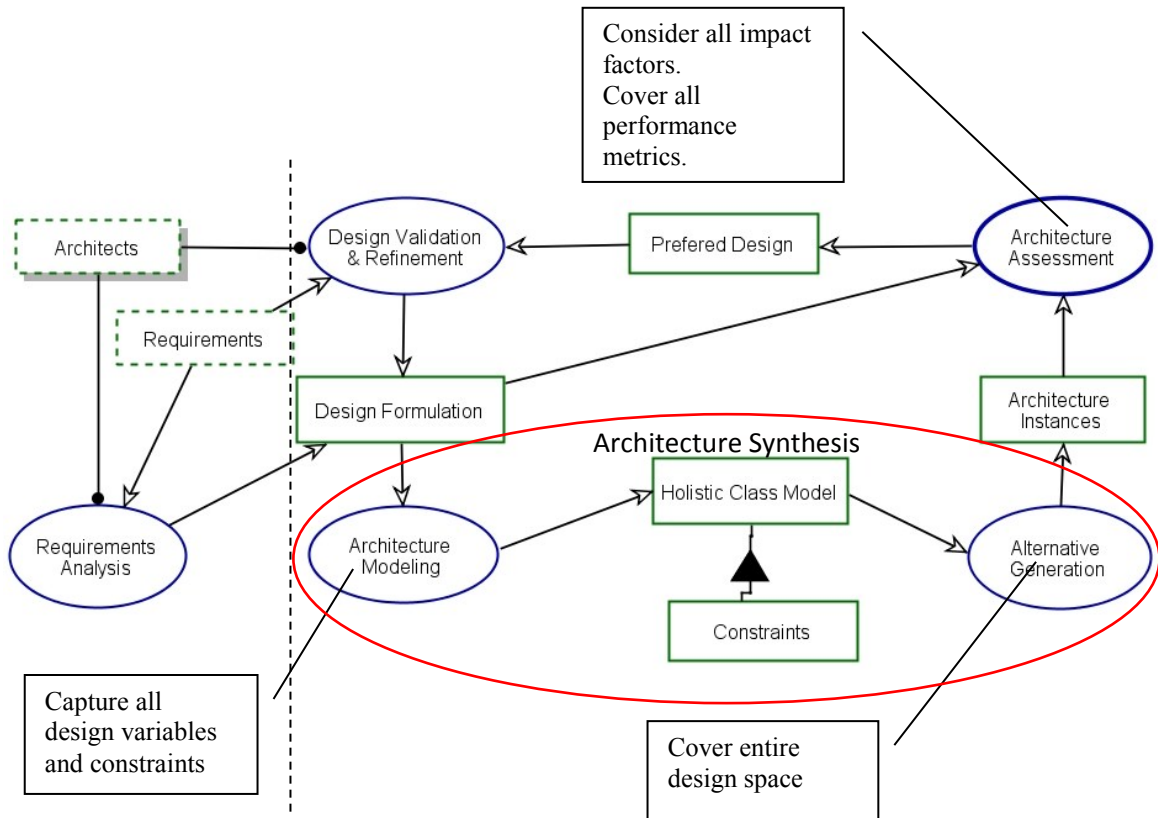


Figure 4.1. Framework of the Search-Based Architecture Development Process

- *Context and Boundary.* The context is the set of entities that interact with systems through their external interfaces. These entities can impact the system through input. They can be impacted by its outputs. The boundary identifies the scope of the problem to be solved.
- *System Function Breakdown.* System function breakdown is a set of functional relationships regulating both the reception and delivery of inputs and outputs. Functional requirements do not convey any requirements with regards to the technology being used, or the process followed in the design [65].
- *Performance Metrics.* Performance metrics include both stakeholder specified and architect identified Key Performance Attributes (KPA). They measure both the quality of the services provided by the system function and the

outputs generated by the system. The KPA can be decomposed into Measure Of Effectives (MOE) [66]. MOE can in turn be decomposed into Figures Of Merit (FOM). For any design under consideration it is necessary to be able to estimate or measure the values of these FOMs [65].

- *Constraints*. Constraints include recourse, budgets, schedule, and various other types of limitations or restrictions. One particular type of constrains is technology constrains. The technology requirement consists principally of limitations specified by the customer on the technologies available to build the system [65].
- *Trade-off Requirements*. Trade-off requirements specify the nature of trade-offs among input/output, system's technologies, and systems requirements. Trade-off requirements will make the actual system selection based on the priorities of the customer [65].
- *Plans*. Plans include various tasks such as choosing appropriate analysis, decision, and optimization techniques to be used in the architecture assessment, prioritizing the objectives to be addressed, and formulating a general concept that guides the problem solving.
- *Architecture/Design Patterns*: An architect may choose to apply architecture or design patterns to improve design efficiency. Architecture or design patterns are descriptions, best practices, or templates for how to solve a problem that can be used in many different situations. In software engineering, design patterns are defined as general, reusable solutions to a commonly occurring problem, within a given context, in software design [67].

Note that the list of elements in the design formulation identified above is not intended to be complete. The architect can either develop additional one or use a subset of this list according to both the problem to be solved and the current design phase.

**4.1.2. Search-based Architecture Development Process.** Once the requirements have been analyzed, the architecture synthesis can proceed. The architecture synthesis includes both architecture modeling and alternative generations. A generative class model that can describe a collection of systems is first developed. A generative class model should not only encode the design knowledge but also capture all of the design variables,

along and their domains. Such requirements usually necessitate a holistic model that can capture all of the structural, behavioral and dynamic aspects of a system as well as constraints. Moreover, such system model may also need to support simulation and system analysis, which are very useful in both system assessment and verification / validation. Then, an architecture generative mechanism is applied to generate all of the architecture alternatives within the design space specified. Next, the architecture assessment process can proceed with the following activities:

- Analyze the behavior of the generated architecture alternatives for verification or validation.
- Derive the performance metrics of the generated architecture alternatives using analysis models or through simulation;
- Search for the best architecture alternative(s) using an appropriate optimization algorithm.
- Making decisions regarding the preference of one or a set of architecture alternatives based on the evaluation of multiple objectives

The architecture assessment process is represented as an aggregated process in Figure 4.1. Its details are exposed in Figure 4.2. The optimization as a search process should be capable of covering the entire solution space. Since the entire architecture alternative space is usually vast, it is not necessary to generate all the possible alternatives in one step. Rather, the search should be guided by the optimization process. Therefore, only a small set of architecture instances are generated and assessed in each iteration given an iterative optimization algorithm is used. Accordingly, there will be a tight coupling between the architecture generation process and the architecture assessment process. This architecture assessment process should consider all performance metrics of interest, covering all factors impacting them so as to yield unbiased results. The solution from the optimization is subject to verification and validation to ensure the selected architecture alternative(s) can

- Conform to the constraints set in the requirements,
- Perform the intended functionality,
- Generate desired behavior, and
- Satisfy the performance requirements.



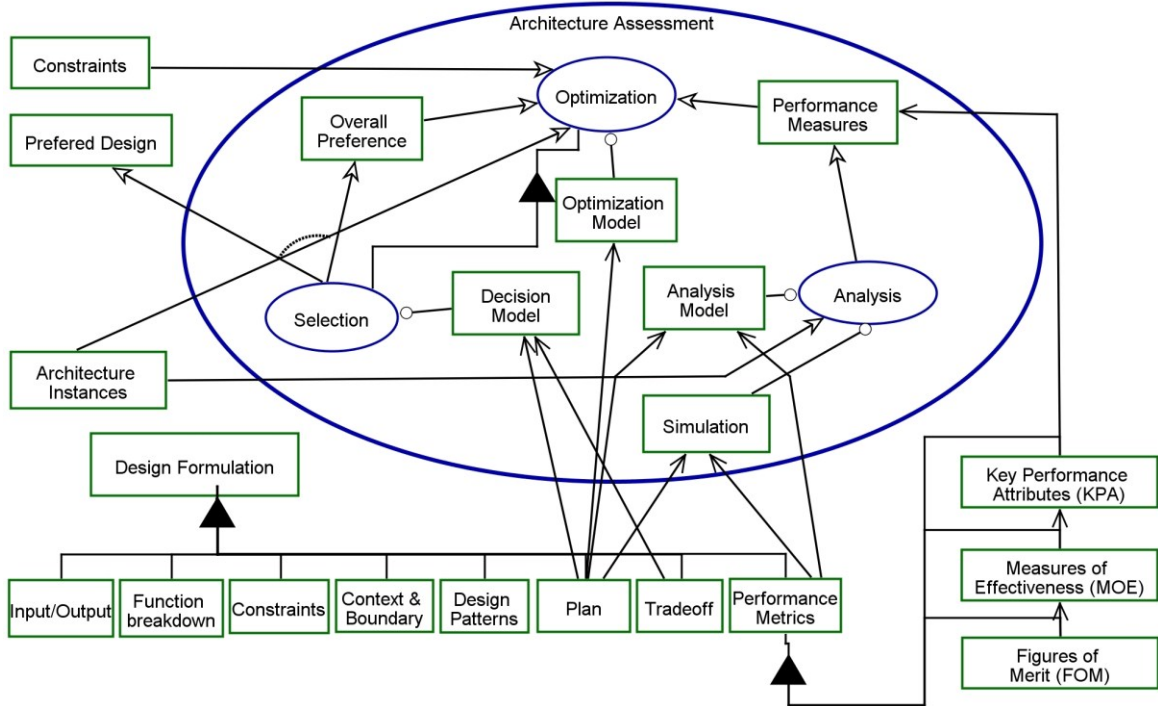


Figure 4.2. Elaboration on the Architecture Assessment Process

Additionally, both the emergent behavior and side effects need to be examined and interpreted for undesired results. The architecture can be further refined with a refinement plan if necessary. This refinement plan can include either the entire or subset of elements in the design formulations, as discussed in the requirements analysis phase. Once a refinement plan is made, another round of the design cycle can proceed. This entire development process is intended to proceed automatically as the design space might be vast. However, it is crucial for human experts to intervene and guide the requirements analysis process, the design validation, and the refinement process. The role of human experts is illustrated using the Agent link in the OPD in Figure 4.1.

As illustrated in Figure 4.2, three key tasks exist in the architecture assessment process: analysis, decision making, and optimization. Each is facilitated by a specific type of model commonly used in engineering design. The following discussion focuses on the objectives and inputs/outputs of these models. Section 4.3 focuses in detail on the techniques available for each type of model.

The analysis process derives the behavioral properties and/or performance measures from a system model using various analysis methods (or models in general) and/or simulations. The input to the analysis model is extracted from the information captured in the system model. Such information can include the structural properties, the behavioral properties, the numerical properties, the relationships between these properties, the interactions between system components, the transitions of system states and more. Depending on the needs of a specific analysis model used, such input data may be subject to a preprocessing process. Both performance measures and behavior properties can be output from an analysis model. Depending on the modeling language used, a system model can sometime double as an analysis model. For example, a Petri net model can be used as both a system model to describe a system and as an analysis model to reason the behavior of a system and simulate the behavior. The reason that a Petri net model can play these dual roles is that Petri nets have rigorous mathematical definitions and they can precisely model the states of a system and, under what conditions, transitions between these states will happen. On the other hand, a design problem usually involves multiple domains. Each domain can develop one or more analysis models. These models range in their required input, type and amount of information, domain of outputs, and degree of accuracy.

The selection process is facilitated by a decision-making model, which is used in conjunction with the optimization model to select good designs that constitute a desired trade-off between conflicting objectives. Various performance measures output from multiple analysis models, expressed in an  $n$ -dimensional (with “ $n$ ” being the number of design objectives), provide the input to a decision model. The output of the decision model is the preference for each solution.

Many real-world optimization problems involve the simultaneous optimization of several incommensurable and often competing objectives. For nontrivial multi-objective problems, there is no single optimal solution, but rather a set of alternative solutions. These solutions, known as Pareto-optimal solutions, are optimal in the wider sense that no other solutions in the search space are superior to them when all objectives are considered [68]. In such cases, decisions have to be made in the presence of trade-offs between conflicting objectives. Based on the system and the design objectives to be

evaluated, the efficiency of the decision tools, and the decision maker's preference, this selection can be an automatic, semi-automatic, or even a manual process (given that there are only a very limited number of solutions to be evaluated).

The optimization process is primarily a search process in the search-based architecture development process. Multi-objective optimization involves two search spaces, the decision variable space and the objective or criterion space. Although the two spaces are related by unique mapping between them, often the mapping is nonlinear and the properties of the two search space are not similar [69]. The design variable space, comprised of architecture instances, is discrete in nature and usually is subject to certain constraints. The choice of an appropriate search algorithm depends on several factors, including the nature of design variable space (e.g. linear or nonlinear, continue or discrete, deterministic or scholastic, convex or nonconvex, etc.), the nature of constraints, the interaction both between design variables and between design objectives, the efficiency of the search algorithm and their ability to found global optima, knowledge of the system and the objectives. The input to the optimization model is a set of values evaluated according to the objective functions. The output of the optimization model is either one or a set of architecture instances.

## 4.2. ARCHITECTURE MODELING

This section provides special guidelines with respect to architecture modeling. The emphasis is on the special needs for an architecture model to support automatic design space exploration. It structures the landscape and identifies regions of related topics for later sections of this dissertation.

**4.2.1. System Design Set.** Alfaris [13] formalizes the tasks in architecture modeling as a design set. According to [13], the system design set  $S$  includes related components ( $S_c$ ) and a structure ( $S_s$ ). The structure ( $S_s$ ) allows components to interact with each other through interfaces ( $S_i$ ). Together,  $S_c$ ,  $S_s$  and  $S_i$  comprise the system's form ( $S_f$ ). This form executes certain system Behaviors ( $B_s$ ). These behaviors include both anticipated behaviors ( $B_a$ ) and emergent behaviors ( $B_e$ ) that should enable system's functions ( $F_s$ ). The combination of  $S_f$  and  $B_a$  defines the system's architecture ( $S_a$ ). In the context of working with a set of architecture alternatives, as in the search-based

architecture development framework, an additional component (constraints  $C$ ) needs to be identified. Constraints are conditions, or restrictions, attached to the constrained elements for the purpose of declaring some additional, semantic information. A constraint is an assertion that indicates what restrictions must be satisfied by a correct design [28]. As such, constraints can be represented as Boolean expressions. They can specify the range of possible values for any design elements and, therefore, can be used to define design options. Both the elements in a design set and their relationships are depicted in Figure 4.3 using OPD.

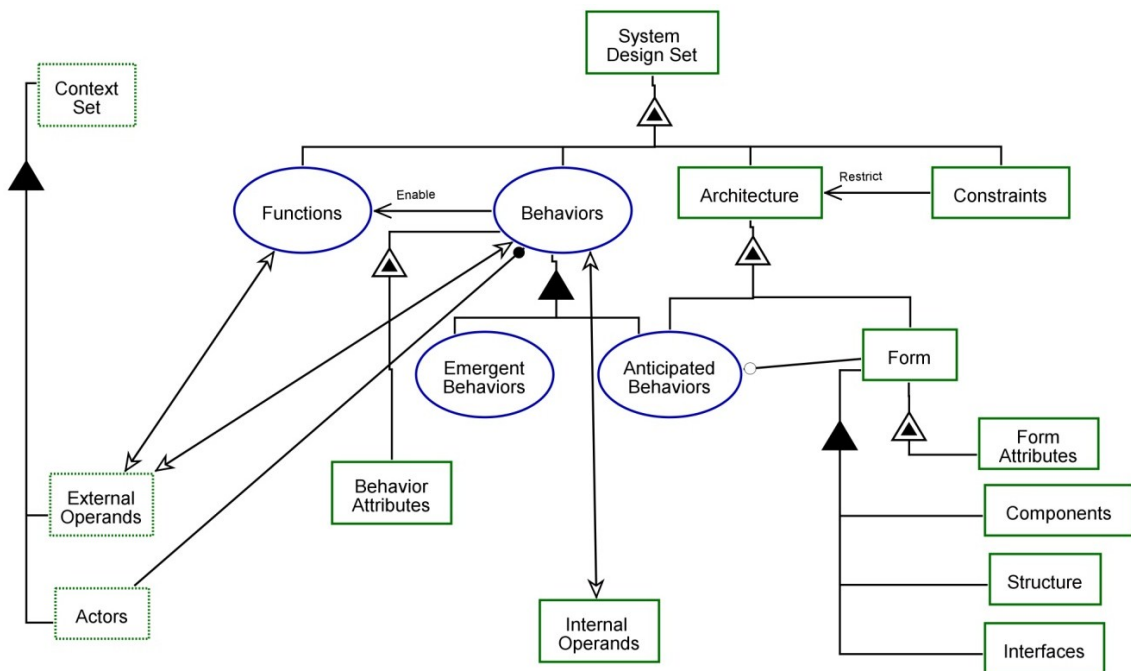


Figure 4.3. System Design Set ([13])

The design set Figure 4.3 summarizes the components in a system architecture. A system architecture, however, has three major aspects that are more relevant to system analysis or architecture reasoning. These aspects are structure, dynamics, and behavior. A brief survey defining each is provided in the following discussion.

Structure in UML [33] refers to “a composition of interconnected elements, representing run-time instances collaborating over communications links to achieve some common objectives describes the assembly of components within a system”. Dori [37] defines structure as “pertaining to the relatively fixed, non-transient, long-term relationship that exists among components or parts of the system”. When explicitly considering time in the definition, structure has also been viewed as a snapshot [37]. Note that, in Alfari’s [13] definition of the system design set, the concept of structure is used somewhat narrowly. It refers only to the composition of system components without including those components and their interfaces. The concept of structure, in many system modeling contexts, is referred to as the collection of composition, components, and interfaces, which is defined as form in [13]. In this dissertation, the term, structure, is used broadly and it is equivalent to the form defined in Alfari [13].

Dynamic aspects describe the changes of a system along time during operation, together with the causes and effects of these changes. The concepts of both states and transitions are often used to describe the dynamic aspects of a system. A state is defined as “a situation of position at which the object can exist for a period of time” in [37]. According to UML [33], “a state models a situation during which some (usually implicit) invariant condition holds”. A transition, on the other hand, describes the switch between states. It describes the transit aspect of a system in contrast with the static aspect. A transition is, therefore, often associated with action or process that transforms system. System model without dynamic aspects cannot precisely describe the state of a system at a particular point of time, or can only provide a snapshot of the system at a particular point of time but cannot describe how and why the system changes over time.

Behavior of a system can be viewed as the collective effects (or consequences, outcomes) of the actions and interactions of system components [33]. This view emphasizes behavior’s association with objects. On the other hand, “A behavior describes how the states of these objects, as reflected by their structural features, change over time” according to UML [33]. As such, the system dynamics discussed above provides a way to describe the behavior of a system. The aim of a system design is to achieve both the desired behaviors that are outputs of functions and certain desired properties while both predicting and limiting undesired behaviors [13]. Both anticipated and emergent

behaviors need to be estimated and constrained during system design in order to prevent undesired behaviors.

In summary, a system architecture is “the overall system’s structure-behavior combination, which enables it to attain its function while embodying the architect’s concept” [37].

**4.2.2. Object-Oriented Abstraction and Metamodel.** All information needed by the analysis models discussed in Section 4.1.2 should be captured in the architecture model and/or its associated constraints. An effective way to do so is by using abstraction appropriately to extract the needed information. Abstraction captures only those details about an object that are relevant to the current perspective [70]. Abstraction applies to every aspect of modeling. Abstraction is defined as “a concept or idea not associated with any specific instance” in [71]. Therefore, the results of abstraction are concepts. The easiest, most natural way to describe a concept is to list its properties [3]. According to Czarnecki’ study [3], concepts can be regarded as natural modeling elements. Therefore, concepts are directly related to classes in object-orientation (especially the classical object model). The concept of object is such a fundamental abstraction that it can cover virtually any entities. As a result, it is a more natural way to represent things.

Class in OOM is a construct for defining objects. In UML, class (in MOF level) is a universal way to define any entities, including objects, procedures (actions), and relationships, except for atomic attributes. With such capabilities, the abstract syntax of UML diagrams can be defined by UML itself. In another word, the metamodel of UML is itself a UML class diagram, together with OCL-constraints. It defines the context-free as well as context-sensitive syntax of all UML diagram types [35]. Figure 4.4 illustrates the metamodeling concepts used in UML [28]. A model that is instantiated from a metamodel can, in turn, be used as a metamodel for a lower level model in a recursive manner. A model typically contains model elements. These are created by instantiating model elements from a metamodel (i.e., metamodel elements). MOF level class is a metaclass known as Element in [28]. It is an abstract metaclass with no superclass used as the common superclass for all metaclasses. MOF defines all metamodel (UML) constructs using a quad-fold Element {attributes, associations, constraints, and operations}, along with textual semantics defined for each element within the quad-fold.

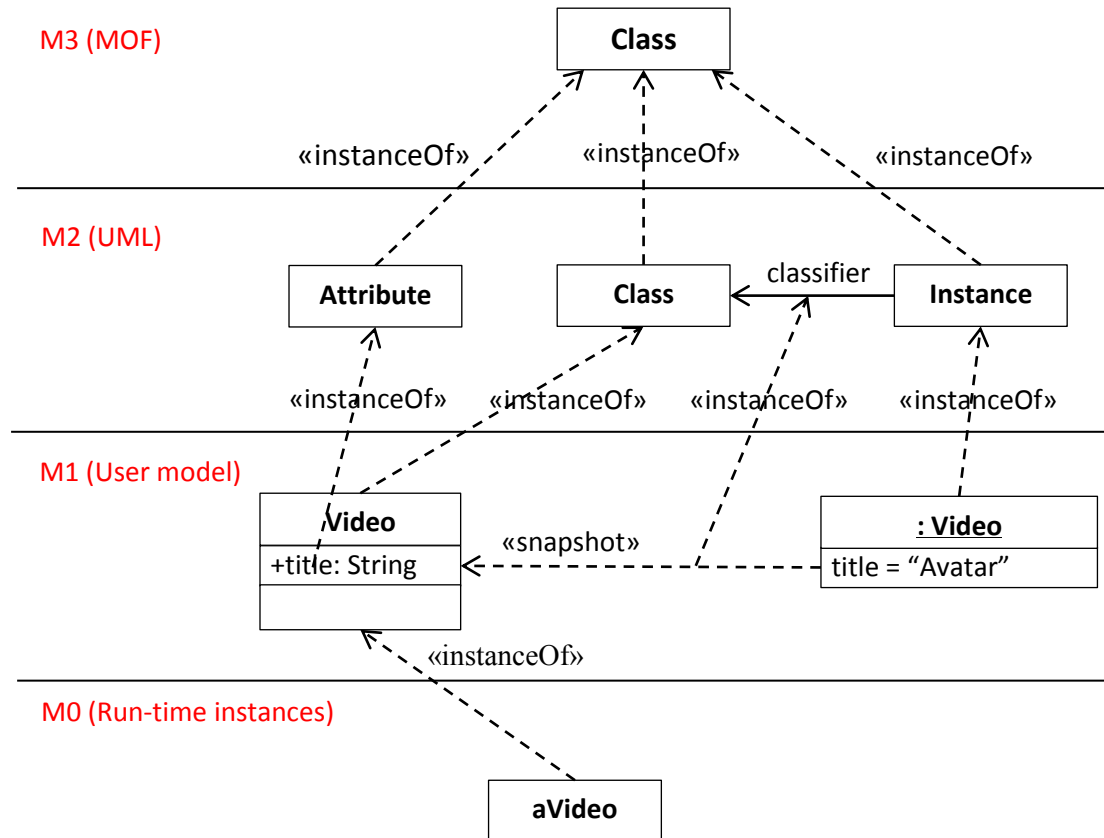


Figure 4.4. An Example of the Four-Layer Metamodel Hierarchy [28]

Abstraction can hide implementation details. Consequently, a system can have multiple layers of abstraction. Each relatively abstract, higher level builds on a relatively concrete, lower level, resulting in an increasing design resolution, or granularity. Each level represents a different model of the same system. Different set of objects and compositions are involved in these models [72]. A system abstraction at a relative concrete level is usually subject to more constraints than that at a relative abstract level. Hence design space shrinks as design resolution increase.

Abstraction can create and use concepts that are purely theoretical entities (i.e., without physical embodiment). They, therefore, cannot be instanced. The use of abstract concepts can simplify a system description. For example, software engineering uses an abstract data type, which is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of

those operations [73]. Components implementing an abstract concept can take a variety of forms. For example, the interface of a component (object) can be specified by both the input/output and the service provided by the component during its interaction with other system components. Such an interface can be implemented by objects in a variety of ways. Such type of abstraction is so useful that design pattern in software engineering advocates the practice of “program to interface” [74].

Since abstraction provides less detailed definition of a concept than its real-world embodiment, the use of abstraction often implies approximation. Abstractions, though not necessarily exact, should be sound [70]. Some considerations in using abstraction are summarized below:

- *Simplicity vs. Completeness*: Architectural abstraction has to maintain information completeness while applying the KISS (Keep it simple, stupid) principle.
- *Precision/Fidelity*: Architectural abstraction must keep the approximation error within a reasonable range.
- *Multiple Aspects/Angles and Consistency*: Consistent definition of a system or its component must be maintained while abstracting the same subject from different aspects or angles.
- *Levels/ Resolution of Abstractions*: abstraction has to be detailed enough to be useful.
- *Understandability*: Abstraction should yield meaningful results that are human comprehensible, or interpretable. Therefore, human experts must be involved in developing abstraction.
- *Formality/ Representation*: Abstraction can be represented using textual or graphical format depending on the domain to be abstracted. Operations can then be defined on such representation to either support analyses or automate such analyses. Examples are mathematical operators on equations, graph theory on graphic representation, and regular expression programming on text.

**4.2.3. Modeling Process.** Modeling process, in this context, refers to a systematic way of developing a system model in terms of identifying both its forms and behaviors as well as being aware of possible design options. It also includes the rationale to derive such



information. This section discusses both the modeling process and related techniques that support the identification and modeling of a collection of systems. The modeling elements created in a modeling process are instances of the design set implemented by the chosen modeling language. For example, when using OPM as the modeling languages, the modeling elements include objects, processes, states, and links.

The modeling process is strongly influenced by modeling paradigms. Popular modeling paradigms include functional programming, object-oriented programming, and model driven architecture. As discussed in Section 4.2.2, object-oriented modeling is preferred for modeling most systems. As a result, the discussion here focuses on the object-oriented paradigm. Several object-oriented modeling processes and development methods have been proposed. Prominent examples include the Unified Process [75], the Catalysis approach [76], and the approach for real-time applications [77].

Typical object-oriented modeling consists of two steps: Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD). OOA applies object-modeling techniques when analyzing the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. OOA is part of the design formulation (as discussed in Section 4.1.1). It focuses on what the system does. The result of OOA is the function breakdown (as discussed in Section 4.1.1). OOD focuses on how the system does it. The result of OOD is derived system behavior (as discussed in Section 4.2.1).

Traditional OOA/D methods [29], such as OOSE [78], OMT [79], and Rational Unified Process [80], [81], focus on developing single systems only [3]. Such methods are inadequate in search-based architecture development, which requires explicit modeling of large design alternatives rather than single systems. As identified in [3], a general problem associated with the existing modeling techniques used in OOA/D methods is an inadequate modeling of variability. Their variability modeling capabilities are limited to variability of certain objects over time or creation of different variants of an object (e.g., inheritance and parameterization in object diagrams). These OOA/D methods do not include the abstraction and modeling of commonality, variability, and dependencies [3].

In software engineering, domain engineering, also known as product line engineering, is the systematic activity of using domain knowledge and reusable assets in the production of new software systems. The key aspects of domain engineering are variability and dependency modeling. Many of the techniques developed for domain engineering can be used in the variability and dependency modeling of systems in general. The application of feature modeling, a major technique in domain engineering, to search-based architecture development will be further discussed in Section 5.1.3.1. The engineering process of using domain engineering in the design space modeling of the search-based architecture development process is discussed here. Domain engineering encompasses three main process components: domain analysis, domain design, and domain implementation [3].

Domain analysis is used to define the domain (identifying domains and their boundaries), collect relevant domain information, and produce a domain model [82]. A domain model is an explicit representation of both the common and the variable properties of the systems in a domain, as well as the dependencies between the variable properties [3]. In general, a domain model consists of the following components: domain definition, domain lexicon, concept models, and feature models [3].

Domain design uses the domain model produced during the domain analysis phase to produce a generic architecture to which all systems within the domain can conform [17]. Such a generic architecture is an architectural pattern that can solve a problem common across the systems within the domain [18]. Domain implementation involves applying appropriate technologies to implement components, automatic component assembly, reuse infrastructure, and application production process [3].

Domain engineering methods aim at supporting the development of models for classes of systems. OOA/D methods, however, concentrate on single systems [3]. Domain engineering supports both a multi-system-scope engineering process and adequate variability modeling techniques. OOA/D methods provide effective system modeling techniques [3]. Thus, the integration of domain engineering methods with OOA/D methods can provide the full engineering process support to the search-based architecture development process. Such an integration can take four forms [3]:

- Upgrading older domain engineering methods,

- Specializing customizable domain engineering methods,
- Extending existing OOA/D methods, and
- Integrating two or more methods developed for above.

Many of the techniques developed for these methods are designed specially for software engineering. Some of the principles, however, can be applied to non-software systems. The integration of domain engineering and OOA/D methods for general system development, particularly in the context of search-based architecture development, are discussed next. Such an integration can encompass the following two steps.

Step 1. Augment OOA with context analysis: “The purpose of context analysis is to define the boundaries and contents of the system to be analyzed” [3]. Variability should be analyzed along with establishing the relationships between the domain of focus and other domains or entities [3].

Step 2. Augment OOD with domain modeling: The purpose is to identify and model the commonalities, variabilities, and their dependencies in a domain model [3]. This phase can involve the following activities:

- (1) *Entity Analysis*: The main purpose here is to capture both major system entities and the relationships between them [3].
- (2) *State and Process Analysis*: The main purpose here is to capture the major states that the system needs to go through to achieve certain functions. Then identify the processes that enable the achievement of these states or the transitions between them.
- (3) *Operational Analysis*: Operational analysis identifies how the system operates by capturing the relationships between the objects, object state, and processes in the system. It also maps processes to objects.
- (4) *Domain and Constraint Analysis*: Domain and Constraints Analysis identifies the attributes to describe the class of the object identified in the Entity Analysis step, along with the domain and its boundary of each attribute. It also identifies the implementation constraints for the identified object, processes, and states.
- (5) *Commonalty, Variability, and Dependency Analysis*. This step involves analyzing system functionalities, contexts, interfaces, and both similarities and

variations between entities, activities, events, relationships, structures, etc. The purpose is to identify the common elements, variable elements, and the dependencies and constraints between these elements. The design options, or variants, can be identified in a variety of ways. For example,

- Alternative and optional functionality [83]. For example, in responsibility-driven design, design variants can be defined around a responsibility and the input/output that it exchanged.
- Varying constraints and business rules [83]. Such constraints include constraints imposed by the chosen alternative, implementation constraints, and any non-functional constraints (technological or environmental).
- Varying user or system interfaces [83].
- Performance and scalability differences [83].
- Varying functional and behaviors mapping. Functions and behaviors may be mapped to either different physical elements or different internal interactions between those elements.

Although the search-based architecting approach intends to be an automatic process, designers must be actively involved in both the model synthesis phase and the validation phase. During model synthesis phase, designers should assist in the identification of design options as computers do not have the knowledge and data to do so. Similarly, in the validation phase, designers need examine the behavior produced by the system model and ensure it satisfies the requirements.

A system design process is a hierarchy reduction of ambiguity. Levels of system ambiguity can refer to both different levels of design details (or design resolution) and different levels of abstraction types. The former is associated with design decomposition activities. The purpose is to achieve more detailed and refined system designs as the design progresses. The latter refers to the nature of design models at various design phases, such as the functional architecture design, the system architecture design, and the physical architecture design. The system design completed at a certain level also establishes requirements for the next level. As a result, requirements flow down as the design progresses [13]. Furthermore, additional implementation constraints can be identified as more detailed information is available in each refined design level.

Therefore, as the design proceeds in such a process, the constraints increase (e.g., a physical model is subject to more constraints than a relative abstract model), the design complexity increases (as design resolution increases), the design space shrinks, and the ambiguity reduces.

### **4.3. ARCHITECTURE ASSESSMENT**

This architecture assessment in general includes three subtopics, analysis, selection, and optimization. The following discussions, therefore, are divided according to these three subtopics.

**4.3.1. Architecture Analysis.** The architecture analysis involves using analysis models or simulations to assess system performance. Architecture analysis is domain dependent and problem specific as the performance metrics, the extraction of raw data, and the problem formulation are all highly problem specific. Nevertheless, the analysis and simulation methods share some commonality. This section highlights some of these methods that can be used in architecture analysis, along with the discussion of some possible issues and concerns in applying these methods.

In order to distinguish the roles that simulation plays in architecture analysis, the architecture analysis methods discussed here are first roughly grouped into two categories: the evaluation based methods and emulation based (or reasoning about system interactions) methods. Parunak [84] used such classification in comparing agent-based modeling and equation-based modeling. Some of his conclusions apply largely to most of the evaluation based methods and emulation based methods discussed here. For example, both families recognize a system comprised of two kinds of entities: individuals and observables, each of which may have a temporal aspect [84]. “Individuals are bounded active regions of a domain while observables are measurable characteristics of interest” [84]. Evaluation based methods focus on numerical relationships, or mapping, between observables while emulation based methods focus on the causal relationships among entities or the behaviors resulting from individuals interacting with each other [84]. However such distinction is a tendency rather than hard rules. The two methods can be combined [85]. These two types of methods are further discussed in the next two sections.

**4.3.1.1 Evaluation-based approaches.** Here, analysis models for calculating performance measures are discussed in general followed by the discussion of some of those methods developed specially for architecture assessment. The search-based architecture development also poses additional challenges in architecture assessment such as ambiguity, error propagation and evaluation of large number of alternatives.

Alfaris provides an extensive review of analysis models for computing performance measures in his dissertation [13]. Generally speaking, analysis models differ in the nature of the metrics (qualitative or quantitative models), the way that the model is derived (deductive, inductive, or floating models), the fidelity or resolution of the solution produced (exact or approximation), the way that solutions are obtained (analytical or numerical), and the speed that solutions can be obtained. The designers have to make trade-offs sometimes between these aspects in choosing an appropriate analysis model for the system of interest.

A strong mathematical analysis usually requires a precise model, well-defined abstraction, and accurate data. Alternatively, when such details are not available, the architecture analysis can be performed by domain experts. Metrics may give very good values to individual observables but as a whole the architecture may not be at all suitable for the system in question [86]. Metrics, therefore, cannot replace the assessment of experts completely in some cases. Some popular system assessment methods that incorporate subjective information are Architecture Tradeoff Analysis Method (ATAM) [87], Quality Function Decomposition (QFD) [88], [89], Analytic Hierarchy Process (AHP) [90], Analytical Network Process (ANP) [91], Technique for Ordered Preference based on Similarity to Ideal Solution (TOPSIS) [92], elimination and choice expressing reality (ELECTRE) [93], preference ranking organization method for enrichment, evaluation (PROMETHEE) [94], Joint Probability Distribution Method (JPDM) [95], fuzzy logic based approach [96–98], Architecture value map (AVM) [66], and the canonical decomposition fuzzy comparative methodology [86].

The advantage to include subject matter expert's assessment and heuristics into the architecture assessment process is that they can address ambiguity, uncertainty and risks easily and the assessment can scale well to even complex systems [86]. The disadvantage is that these methods are low-resolution, subjective and unrepeatably [86].

Some of these methods places a heavy cognitive load on the decision maker and therefore are very difficult to be incorporated into automated search process [66]. For example, the QFD, as one of the most used system assessment method in the system engineering field, requires the subject matter experts to be actively involved in the assessment process. QFD is a “method to transform user demands into design quality, to deploy the functions forming quality, and to deploy methods for achieving the design quality into subsystems and component parts, and ultimately to specific elements of the manufacturing process” [99]. One of the most used techniques to implement QFD is the house of quality. A house of quality contains a relationship matrix that links customer’s requirements with the technical performance measures of the system with varying strengths. Both setting values for this relationship matrix and setting the rating and weight values for various dimensions involved in this house of quality require the active involvement of subject matter experts. Another example is the ATAM method, which is one of the most widely used and known method for the architecture assessment in software engineering. “The main points of ATAM are to elicit and refine a precise statement of the key quality attribute requirements concerning the architecture, to elicit and refine precise designing decisions for the architecture, and based on the two previous goals, to evaluate the architectural design decisions to determine if they fulfill the quality attribute requirements satisfactorily” [87]. The ATAM uses scenarios to analyze whether the architecture fulfills all the necessary requirements and to see risks involved in the architecture. The ATAM proceeds in nine steps: presenting the method for the group of experts, presenting business drivers, presenting the architecture, identifying architecture approaches, generating quality attribute utility tree, analyzing architecture approaches, brainstorming and prioritizing scenarios, again analyzing architecture approaches, and finally presenting the results [87].

#### **4.3.1.2 Emulation-based approaches and reasoning about system interactions.**

System properties resulting from the interactions of system components, action sequences and procedural specifications usually need to be captured and reasoned with the aids of modeling languages that are capable of capturing the causal relations between system components. Such properties can then be obtained through either analysis or simulation. Some related methods of this category are discussed and compared below.

Probabilistic Graphical Models [100], such as Bayesian Belief Networks (BBNs) [101–104] and Markov networks [105], use graph-based representations to encode the conditional independence structure among a set of random variables. BBNs use directed graph while Markov networks use undirected graph. “Both families provide the duality of independences and factorization, but they differ in the set of independences they can encode and the factorization of the distribution that they induce” [100].

Bayesian Belief Networks describes the relationships between causes and effects in a probabilistic sense (i.e., via conditional probabilities) and thus allow modeling and reasoning about uncertainty. Both associative and causal types of relationships can effectively be modeled and processed in a BBN [103]. The main use of BBNs is statistical inference. Given some observations, values of all the other probabilities in the BBN can be computed using propagation algorithms. Explicit modeling of causal relationships in a BBN not only allows to represent and respond to changing configurations but also “facilitates the analysis of action sequences, their consequences, their interaction with observations, and their expected utilities, and hence the synthesis of plans and strategies under uncertainty” [106]. BBN in conjunction with Bayesian statistical techniques also facilitates the combination of domain knowledge and data [104].

A Markov network is an undirected graph comprised of a set of random variables having a Markov property [100]. It represents the joint probability distribution over the variables. It is also possible to convert between a BBN and a Markov network [107].

Markov chains [108] are often used as statistical models of real-world processes. A discrete-time Markov chain is a state-transition system where transitions between states are specified by probabilities. The set of all states and transition probabilities completely characterizes a Markov chain.

Petri nets are a discrete-event-driven system modeling and simulation language as discussed in Section 3.2.3. Their core execution semantics is based on conditions, events and effects. The outcome of such causal relationships can be characterized by a state-transition system in a global sense and therefore can be described by a Markov chain. The state space of a Petri net is determined by the initial tokens and the conditions-events-effects-based execution semantics. Such a state space can be described using graphical



representation, which allows the user to actually observe the stochastic processes during a simulation. The major strength of Petri nets is that it combines a well-defined mathematical foundation, an interactive graphical representation, and the capability to carry out simulations and formal verifications. With a concise mathematical definition and a small set of model primitives, Petri nets allow a large number of formal analysis methods to be developed. A Markov chain expresses global states and transitions, the size of which grows quickly as the number of variables and their values increase and is, therefore, subject to explosion, like the state space explosion of Petri nets. A Petri net, in comparison, is somewhat an iterative state space generator because it focuses on expressing the states and events showing just one global state in each simulation step. Therefore, the model size of a Petri net is easier to manage than that of a Markov chain, irrespective of the number of tokens present or the domain size of token colors.

System Dynamics [109–111] is “a computer-aided approach to policy analysis and design” [112]. In system dynamics modeling, dynamic behavior is thought to arise due to the principle of accumulation [113]. The basic building blocks of a system dynamics model are stocks (or accumulations, state variables) and flows. A Stock represents an entity or variable that changes in a system. A flow is the rate of change in a stock. The dynamics of a system is caused or generated by loops of internal feedback and circular causality as well as time delays [112]. There are two types of feedback loops: positive loops and negative loops. Positive (or self-reinforcing) loops tend to reinforce or amplify the initial action while negative (or self-correcting, balancing) loops counteract and oppose the initial action [109]. Combined, positive and negative circular causal feedback processes can generate all manner of dynamic patterns [112].

Mathematically, the basic structure of a system dynamics simulation model is a system of coupled, nonlinear, first-order differential (or integral) equations [112]. The simulation is, however, achieved through numeric integration instead of solving differential equations analytically.

The system dynamics considers behavior as a consequence of system structure [112]. It models interdependencies among variables using structures. Unlike the event-oriented, reactionary approach of Petri nets, the system dynamics advocates the continuous view of structure and dynamics. Such view focuses not on events or discrete

decisions but on the policy structure underlying decisions [112]. Events and decisions are merely surface phenomena result from underlying system structure and behavior [112]. System dynamics also takes endogenous point of view of system behavior, i.e., the causes are contained within the structure of the system itself [112]. Therefore, most system dynamics models are time invariant. However, as identified in [1], using numeric values and arithmetic equations to specify the behavior of a system has difficulties to achieve change of model structure given certain triggering event.

Agent Based Modeling and Simulation (ABMS) [114–116] studies the actions and local interactions of constituent entities (agents) and their impacts on the system as a whole. In an Agent Based Model (ABM), a system is modeled as “a collection of autonomous decision-making entities called agents, each of which individually assesses its situation and makes decisions on the basis of a set of rules” [85]. Applications of ABMS span a broad range of areas and disciplines. ABM is “most appropriate for domains characterized by a high degree of localization and distribution, dominated by discrete decisions” [84] and there is potential for emergent phenomena. Bonabeau summarizes [85] the benefits of ABM over other modeling techniques as: (1) “ABM captures emergent phenomena from the bottom up” (i.e., by modeling and simulating the behavior of the agents and their interactions) (2) “ABM provides a natural description of a system” (i.e., from the perspective of its constituent units’ activities); and (3) “ABM is flexible” (e.g., adding agents, tuning the complexity of the agents, change levels of description and aggregation). The emphasis on modeling the heterogeneity of agents and the emergence of self-organization distinguish ABMS from other simulation techniques such as discrete-event simulation (Petri nets) and system dynamics [115].

**4.3.2. Architecture Selection.** As discussed in Section 4.1.2, multi-objective optimizations need an selection process to choose good designs that constitute a compromise of several different objectives. Such selection processes are supported by decision models. This section focuses on the decision models used in an optimization while the next section will focus on the search process of an optimization.

Depending on when the preference for each objective is expressed, multi-objective optimization methods can be broadly classified into two categories: decision making before search methods (also known as scalarization approaches), and search

before decision making methods (also known as Pareto approaches). As summarized in [13], [69], [117], examples of scalarization approaches include weighted sum approach, multi-attribute utility analysis,  $\epsilon$ -constraint methods, compromise programming (non-linear-combinations), physical programming, goal programming, lexicographic approaches, acceptability functions, and fuzzy logic; examples of Pareto approaches include exploration and Pareto filtering, multi-objective genetic algorithms, adaptive weighted sum method, normal boundary intersection, and multi-objective simulated annealing.

There are other classifications of optimization algorithms according to various considerations. Cohon [62] classified them into the following two types based on whether Pareto-optimal solutions are generated or not:

- Generating methods. In such methods, a set of non-dominated solutions are generated for the decision maker without *a priori* knowledge of relative importance of each objective. The solutions obtained are then present to the decision maker for selection.
- Preference-based methods. In such methods, some known preference for each objective is used in the optimization process.

Hwang and Masud [63] and later Mittinen [64] fine-tuned Cohon's classification into the following four classes of methods:

- No preference methods are generating methods that do not assume any information about the relative importance of each objective. Instead, a heuristic is used to find a single optimal solution. It is worth noting that these methods do not make any attempt to find multiple Pareto-optimal solutions [69].
- *A priori* methods are preference-based methods that use information about the preferences of objectives *A priori* and usually find one preferred Pareto-optimal solution.
- *A Posteriori* methods are generating methods where preference is used *a posteriori*. A set of Pareto-optimal solutions are produced by the algorithm. The decision maker then selects the most preferred one according to some further considerations.

- Interactive methods are preference-based methods that use the preference information progressively during the optimization process. It requires the interaction with the decision maker.

Some selected *a priori*, *a posteriori* and interactive methods are further discussed and compared below.

**4.3.2.1 A priori approaches.** In the decision making before search approaches, the designer decides how to aggregate different objectives into a single objective function (also known as fix-up) before the actual search is performed [117]. Such approaches require *a priori* knowledge to make rational aggregation. Several scalarization methods have been developed. A few of them are briefly reviewed here.

In the weighted sum approach, the “scalar substitute objective is obtained by assigning subjective weights to each objective and summing up all objectives multiplied by their corresponding weight” [118]. Optimization of this composite objective (scalar substitute objective) results in the optimization of individual objectives, which should not be related [119]. The weights reflect the trade-off (or preference) among the objectives. Hence, the outcome of such methods is highly affected by the chosen weights. The weighted sum approach can also be utilized to find the Pareto-front. This is achieved by varying the weights along the curve of a convex area. Such usage, however, does not apply to non-convex Pareto-fronts since not all points on the Pareto-front can be determined [69].

Utility approaches are based on the general formulations of utility theory. Most scalarization approaches can somehow be represented via the utility function approach [120]. An individual utility function is defined for each objective to represent the relative importance of the objective. “The overall utility function is an amalgamation of the individual utility functions and is a mathematical expression that attempts to model the decision-maker’s preferences.” [121]

$\epsilon$ -constraint methods choose one of the objective functions and treat the rest of the objectives as constraints by limiting each of them within certain pre-defined limits. Unfortunately, “the outcome of single-objective constrained optimization results in a solution which depends on the chosen constraint limits.” [69]

Goal programming methods [122–124] attempt to find solutions which attain a predefined target for one or more objectives. If no such solution can be found for all objective functions, the task is then to find solutions which minimize deviations from the targets. Note that, this task is somehow similar to that in satisfying decision-making and the obtained solution is a satisfying solution, which can be different from an optimal [69].

**4.3.2.2 A posteriori approaches.** In the search before decision making approaches, the search for optimal solutions is performed with multiple objectives being evaluated simultaneously, typically using the concept of “dominance” to rank solutions. Particularly, a solution  $x_1$  *dominates* another solution  $x_2$  if (1)  $x_1$  is no worse than  $x_2$  in all objectives and (2)  $x_1$  is strictly better than  $x_2$  in at least one objective [69]. This means a dominant solution is at least better in one objective while being at least the same in all other objectives. Strong (strict) dominance, however, requires  $x_1$  to be better in all objectives than  $x_2$ . The Pareto-optimal set is the entire set of non-dominated solutions among the search space, where the rest of the solutions are called dominated solutions [125]. Most Pareto-methods are concentrated on the approximation of the Pareto set [125]. They try to find a set of solutions as close as possible to the Pareto-front while keep the solutions diverse.

“All elements in the Pareto-optimal set define reasonable solutions and are subject to further decision factors in order to choose a design for a given problem” [117]. In this manner an unbiased search can be performed. Moreover, Pareto methods also allow a single search to serve several problem-specific decisions without the need to repeat the search [117]. This feature gives Pareto methods an advantage over single objective methods because the designers are provided with a wide range of non-dominated solutions from which one or more solutions can be chosen. This post-search selection can be supported by further analyses using domain knowledge, additional problem information, or decision criteria, which are not necessarily formulated in the design task.

**4.3.2.3 Interactive methods.** Interactive methods require minimum knowledge *a priori* but need the involvement of the decision maker occasionally during the optimization process. When some Pareto-optimal solutions are found, their locations and interactions are analyzed. The decision maker then provides some information about the search direction, weight vector, reference points, and other factors [69]. These

preferences are then incorporated in formulating and solving the optimization in the next iteration. Some of the most popular interactive methods include: interactive surrogate worth trade-off method [126], step method [127], guess method [128], non-differentiable interactive multi-objective bundle-based optimization system approach [129], reference point method [130], and light beam search [131].

No decision model is superior to others under all circumstances. The designer needs to select appropriate ones based on both the problem to be solved and the optimization algorithm employed. For example, the interactive methods require the involvement of the decision maker during the optimization. Hence it is only a semi-automatic process and, therefore, cannot handle large design space. Pure *a priori* methods are not flexible enough since the change of preference will affect the optimality of the obtained solution. *A posteriori* methods allow the designer to re-evaluate the obtained solutions after the optimization process. Deb [69] compared many decision models used in the multi-objective optimization. Here, the weaknesses of some of the widely used decision models are discussed based on Deb's study [69].

*Disadvantages of weighted sum methods:* Such methods require a precise weight value for each objective. As discussed in [69], since the mapping between the distribution of weight vectors and the Pareto-optimal solutions is usually unknown, it becomes difficult to set the weight vectors to obtain a Pareto-optimal solution in a desired region of the objective space. Similarly, different weight vectors do not necessarily lead to different Pareto-optimal solutions. Furthermore, most single-objective optimization algorithms are designed to find a solution that only satisfies the first-order optimality criterion but not necessarily be a global optimum. In addition, "if the chosen single-objective optimization algorithm cannot find all optimum solutions for a weight vector, some Pareto-optimal solutions cannot be found" [69].

*Disadvantages of  $\epsilon$ -constraint methods:* In such methods, the solution largely depends on the chosen  $\epsilon$  vector, which must lie within the minimum and maximum values of the individual objective function. "As the number of objectives increase, there exist more elements in the  $\epsilon$  vector, thereby requiring more information from the user." [69] Such methods also suffer the issue of non-uniformity in obtained Pareto-optimal solutions as the weighted sum methods do.

*Disadvantages of utility methods:* Such methods require designers to specify a utility function which is globally applicable over the entire search space. Such a utility function might be over-simplified. Moreover, the obtained solution entirely depends on the chosen value function.

*Disadvantages of fuzzy logic methods:* Such methods rely heavily on subjective judgment, which not only is subject to the limitation of human expertise but may also not always be available or may not be possible to be integrated into an automated computational process. The aggregation rules might also be subjective and often lack sound justification. The fuzzy rules try to establish a nonlinear mapping between design properties and the objectives. It is often either impractical or impossible to find an exact set of rules for a specific situation. Such methods also rely on converting a multi-objective optimization into a single-objective optimization and therefore suffer the same problems as other scalarization approaches.

**4.3.3. Optimization.** The architecture optimization in general is a constrained (e.g., by design requirements and restrictions), multi-objective optimization on a discrete design space. Optimization models used in the architecture search enable “moving from one configuration to the other in an ongoing search for better solutions, but more importantly it is established with the aim of control and guidance” [13]. In general, more than one acceptable design may exist. The multi-objective optimization requires a selection process to handle the trade-off among conflicting goals as discussed in last section.

Optimization methods have reached a high degree of sophistication, especially with the rapid advancement of computer technology. There are many optimization algorithms developed, some of which are presented in Figure 4.5. From the searching process perspective, optimization algorithms can be classified into either deterministic or stochastic (or heuristic) methods. Deterministic methods can be classified into gradient based methods and derivative-free methods [132].

Gradient-based algorithms can find local optima with high reliability and, in many cases, with high efficiency but might be trapped by local optima. Heuristic based algorithms can escape local optima and are stochastic in nature. They cannot guarantee the optimality of the solutions obtained and often yield different set of solutions each

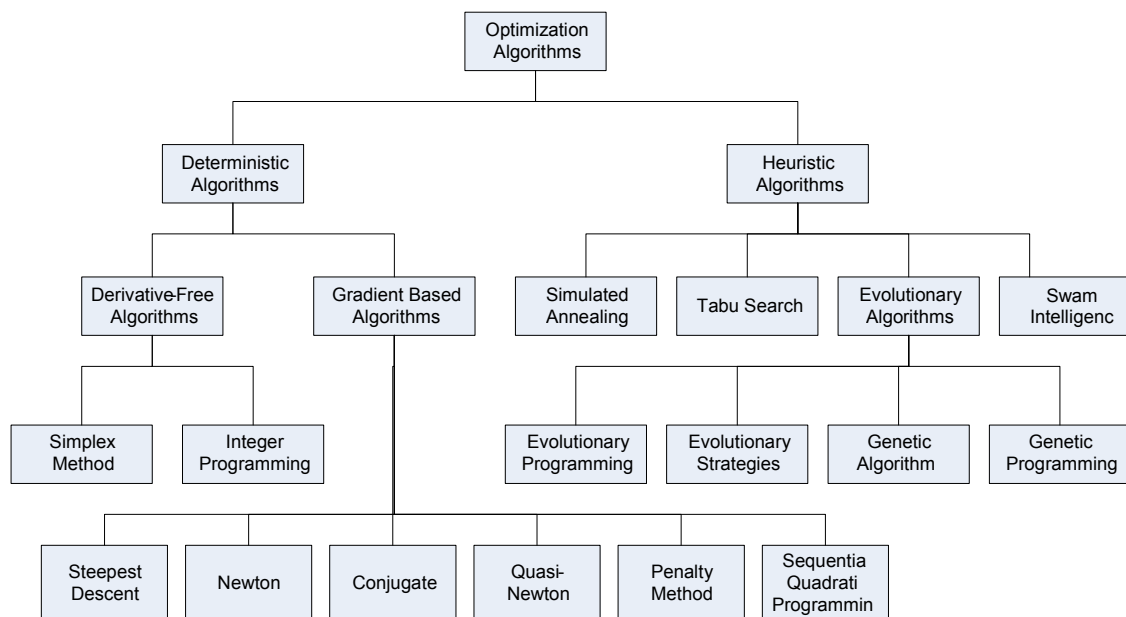


Figure 4.5. A Simple Taxonomy of Optimization Algorithms ([13])

time they are run. No existing optimization technique is guaranteed to find the global optimum of a nonlinear, non-convex problem [133], [134]

No single optimization technique is applicable in general to all types of problems. The most effective way, however, to solve a given problem will always be dependent on the specifics and details of that unique problem [135]. A hybrid method that combines optimization methods in a complementary way may ideally both benefit from the relative strengths of each individual method and restrain its weaknesses.

In the case of architecture development, the design space could be exceptionally large thus precluding the use of brute force algorithms. On the other hand, deterministic algorithms that would be fast enough either might not exist or would be too complicated to define. Hence the heuristic based search algorithms are more appropriate in such application, as they can find good enough solutions from a large design space within a reasonable amount of time with little or no reliance on the knowledge of the search space. Some heuristic based optimization algorithms that can possibly be applied to the search-based architecture development process are briefly discussed below. All these algorithms are good at handling problems with discrete solution space.



Hill Climbing (HC) [136] is an iterative algorithm that starts with an arbitrary initial candidate solution, then attempts to find a better solution by examining the set of “near neighbors” to the current solution. If a near neighbor can be found with a better fitness value, a move to the new solution is made. Such “walk up the hill” process is repeated until no further improvement can be found. The “near neighbors” are defined on the solution space. What constitutes a “near neighbors” is problem specific. Two types of strategies exist regarding the move to a better neighbor solution: (1) in the next ascent HC, the move is made to the first neighbor with an improved fitness; (2) in the steepest ascent HC, the move is made to the neighbor that gives the greatest increase in fitness after the entire neighborhood is examined [136].

Such HCs are only guaranteed to find local optima. Near-global optima can be reached by using restarts (known as multiple-restart hill climbing), or more complex schemes based on iterations (e.g., iterated local search), on memory, (e.g., reactive search optimization and tabu search), on memory-less stochastic modifications (e.g., simulated annealing) [137]. HC algorithms are memory efficient since they do not maintain a search tree. They consider only the current state and immediate future states [138]. A HC is easy to implement but surprisingly effective in many SBSE problems as discussed in [19], [139], [140].

Simulated Annealing (SA) [141], [142] is inspired by, and derives its name from, the annealing process in metallurgy. SA is another local search algorithm exploiting neighborhood concepts. It avoids the local optima (maxima) problem of HC by permitting moves to less fit solutions. At each iteration of the search process, SA attempts to replace the current solution with a random solution chosen according to a candidate distribution, which is often sampled from the neighborhood of the current solution. The new solution may be accepted with a probability that is a function of both the drop in fitness and a global parameter  $T$  (called the temperature).  $T$  is gradually reduced during the search process. Thus, with this  $T$  parameter, the SA can avoid local optima to a certain extent by giving more chances to less fit solutions in the earlier exploration stages but increasingly choosing the better solutions in the latter converging stages. The SA has been applied to several SBSE problems as discussed in [139], [140], [143–145].

Tabu search [146], [147] is another meta-heuristic local search algorithm that proceeds by setting barriers or restrictions to guide the search process. Tabu search uses a local search procedure to iteratively move from one potential solution to an improved one in its neighborhood until some stopping criteria are met. It avoids being stuck at local optima by using memory structures (known as the tabu list) which are a set of rules and banned solutions used to filter which solutions will be admitted to the neighborhood to be explored [146]. Such rules are applied to the neighborhood of the current solution resulting in the set of available moves, from which the best move is selected. Both the tabu rules and the ways of defining neighborhood vary greatly depending on the problem or the application. The memory structures used in tabu search can be divided into three categories [148]: short-term, intermediate-term, and long-term. Short-term memory prevents revisiting solutions recently considered. Intermediate-term rules bias the search towards promising areas of the search space. Long-term rules promote diversity in the search process (e.g., resets when the search gets stuck). The application of tabu search in architecture related problems can be found at [149], [150]

Genetic Algorithm (GA) [151] is one of the most used Evolutionary Algorithms (EAs). In GA, solutions (known as candidates, individuals or phenotypes) are encoded in a string form known as chromosomes (or genotypes of the genome). GA uses an iterative evolution process starting from a population of randomly generated candidates. In each generation, multiple candidates are stochastically selected from the current population based on their fitness. These candidates are then modified (by applying mutations, crossovers, or other reproduction operators) to form the offspring. The new population for the next iteration of the algorithm is produced from the offspring and the original population using a selection process. The GA terminates when certain pre-determined termination criteria (e.g. the maximum number of generations exceeded, satisfactory fitness level reached, etc.) are met. Many variants of this overall process exist, but the key ingredients i.e., recombination and selection guided by fitness functions, remain the same.

There is a variety form of EAs besides GAs, for example, evolution strategies, genetic programming, and evolutionary programming. Evolution strategies [152], [153] use primarily mutation and selection as search operators and use vectors of real numbers

as representations of solutions. In genetic programming, computer programs, rather than function parameters, are optimized and a tree-based chromosome is often used [154]. Evolutionary programming is similar to genetic programming, but the structure of the program is fixed and its numerical parameters are allowed to evolve [155]. It uses mutation as the main variation operator.

EAs, as popular search techniques, have many applications in architecture related problems, for example, the architecture design [6], [15], formulation of predictive models of software projects [156], [157], and testing [158], [159].

The multi-objective evolutionary algorithm (MOEA) is a popular Pareto-based optimization approach. Deb [69] suggested the following principle for an ideal multi-objective optimization procedure:

Step 1: Find multiple trade-off optimal solutions with a wide range of values for objectives.

Step 2: Choose one of the obtained solutions using higher-level information.

There are a number of advantages with ideal multi-objective optimization procedure as noted in [69].

- In such procedure, the decision-making becomes easier and less subjective. In Step 1, no preferences for the objectives need to be specified. The task is to find as many well-distributed, good solutions as possible. In Step 2, problem information, domain knowledge, or even subject experts can be used to conduct more detailed analyses before a final solution is chosen.
- The output of the algorithm is a population of solutions. If multiple optimal solutions are expected, such algorithm can yield multiple optimal solutions in its final population. On the other hand, if a single optimum is expected, all population members can be expected to converge to it as the algorithm runs.
- Such procedure also “eliminates the fix-up and can, in principle, find a set of optimal solutions corresponding to different weight and  $\epsilon$ -vectors” [69].
- “The avoidance of multiple simulation runs, no artificial fix-ups, availability of efficient population-based optimization algorithms, and above all, the concept of dominance helps to overcome some of the difficulties and give a user the practical means to handle multiple objectives”[69].

In summary, the MOEA is well suited for the search-based architecture development process. In addition, EAs “require little knowledge about the problem being solved, and they are easy to implement, robust, and inherently parallel” [160]. Deb [69] also summarized a number of deficiencies (especially when multiple Pareto-optimal solutions are expected) of many classical multi-objective optimization algorithms comparing to MOEA. Deb [69] noted that:

- (1) Only one Pareto-optimal solution can be expected to be found in one simulation run
- (2) Not all Pareto-optimal solutions can be found by some algorithms in nonconvex multi-objective optimizations
- (3) All algorithms require some problem knowledge, such as suitable weights or  $\epsilon$  or target values.

Moreover, another problem with the methods that solve multi-objective optimizations by converting multi-objective optimization into single-objective optimization is that the solution obtained from solving single objective optimization is specific to the parameters used in the conversion process. In order to find a different Pareto-optimal solution, the parameters must be changed and the resulting new single-objective optimization problem has to be solved again [69]. Thus in order to find  $N$  different Pareto-optimal solution, at least  $N$  different single-objective optimization problems need to be formed and solved. Even doing so, some algorithms do not guarantee finding solutions in the entire Pareto-optimal region [69].

This section presented the search-based architecture development framework and its implementation guidelines, along with the discussions of some applicable techniques for each of its components. The implementation of such a framework entails a system model that can capture all the information needed for architecture specification and analyses, as well as a way to define the design space. Such kind of model cannot be readily developed using existing modeling techniques. Therefore, a holistic modeling approach is developed and presented in the next section.

## 5. HOLISTIC MODELING APPROACH

This section presents the development of a holistic modeling approach. It starts with a definition of the holistic modeling approach. Then the landscape of drawbacks and open issues of current modeling languages and paradigms is investigated. The purpose is to find the road to a solution that can address the specific needs of the search-based architecture development process. Follows the discussion, the characteristics of an ideal holistic modeling language are summarized. In order to achieve such holistic modeling, an integration of some existing modeling languages is proposed. Accordingly, an architecture alternative generation mechanism based on the proposed modeling approach is developed.

### 5.1. DEVELOPING A HOLISTIC MODELING APPROACH

In the search-based architecture development process, the design space is comprised of architecture models, which are actively involved in the assessment and search process. Hence, an integrated architecture model that contains all aspects of information needed for both design and analysis is preferred. Moreover, such an architecture development process also requires both a generative class model to represent the design space and a set of instance models to participate in the computation. Thus there is a need for holistic modeling. Particularly, the concept of a holistic modeling approach in this context is fivefold:

- One integrated model for system specification instead of multiple disjoint diagrams,
- Capture structural, behavioral, and dynamic aspects of the system of interest
- Capture design space (or constraints)
- Can be used as both static presentation and dynamic simulation.
- Support system analysis.

#### 5.1.1. Strengths and Weaknesses of Some Existing Modeling Languages.

Jorgensen [161] conducted an extensive study on modeling languages for active process modeling. The languages studied include UML, System Dynamics, Petri nets, and BPML (Business Process Modeling Language) as well as other textual, informal, and semi-

formal process languages. Jorgensen's studies shows that these languages share some common weaknesses as far as the interactive process modeling is concerned. Such weaknesses also apply when more general system modeling is concerned. Hence it is cited here. Particularly, during these studies, Jorgensen [161] notated the following:

1. Many languages are complex, containing numerous types and views not integrated in a systematic manner. This is especially the case for UML.
2. In many cases mathematical, logical or technical concepts are applied instead of user or domain oriented (needs). Petri nets and constraint-based languages exemplify this.
3. The languages that are precise and formal enough for automatic execution offer few opportunities for human contributions to interactive activation. The languages do not handle process models with varying degrees of specificity.
4. The semantics of language elements is generally static and not easily adopted to local context or multiple perspectives.

As the literature review suggests, existing modeling languages emphasize and excel at only certain aspects of system modeling. The search-based architecting is still in need of a holistic modeling language. This section focuses on three major languages, UML/SysML, OPM, and Petri net, which are more relevant to the needs of search-based architecting. Table 5.1 summarizes the performance of these languages in some major aspects of comparison. The detailed discussion will be followed.

Although UML and SysML are the *de facto* object-oriented modeling languages for software engineering and systems engineering respectively, they have some drawbacks as far as the search-based system architecture development is concerned. Such drawbacks can be summarized as complexity, multiplicity, inconsistency and insufficient support of system analysis. The details are discussed as follows:

UML/SysML is intended to be a comprehensive modeling language capable of providing as much details as needed for building a product. Such intension inevitably results in its complex in terms of both language structure and entity definition. For example, UML contains more than 200 different graphical primitives and 13 diagram types [20], many of which involve advanced but convoluted concepts. Mastering and correctly using such languages requires highly skilled professionals and the language itself might be even more complicated than the problem to be solved. On the other hand, such complexity is not necessary for use in conceptual designs or architecture designs but

Table 5.1. Comparison of UML/SysML, OPM, and Petri Nets

<i>Aspects</i>		<i>UML/SysML</i>	<i>OPM</i>	<i>Petri Nets</i>
Model format	Graphic	O	O	O
	Text	X	O	X
	Mathematics	X	X	O
	Model Singularity	X	O	O
Model Coverage	Structure	O	O	X
	Behavior	O	O	O
	Dynamic	X	X	O
	Mathematics	O (OCL)	X	By programming
Model Capability	Presentation	Good	Excellent	Poor
	Specification	Excellent	Good	Excellent
	Communication	Excellent	Excellent	Poor
	Simulation	Poor	By extension	Excellent
	Analysis	Poor	Poor	Excellent
Model Notation	Compactness	Poor	Good	Excellent
	Usability and convenience	Poor	Excellent	Poor
	Advanced expression	Excellent	Good	Poor

Note: The dimensions within the notation category are adopted from [162]. Their definitions are as follows:

- Compactness: the number of (1) different symbols required to fully model the system, and (s) distinct diagram types.
- Usability and convenience: the time required to model the system, including necessary rework, number of entities in a single diagram, and the level of support for complexity management from a tool independent stand point.
- Advanced expression: the ability of the methodology to represent specific types of model components such as object, states, logical conditions, message sequencing, deployment or physical views, and packaging or encapsulation.

will complicate the design task correctly using such languages requires highly skilled professionals and the language itself might be even more complicated than the problem to be solved. On the other hand, such complexity is not necessary for use in conceptual designs or architecture designs but will complicate the design task.

A complete UML/SysML model specifying a system usually consists of multiple views such as use case view, structure view, behavior view, and implementation view. Each of these views may employ multiple diagrams. The UML/SysML specifications have not explicitly identified the necessary, direct, one-to-one, semantic mapping between related entities from different UML/SysML diagrams. For example, the definition of state in UML is arbitrary. According to UML [33], “a state models a situation during which some (usually implicit) invariant condition holds”. “The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some behavior (i.e., the model element under consideration enters the state when the behavior commences and leaves it as soon as the behavior is completed)”. It is not clear how such so-called dynamic conditions can be mapped to the actions or activities in the activity diagrams. A composite state either contains one region or is decomposed into two or more orthogonal regions. Each region has a set of mutually exclusive disjoint subvertices and a set of transitions [33]. However, it is not clear how such orthogonal regions can be reflected in the activity diagrams. A state can have such associations as `doActivity`, `entry`, and `exit`. These are defined as behavior but not necessarily reflected in the activity diagrams. A state can either be explicitly associated with an object identified in the class diagrams or implicitly with a set of objects. State transitions are triggered by events. Such events could be but may not be explicitly identified in other diagrams. Many other diagrammatic languages with multiplicity features suffer the same inconsistency issues as UML/SysML. Although vendors of UML/SysML modeling tools may choose to implement, more or less, such consistency constraints in their products (such as Artisan Studio), integrating multiple graphical representation and maintaining full consistency are still challenging.

On the other hand, these diagrams are intended to be illustrations of design concepts; they are not inherently computable graph structures [1]. Automatic analyses



and simulation using UML/SysML models requires precise execution semantics. Hence, the Semantics of a Foundational Subset for Executable UML Models (FUML) Standard [163] is recently developed. With such a semantic supplement, UML execution models can be executed, independent of target implementation, by means of a virtual machine. Since graphical modeling notations are not appropriate for detailed programming a standard textual action language conforming to FUML semantics was also developed called Action Language for Foundational UML (ALF) [164]. Recently, a reference implementation of FUML activity models was also developed using Java [165]. This implementation is capable of accepting as its input an XMI file from a conformant UML model. Additionally it provides an execution trace of the selected activity model(s) as its output. This reference implementation, however, provides simulation capabilities only. No time events or constraints are implemented. Support for formal analysis, such as construction of occurrence graphs (representing all reachable states), has yet to be developed [166]. Its ability to analyze, verify, and validate system requirements and design is, therefore, limited. Since these standards have just been published on 2011, their vendor supports are rare.

On the other hand, comparison studies [37], [39], [162], [167], [168] show that OPM have some advantages over UML in both software systems design and system modeling and design in general. Firstly, OPM is able to avoid the model-multiplicity issues of UML [168]. While UML is a multiple-view, object-oriented modeling language, OPM supports a single unifying, structure-behavior view [168] (i.e. both object and process oriented). UML/SysML uses several views to separate concerns, while OPM handles complexity by gradual refinement/abstraction of information and smooth transition across lifecycle phases [162]. Secondly, OPM is geared towards modeling systems in general [37]. OPM provides a much smaller set of modeling primitives and notations that are easy to comprehend while still maintaining good specification quality [168]. Over complicated modeling formalisms, on the other hand, will jeopardize both comprehensiveness and specification quality. Furthermore, OPM has not only adopted and extended many object-oriented concepts and ideas but also incorporated a number of fundamental ideas that go beyond object-oriented principles, for example, the definition of processes independently of objects and the way objects interact with each other via

processes [37]. Such feature further enhances the flexibility of modeling a system which in turn also increases user's comprehension and processing capability. In addition, it is easy to extend OPM or map OPM to other modeling formalisms, for example, the UML [39] and SysML [169].

However an OPM model usually cannot capture as much details as UML/SyML can [162]. Nevertheless, the granularity of an OPM model is high enough for general system modeling and even for detail-demanding tasks like code generation [30], [152]. A major drawback of OPM is that it does not have a formal mathematical definition, does not have well-documented execution semantics, and does not specify a formal computational model for either discrete or continuous event systems [1]. It cannot capture the dynamic aspects of a system either, i.e., an OPM model cannot describe the state of a system at a particular point of time. OPM as a visual modeling language provides a limited set of rules to specify the precedence of process execution order [1] and does not supported advanced features such as nested state either. The animation of OPM model supported by OPCAT provides the capabilities to check logic correctness of the modeled behavior only. Such animation is not formal enough to support strong analysis. Furthermore, a standard OPM (without extension) does not have numeric concepts and time concepts. Nevertheless, its flexible definition of object and process can be mapped onto operands and operators, respectively, of a wide range of formal computational models [1] and thus allows enhanced, formal definition of its modeling primitives. As a matter of fact, the OPCAT has already incorporated some numeric and time concepts.

Unlike UML and OPM, Petri nets have well-defined execution semantics and rigorous mathematical representation [70], which contains very few, but powerful, primitives. Such concise mathematical definition is a dominating strength of Petri net because it not only allows extending the basic Petri nets to achieve more enhanced functionalities but also makes it easy to develop many formal analysis methods and tools. Because Petri net has well-defined execution semantics, it can easily be implemented by programming language. Moreover, there exists a large collection of analysis methods and tools developed for various types of Petri nets, making Petri nets a very powerful tool for modeling, simulating, and analyzing discrete event systems. As discussed in Section 3.2.3, with the use of tokens, Petri nets can describe the dynamic aspects of a system

which neither UML nor OPM can. CPNs extend the vocabulary of basic Petri nets by allowing tokens to have an associated attribute. CPNs also support hierarchical Petri nets making it easier to scale to large system modeling. The incorporation of high-level programming languages also provides CPNs with the primitives for definition of data types and manipulation of their data values [170]. Therefore complex information can be represented in the token values and inscriptions of a CPN model [171], making CPNs capable of modeling complicated behavior with great flexibility. Jensen [170] provides an in-depth discussion of the advantages of CPN.

Although the reference implementation of FUMML provides convenient simulation capability by allowing direct execution of a SysML activity model, CPN provides capabilities beyond those of which reference implementation and many other executable formalisms are capable of. A detailed comparison of the simulation capabilities between CPN and FUMML can be found in [166]. Hence only the key points are highlighted here. First, CPNs combine a rigorous mathematical definition, an interactive graphical representation, and capabilities to carry out simulations and formal verifications into a concise modeling formalism. The FUMML reference implementation only provides textual execution trace. Secondly, it is possible to use the same (or at least very similar) models to check both the logical and functional accuracy of a system and to analyze performance [172]. Third, CPNs are very flexible in token definition and manipulation making CPN modeling even more flexible. Finally, CPNs can be extended with a time concept that has not yet been implemented in FUMML.

However, Petri nets are weak in defining the structural aspects of a system. For example they cannot represent long-term relationships between system objects. CPNs are not object-oriented. Additionally, they do not have the facilities to support either model reuse or scalability like the classification-instance, inheritance, and polymorphism supported by most object-oriented formalisms. Various versions of object-oriented Petri nets have been proposed in literature, such as [173–179]. These object-oriented Petri nets extend the basic Petri net, or CPN, with object-oriented concepts and constructs. They also support various degrees of object-oriented concepts or ideas, such as inheritance and polymorphism. Although they can capture persisting objects, they still cannot capture

long term relationships between objects. Thus, these object-oriented Petri nets still have difficulty to capture full structural aspects of the system.

**5.1.2. Characteristics of an Ideal Holistic Modeling Language.** Based on the needs of the search-based architecture development process, an ideal holistic modeling language as defined in the beginning of Section 5.1 should have the following characters:

- It must be domain independent;
- It must be universal and support generic object-oriented concepts;
- It must be capable of modeling the structural, behavioral, and dynamic aspects of a system;
- It should support both graphical and textual syntax;
- It must be precise, mathematically rigorous, and executable;
- It must support system analysis;
- It must capture design space or constraints;
- It must support hierarchical abstractions;
- It should consist of a relative small set of modeling constructs and notations.
- It should be easy to understand and use. i.e., the modeling constructs and notations should be intuitive to architect;
- It should facilitate data exchange for sharing models and communicating with other computer programs and database;
- It should facilitate the communication between stakeholders and architects from different knowledge domains;
- It must be easy to implement using programming language;
- It should encourage the use of one integrated representation instead of multiple disjoint diagrams.

**5.1.3. Combining UML/SysML, OPM, Petri Nets, and Feature Models.** Based on literature review conducted, a holistic modeling language as identified in Section 5.1.2 has yet to be designed. Each of the modeling languages studied has only been able to partially fulfill these needs. Defining and implementing a fully-fledged modeling language not only is a very challenging task but also has the disadvantage of lacking supports and acceptance. Therefore, instead of developing a new modeling language from scratch, this research proposes the integration and combinational usage of existing

modeling languages, i.e., the OPM, CPN and feature model. This approach not only allows user to benefit from the advantage of these individual modeling language but also allows the existing software tools and analysis methods developed for them to be reused. The way that these languages can be integrated is illustrated in Figure 5.1:

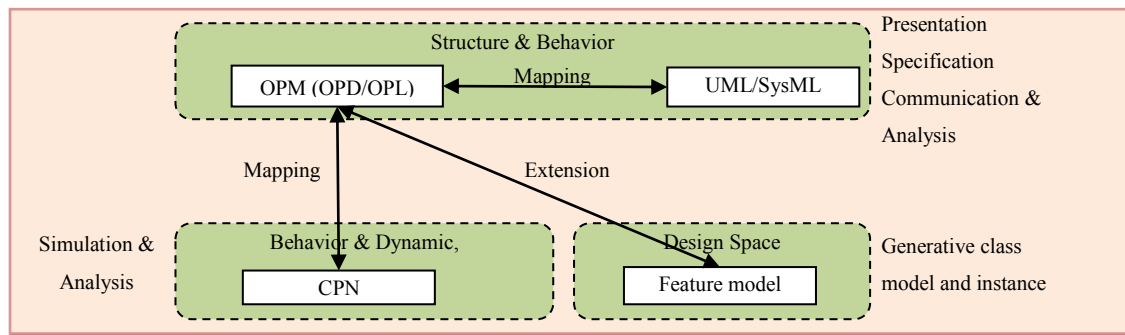


Figure 5.1. Combining Existing Modeling Languages to Achieve Holistic Modeling

The integration works like this: The formal system model is to be specified by OPM which serves as the hub of integrating other modeling formalisms. The reason that OPM is selected to play the integrator's role is that it is the closest to holistic modeling among those languages investigated. Additionally, it contains a very small set of language primitives which make it easy to extend OPM's definition to include new capabilities. A UML (or SysML) model with multiple diagrams can be generated by either using the generation capability provided by OPCAT [42] or following some other proposed mapping schemes [39], [169]. UML (or SysML) models are expected because they are usually considered as more standard way for illustration or communication. A standard OPM model, however, still lacks the ability to capture dynamic aspects of system behavior, certain numeric properties (e.g., time), and constraints. Additionally, it lacks well-documented execution semantics. This research proposes utilizing CPN to formally define the execution semantics of OPM such that the simulation capability and analysis methods developed for CPN can be utilized. Moreover, OPM models are not

intended to capture design space. Thus, this research propose incorporating feature model concepts and domain engineering into OPM modeling so that OPM can be used to develop a class model that represents a collection of instance models.

The mapping between these modeling languages must be developed. This dissertation employs the existing work to map OPM to UML [39] or SysML [169]. This dissertation proposes extending OPM with feature model concepts. Such extension will be introduced in Section 5.1.3.1. This dissertation also proposes the mapping between OPM to CPN as a way to supplement OPM with well-defined execution semantics. Such mapping will be introduced in Section 5.1.3.2.

The holistic modeling approach proposed here uses OPM as the formal language for specifying a system. Thus, the OPM model should provide extended information to incorporate the concepts of the feature model for design space specification and to support the generation of CPN model. Such an extension can be achieved by defining the metamodel of the OPM/H using the object-oriented paradigm such as the MOF of UML [33]. In doing so, the extended information can be incorporated into the metamodel of the extended OPM in the form of properties added to related metaclass. A formal definition of the extended OPM is given in Section 5.1.3.1 below. There are a few other extensions to OPM in literature. For example, Mor Peleg and Dov Dori [180] proposed OPM/T. This is an extension of OPM for the specification of reactive and real-time systems. This extension (provided in OPM/T) includes triggering events, guarding conditions, temporal constraints, and timing exceptions. This research adopted some of Mor Peleg and Dov Dori's ideas [181] in developing the extended OPM.

**5.1.3.1 Formal definition of the extended OPM.** The metamodel of an extended OPM for holistic modeling (known as OPM/H hereafter) can be defined, using an object-oriented paradigm, as follows: (Optional properties are enclosed in “<” and “>.”)

$$Sys = \{O, S, P, L, M_0\}$$

where

1.  $Sys$  = OPM/H model of the system.
2.  $O$  = a set of objects in the system. That is,  
 $O = \{O_i, i = 1, 2, \dots, I_o\}$ ,

where

$O_i$  = the object  $i$  in the system. It is defined by an 11-tuple property set, (Name, Type, <Value>, <Constraint>, Essence, Affiliation, <States>, <multiplicity>, <Description>, <URL>, <Dynamic>). The property sets can be extended with additional fields if necessary.

$I_o$  = the total number of objects in the system.

1.  $S$  = state set defined for each object in the system, i.e., an elaboration of the state property of the object class. That is,

$$S = \{S_i, i = 1, 2, \dots, I_o\},$$

where

$S_i = \{S_{ij}, j = 1, 2, \dots, I_{si}\}$  is the set of states in object  $O_i$ .

$I_{si}$  = the total number of states in object  $O_i$ .

2.  $P_k$  = a set of processes in the system. That is,

$$P = \{P_i, i = 1, 2, \dots, I_p\},$$

where

$P_i$  = the process  $i$  in the system. It is defined by an 8-tuple property set, (Name, Essence, Affiliation, <Guard condition>, <Code segment>, <Time delay>, <Description>, <URL>). The property sets can be extended with additional fields (e.g., adding a Body field).

$I_p$  = the total number of processes in the system.

5.  $L$  = a set of links among distinct things (objects or process) in the system. That is,

$$L = \{L_i, i = 1, 2, \dots, I_l\},$$

where

$L_i$  = the link  $i$  in the system. It is defined by a 3-tuple property set, (Source, Destination, <TypeProperties>). Among them, the TypeProperties is a set of properties, the value of which depends on the type of the link as summarized in Table 5.2. The property sets can be extended with additional fields if necessary.

The “XOR” and “OR” relations are special types of links. An XOR (or OR) relation connects one entity (object, process, or state) at its singularity end (source or destination) to a set of links (other than XOR or OR) at the other, multiplicity, end (destination or source). An XOR relation applies the XOR operation to the set of links

Table 5.2. Properties of OPM Links

<i>Category</i>	<i>Links</i>	<i>Properties</i>
Structural Relations	Aggregation-Participation	Participation constraint
	Exhibition-Characterization	
	Generalization-Specialization	
	Classification-Instantiation	
	Unidirectional Relation	Tag, Source participation constraint, Destination participation constraint
	Bidirectional Relation	Forward Tag, Backward, Tag, Source participation constraint, Destination participation constraint
Procedure Links	XOR/OR	N/A
	Agent Link	Condition, Path, Description
	Instrument Link	
	Result/Consumption Link	
	Effect Link	Condition, Path, Resource, Description
	Instrument Event Link	Condition, Path, Reaction Time, Description
	Consumption Event Link	Condition, Path, Reaction Time, Description
	Condition Link	Condition, Path, Description
	Exception Link	Condition, Path, Reaction Time, Description
	Invocation Link	Condition, Path, Reaction Time, Description

that it connects before those links are connect to the entity at the other end of the XOR relation. An OR relation works the same way as the XOR relation, except that it applies OR operation to the set of links that it connects to.



$I_l$  = the total number of links in the system.

6.  $M_0 = (OM_0, SM_0)$  = the set of initial markings of an OPM/H

where,

$OM_0$  = marking of all objects in the system. That is,

$OM_0 = \sum_{i=1}^{I_o} OM_{i,0}$ , where  $OM_{i,0}$  is the initial marking of object  $O_i$ , i.e., the initial value of object  $O_i$ .

$SM_0$  = marking of state of all objects in the system. That is

$SM_0 = \sum_{i=1}^{I_o} SM_{i,0}$ , where  $SM_{i,0}$  is the initial active state of object  $O_i$ .

Note: The possible values for property Essence (in object or process) is either physical or informatical; the possible values for property Affiliation (in object or process) is either environmental or systemic (Refer to the OPM manual [182] for definitions of the values of these properties.)

#### **5.1.3.2 Extend OPM with feature model concepts to capture design space.**

In software engineering, domain analysis and feature models are used to define product line. Such concepts can be incorporated into OPM modeling to define the architectural design space. For example, the concept of features (as in a feature model) can be applied to any model element in an OPM model because features are higher level concepts. Such usage of feature concept can be justified by the definition of features as introduced in [183], i.e., a feature is a prominent or distinctive user visible aspect, quality, or characteristic of a software system or system. Applying the feature concept to OPM model elements is more straightforward than applying it to other modeling languages, such as UML/SysML. Object-orientation makes more specific assumptions about objects, i.e., they have state and behavior and collaborate through interactions [3] while an object concept in OPM is broken down into its constituent object, state, and processes, which all have an explicit appearance in the OPM model.

A design space [30] is “a multidimensional space representing both requirements and design choices. It is spanned by a set of dimensions identifying relevant criteria for characterizing artifacts in a specific domain – components, subsystems, or complete systems”. Design spaces may comprise two types of dimensions: discrete dimensions (enumerate possible alternatives) and continuous dimensions (take values in a range, such as real values).

Following the concepts of the feature model, design elements in an architectural model can be categorized as either common or variable elements. Common elements are always part of a system and, therefore, can be modeled as mandatory elements using feature model concepts. Variable elements are part of only some systems and, therefore, can be modeled as either optional, alternative, or OR-relationship elements using feature model concepts. Common elements are not relevant to the decision making process. Variable elements span the design space, the dimensions of which is constituted by three types of entities, an optional element, a set of alternative elements, and a set of OR-relationship elements. Therefore, variable elements are the design variables, the value of which need to be determined in the system design process. Additionally, each variable element might be described by a set of attributes. Again these attributes can be categorized as either common attributes or variable attributes using the above feature model concepts. These variable attributes constitute the sub-dimensions of the variable element. It is the cross product of these variable attributes that determines the domain of the variable element. The total effective dimensions of the design space of a system are, therefore, the sum of sub-dimensions from all of the main dimensions computed recursively until to the top elements. Extended with the concepts of the feature model, OPM can be used to develop the generative class model.

Before presenting the rules to extend OPM with the feature model concepts, a cardinality concept needs to be defined first:

Cardinality is an interval denoted as  $[min..max]$  applied to an OPM element, where *min* is the lower bound and *max* is the upper bound. Two types of cardinality exist: participation cardinality (corresponding to the feature cardinality in the feature model) and group cardinality (corresponding to the group cardinality in the feature model).

An OPM can be extended with the feature model concepts by following the rules below:

1. A set of alternative things can be grouped and represented by one OPM object (or process, whichever applicable). Fill the value field of this object (process) with a Boolean expression, which is constructed by connecting the values representing alternatives with “XOR”. For example, the expression “(a) XOR (b) XOR (c)” means exactly one alternative out of the set {a, b, c} can be present. Such a Boolean

expression can be replaced with a notation that represents a generative function to be implemented if too many alternatives exist. The “Initial Value” field of the OPCAT can be used to contain such Boolean expressions.

In addition to representing the set of alternatives using a Boolean expression, OPM objects (processes) representing the alternatives can be created and then connected with the parent object (process) using the classification-instantiation link of OPM. Such mechanism is known as “expand” here. To expand is necessary if any of the alternatives needs to connect to other OPM things.

2. Similarly, a set of things with the “OR” relationship can be modeled by the same mechanism described above. However, the “OR” operator should be used to connect the set of values representing the OR-relationship things. If child objects (processes) are to be created, they can be connected to their parent object (processes) by any applicable OPM structural links.

3. The group cardinality of a feature can be captured by adding a multiplicity attribute to each OPM thing. Therefore, if a thing represents a set of alternatives, its multiplicity will be  $[1, 1]$ . If a thing represents a set of OR-relationship things, its multiplicity will be  $[1, N]$ , where  $N$  is the number of end nodes in the relationship. If a thing represents a set of things that are related by compound relationship with both XOR and OR operators in the Boolean expression, set the value of the multiplicity attribute accordingly. Otherwise, if a thing has no child connected to it with OPM structural link, its multiplicity value is  $[1, 1]$  by default. The *Number of instances* attribute of a thing in OPCAT can be used as the multiplicity attribute to model such group cardinality.

4. The mandatory and optional relationships of a feature model can be represented by participation cardinalities in an OPM. Particularly, add a participation constraint attribute to the structural links of OPM. Then apply the above defined cardinality concept to each terminal end of the link. It is known as participation cardinality here. Participation cardinality is a generalization of the mandatory ( $[1, 1]$ ) and optional ( $[0, 1]$ ) concepts of the feature model. The OPCAT provides such a participation constraint attribute.

5. The “requires” relationship of a feature model can be expressed by various OPM procedure links or OPM tagged structural links depending on the relationships between these entities in OPM semantics.

6. Other cross-tree constraints between things are represented by OPM tagged structural links.

7. The “OR” and “XOR” relationships between OPM procedure links can be expressed directly using the “OR” or “XOR” notations of OPM.

8. A root node representing the entire system is optional if all of its children nodes have a participant cardinality of [1, 1].

9. Other extended features and constraints can be added to corresponding OPM elements as feature attributes.

Note that the XOR operators, the OR operators, and the tag values in the OPM tagged structural links representing cross-tree relationships appeared in the above rules are for illustration purpose only. They can be replaced by other syntax entailed by the implementation. For example, the OR-relationship can be expressed, using PL notations, as  $f_1 \vee f_2 \vee \dots \vee f_n$ , with  $f_i \mid i \in [1 \dots n]$  being the set of children participating in the OR relationship [184].

From the rules introduced above it can be seen that the current expressiveness of the OPCAT is capable of modeling these feature model concepts with little extension required. Hence it can be used to specify the design space of an OPM model. In order to illustrate using OPM notations and feature model concepts to define a design space, an example is give here. Figure 5.2 shows a sample feature model for the mobile phone, adopted from [14]. The corresponding OPM model, extended with the feature model concepts, is presented in Figure 5.3 (a). Both the mandatory elements (`Calls` and `Screen`) and the optional elements (`GPS` and `Media`) were captured by the participation cardinality of the aggregation-participation link. The alternative relationships (between `Basic`, `Color`, and `High resolution`) were captured both by the group cardinality applied to the `Screen` (default value 1 is not shown in the figure) and the *Initial Value* field of the `Screen` object as illustrated in Figure 5.3 (b). OPM objects were created for those alternatives because two of them (`Basic` and `High resolution`) were connected to other OPM things. *OR* relationship (between `Camera` and `MP3`) were captured both by the group cardinality applied to the `Media` (value “2” inside the box representing the `Media` object, which is the upper bound of the group cardinality) and the *Initial Value* field of the `Media` object as illustrated in Figure

5.3 (c). OPM objects were created for those alternatives because two of them (*Basic* and *High resolution*) were connected to other OPM things. Both *requires* constraints and the *excludes* constraints were captured by the tagged structural links of OPM.

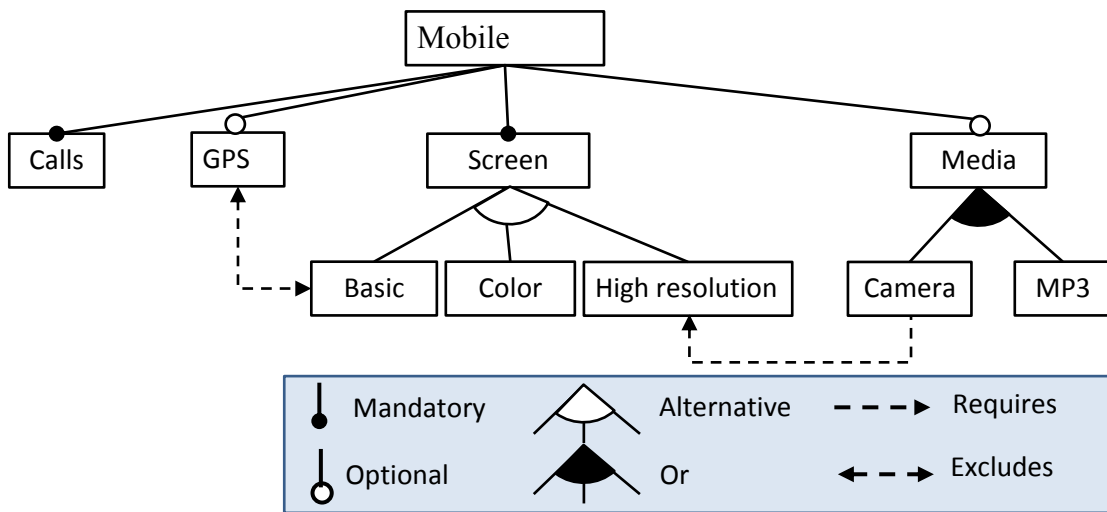


Figure 5.2. A Sample Feature Model ([14])

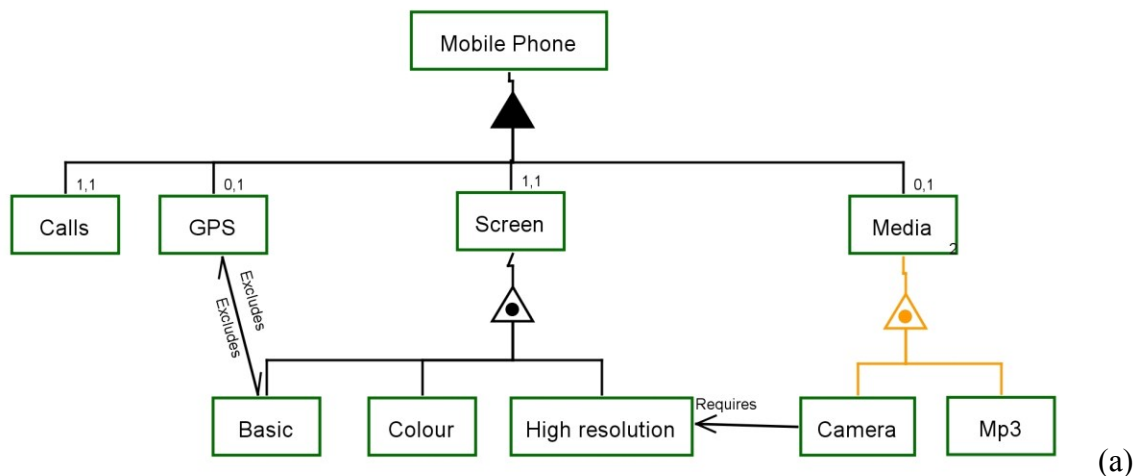


Figure 5.3. An OPM Model (Created by OPCAT) Extended With Feature Model Concepts to Capture Design Space

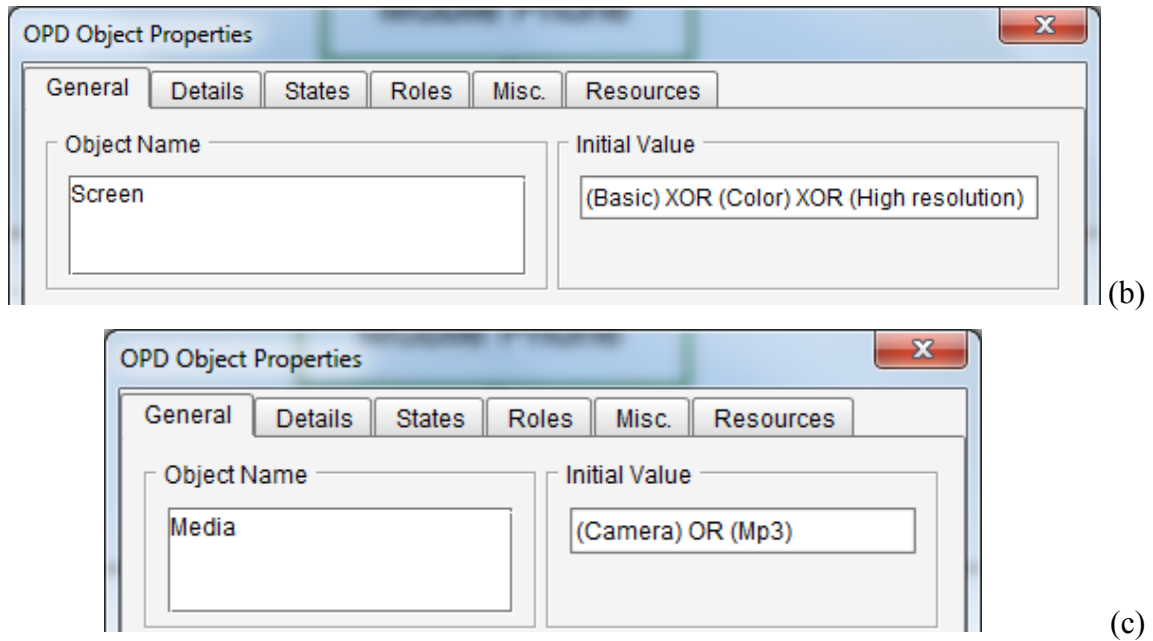


Figure 5.3. An OPM Model (Created by OPCAT) Extended with Feature Model Concepts to Capture Design Space (cont.)

The number of dimensions for the design space of this system was 3. These dimensions, along with their domains, were as follows: (1) GPS: {True, False}, (2) Screen: {Basic, Color, High resolution}, (3) Media: {Camera, MP3, Camera AND MP3, False}.

**5.1.3.3 Supplementing execution semantics of OPM with CPN.** An OPM/H model also contains extended information to support the construction of a CPN model. Such additional information can be viewed as annotations added to a regular OPM model. Such information includes link conditions, guard conditions, code segments, time delays, and markings. These types of information should be defined according to the need of the CPN model to be generated. Their semantics is pure CPN semantics. The details of these types of extended information are as follows:

The link condition, corresponding to the CPN arc inscription [170], [185], [186] or arc annotation [187], is an annotation to the procedure link of OPM. A link condition can include values, variables and expressions used alone or organized in a tuple. An instance of value allows consuming or producing a known value. A variable requires

binding of values to variable. Expressions yield new values through computation. Functions are also allowed in expressions. Functions used in link conditions allow complicated computations that are defined elsewhere. An Expression instance is only used in output procedure links. For more advanced topics regarding defanging link conditions, refer to arc inscription in CPN [185]. A link condition can be added to the “Condition” field of a procedure link. The syntax of the link conditions depends on the programming language chosen for implementing the CPN.

The guard condition, corresponding to the CPN guard [170], [186], [188], is a Boolean expression that evaluates to true or false. A guard condition can be added to the “Guard condition” field of a process (or, if using OPCAT, add to the description field using the format “[Guard: (expression)]”, where *expression* is to be replaced with the intended guard condition). The syntax of the guard conditions depends on the programming language chosen for implementing the CPN. Guards are used for testing variables in input link conditions (enabling restrictions) or restricting values of output link conditions variables. For more advanced topics regarding defining guard conditions, refer to guard in CPN [188].

The code segment, corresponding to the CPN code segment [170], [186], [189], is a piece of code executed when the hosting transition (corresponding to the OPM/H process) fires. A code segment can be added to the “code segment” field of a process (or, if using OPCAT, add to the description field using the format “[Code: (expression)]”, where *expression* is to be replaced with the intended code segment). The syntax of the code segments depends on the programming language chosen for implementing the CPN. For more advanced topics regarding defining code segments, refer to code segments in CPN [188].

A time delay is an expression evaluated to integer. A time delay can be applied both to a process and to an output procedure link from a process. When applied to a process, a time delay corresponds to the transition delay of the CPN. Such time delay can be added to the “Time Delay” field of a process (or, if using OPCAT, add to the description field using the format “[Time: (expression)]”, where *expression* is to be replaced with the time delay expression). When applied to a procedure link, a time delay corresponds to the arc delay of the CPN. Such time delay can be attached to

the end of the corresponding link conditions, using @+ as a separator. The syntax of the time delay expression depends on the programming language chosen for implementing the CPN.

Setting the initial marking of the OPM/H involves two operations: (1) Setting the initial values for related objects by giving a value expression to the “Value” field of each object. This operation will result in creating object instances for those objects. The value expression can be a single value, a set of values or a generative function defined elsewhere. The syntax of the value expression depends on the programming language chosen for implementing the CPN. (2) Selecting the initial state for objects with states.

In addition, there may be some OPM objects that have no impact on the dynamic of a CPN model. These objects should be identified and left out from mapping to CPN to avoid creating redundant information in the CPN model. For example, in a manufacturing system, the cost attributes of machine objects have no impact on the operation of the manufacturing system. Therefore, the OPM/H object representing the cost attributes can be left out from mapping to a CPN. To left out an OPM/H object from mapping to a CPN, mark the object by setting its “Dynamic” properties to false or add “[nd]” to the description field of the object if using OPCAT.

With such extensions, an OPM/H model can be transformed to a CPN model. As both OPM and CPN have graphical syntax, their mapping can be illustrated using graphs as well. Table 5.3 summarizes the mapping between OPM (where the syntax and semantics of OPM is extracted from [190]) and CPN. The basic idea is as follows. Map OPM processes to CPN transitions. Map OPM attribute objects (objects connected to their parent object using exhibition-characterization link) to CPN color sets. Such color set thus defines the set of class attributes for the OPM object being connected by those attribute objects. Map non-attribute objects that have no states and object states of OPM to CPN places. Map the value(s) of an OPM object to CPN token(s). One or a set of tokens on a CPN place represents either the existence of an object or an object being at the state represented by that place. The former corresponds to the cast that the place is mapped from an OPM object with no state and the token(s) on that place represent alternative objects. The latter corresponds to the case that the place is mapped from an OPM state. As discussed in Section 3.1.1, an object in the object-oriented modeling is



defined by three parts, states, attributes and services. By following the mapping scheme discussed above, a CPN token can capture the attribute and state part of an object definition. The service (or method, function or process) part of an object definition can be inferred if the CPN model created from the OPM follows certain naming convention. For example, an object's service can be modeled as an OPM process connected to the corresponding OPM object using an exhibition-characterization link. When such process is mapped to a CPN transition, the transition can be named by prefixing the corresponding OPM process name with the corresponding OPM object name. In doing so, the ownership relation between the object and the process can then be inferred. OPM structural links that have no effect on the system dynamics are not mapped to CPN. The details of the procedure for mapping an OPM/H model to a CPN model are as follows.

Step 1. Create a CPN transition for each OPM process (except for zoomed-in process). Name the transition with the format of "O\_T," where "O" represents the name of the OPM object connected to the process with an exhibition-characterization link and "T" represents the name of the process.

Step 2. Create a place for each OPM state. Name the place with the format of "O\_S," where "O" represents the name of the OPM object corresponding to that state and "S" represents the name of the state.

Step 3. Create a place for each OPM object connected to an OPM process with either an enabling / transforming procedural link or event / condition procedural link.

- a. If an OPM object with states is itself connected to an OPM process with such links, do not create a place for this object. Instead, create a set of arcs, each of which connects to a place created for a state of the object using the procedure in Step 8 below treating the relationship between these arcs as an OR.

Step 4. Objects that are not connected to any processes do not need to be mapped to CPN places.

Step 5. Create a color set declaration for each OPM object that is a child object in the exhibition-characterization link (except for those objects that are marked as "[nd]" in their description field). Name the color set with the name of the OPM object using uppercase letters. Type the color set with the type attribute in the description field of the extended OPM.

- a. Create a product color set declaration for each OPM object connected by more than one child via exhibition-characterization links.
- b. Start with the lowest level (leave nodes) of exhibition-characterization links. Move upward when there are multiple levels of parent-children relations with exhibition-characterization links.

Step 6. Type each place with the color set declared for the corresponding object in Step 5.

Step 7. Declare a variable for each color set identified in Step 5. Name the variable with the corresponding color set's name using lowercase letters.

Step 8. Create an arc for each enabling / transforming procedural link or event/condition procedural link connected with the process being mapped in Step 1 according to the mapping scheme presented in Table 5.3.

Step 9. Add arc inscription to each link identified in Step 8 according to the color set of the place the arc is connected to.

- a. The expression in the condition field of the link can override the arc inscription defined above. Replace the variable name (corresponding to the OPM object name) in the expression with the corresponding variable name defined in Step 7.

Step 10. Create a guard condition, code segment, or time delay for each transition identified in Step 1 using the respective expression in the description field of the corresponding OPM process. Replace the object names within the expressions with the corresponding variable names defined in Step 7.

Step 11. Assign tokens to places with the initial values of the corresponding OPM object

Step 12. When an Exclusive relationship connecting two OPM process exists, add a CPN state between the corresponding CPN transitions. Name the place with the name of the end process (both if bidirectional) preceded with "EXL". Type the place with a unit-like type. (Such typing depends on the language implementing the CPN.)

Step 13. Create a double arrow arc for each effect link that connects an object with no state.

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN

ENTITIES			
Name	Symbol	OPL	CPN
<b>Thing</b>			
Object		<p><b>B</b> is physical. (shaded rectangle)</p> <p><b>C</b> is physical and environmental. (shaded dashed rectangle)</p>	
Process		<p><b>E</b> is physical. (shaded ellipse)</p> <p><b>F</b> is physical and environmental. (shaded dashed ellipse)</p>	
Definition	<p>An <b>object</b> is a thing that exists.</p> <p>A <b>process</b> is a thing that transforms at least one object.</p> <p>Transformation is object generation or consumption, or effect—a change in the state of an object.</p>		<p>For a simple OPM object, the corresponding CPN place is class and a token on that place represents the existence of an instance of that class.</p> <p>No distinction of physical/informatical or environmental/systemic in CPN.</p>
State		<p><b>A</b> is <b>s1</b>.</p> <p><b>B</b> can be <b>s1</b> or <b>s2</b>.</p> <p><b>C</b> can be <b>s1</b>, <b>s2</b>, or <b>s3</b>.</p> <p><b>s1</b> is initial.</p> <p><b>s3</b> is final.</p>	
Definition	<p>A <b>state</b> is situation an object can be at or a value it can assume.</p> <p>States are always within an object. States can be initial or final.</p>		<p>Places are identified as states. Tokens are identified with objects.</p> <p>The color set correspond to the state place identify the set of objects that can visit those places, i.e., the set of objects owning those states. The tokens on a state place identify what objects are in that state</p>

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)


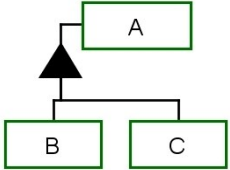
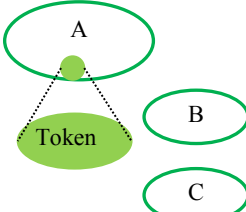
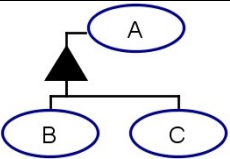
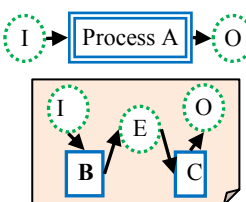
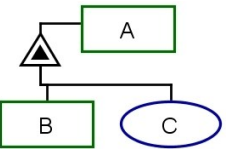
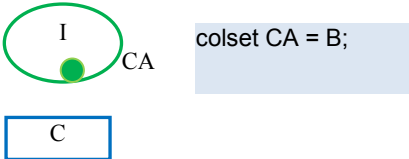
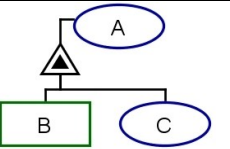
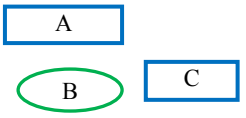
Structural Links			
Name	Symbol	OPL	CPN
<b>Fundamental Structural Relations</b>			
Aggregation-Participation		A consists of B and C.	
		A consists of B and C.	
Definition	A is the whole, B and C are parts.	No CPN mapping of aggregation relationships between OPM objects. Use substitute Transition plus sub-net to map aggregation relationships between OPM processes if hierarchical CPN is to be used. No need to create transition for process A in non-hierarchical CPN	
Exhibition-Characterization		A exhibits B, as well as C.	
		A exhibits B, as well as C.	
Definition	Object B is an attribute of A and process C is its operation (method). A can be an object or a process.	Exhibited OPM objects are mapped to the color set definition of the OPM exhibiting object. Multiple exhibited objects are mapped to product color set. No mapping of exhibition relations for OPM processes in CPN.	

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

Structural Links			
Name	Symbol	OPL	CPN
<b>Fundamental Structural Relations</b>			
Generalization-Specialization		<b>B</b> is an <b>A</b> . <b>C</b> is an <b>A</b> .	
		<b>B</b> is <b>A</b> . <b>C</b> is <b>A</b> .	
Definition	<b>A</b> specializes into <b>B</b> and <b>C</b> . <b>A</b> , <b>B</b> , and <b>C</b> can be either all objects or all processes.		No explicit map of Generalization-Specialization relations. Map children objects/processes only.
Classification-Instantiation		<b>B</b> is an instance of <b>A</b> . <b>C</b> is an instance of <b>A</b> .	For object:
			For process:
Definition	Object <b>A</b> is the class, for which <b>B</b> and <b>C</b> are instances. Applicable to processes too.		
Unidirectional & bidirectional tagged structural links		<b>A</b> relates to <b>B</b> . (for unidirectional) <b>A</b> and <b>C</b> are related. (for bidirectional)	
	Definition	A user-defined textual tag describes any structural relation between two objects or between two processes.	No mapping of tagged structural links.
<b>Link Relations</b>			
XOR Relation		<b>C1</b> consumes either <b>B1</b> or <b>A1</b> .	
Definition			Guard will determine the XOR relation.

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

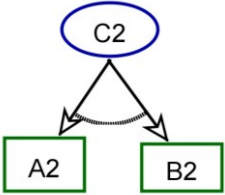
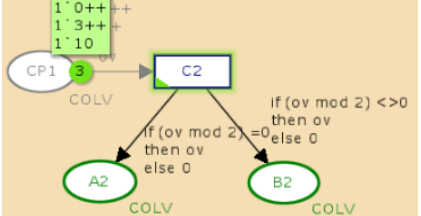
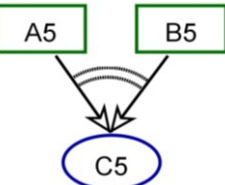
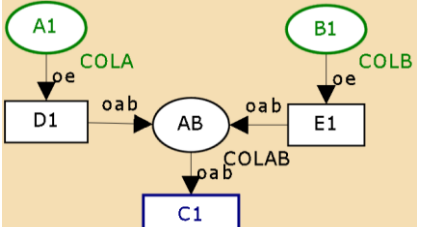
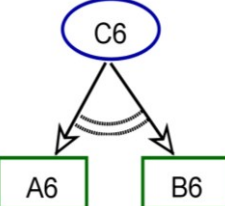
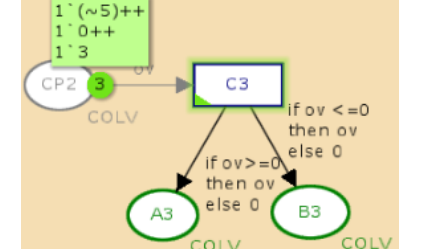

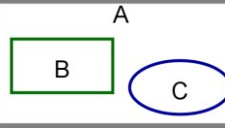
Name	Symbol	OPL	CPN
XOR Relation		C2 yields either B2 or A2.	
Definition			Arc inscription will determine whether it is OR, AND or XOR
OR Relation		C5 consumes A5 or B5.	
Definition			Guard will determine the OR relation.
OR Relation		C6 yields B6 or A6..	
Definition			Arc inscription will determine whether it is OR, AND or XOR
<b>Complexity Management</b>			
In-zooming		A exhibits C. A consists of B. A zooms into B, as well as C.	Sub-net.
Definition	Zooming into process A, B is its part and C is its attribute.		Same as the mapping for aggregation-participation link with parent thing being process. Expose subnet structure, i.e., no substitution for non-hierarchical CPN.
		A exhibits C. A consists of B. A zooms into B, as well as C.	N/A
Definition	Zooming into object A, B is its part and C is its operation.		Same as the mapping for aggregation-participation link with parent thing being object.

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

Enabling and Transforming Procedural Links				
Name	Symbol	OPL	CPN(A)	CPN(B)
<b>Enabling links</b>				
Agent Link		<b>A handles B.</b>	<pre>colset colA = unit; var os: colA;</pre>	
Definition	Denotes that the object is a human operator.			
Instrument Link		<b>B requires A.</b>		
Definition	"Wait until" semantics: Process <b>B</b> cannot happen if object <b>A</b> does not exist.		Place A: class A (existence). A token on A: an instance of A. Color set colA: set of possible instance values of class A (e.g. suppose A represents message, which can carry different values. therefore color set/token corresponds to object attributes/values). Arc variable os: the color set to be bound.	
State-Specified Instrument Link		<b>B requires s1 A.</b>	<pre>colset colA = unit; var os: colA;</pre>	
Definition	"Wait until" semantics: Process <b>B</b> cannot happen if object <b>A</b> is not at state s1.			
<b>Transforming links</b>				
Consumption Link		<b>B consumes A.</b>	<pre>colset COLA = unit; var oe: COLA; val cr = 5;</pre>	
			<pre>colset COLA = unit; var oe: COLA; val cr = 5; var amt: AMT; closeset AMT = INT; colset AxAMT = product COLA * AMT;</pre>	

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

Enabling and Transforming Procedural Links				
Name	Symbol	OPL	CPN(A)	CPN (B)
Definition	Process <b>B</b> consumes Object <b>A</b>		Place A1: class A1 (existence). A token on A1: an instance of A1. Number of tokens: number of instances of A1. Color set COLA: set of possible instance values of class A1. Arc variable oe: the colorset to be bound. Variable: cr: Consumption rate.	Place A: class A. A token on A: an instance of A with amount attribute. Support consuming more than one unit of object A at a time. Compound colorset with one dimension (AMT) identify the amount of object A. Color set AxAMT: set of possible instance values of class A. Expression "amt-cr": amount consumed. Only one token representing A is needed.
State-Specified Consumption Link		<b>B</b> consumes <b>s1</b> of <b>A</b> .	<pre>colset COLA = unit; var oe: COLA; val cr = 5;</pre>	<pre>colset COLA = unit; var oe: COLA; val cr = 5; var amt: AMT; colset AMT = INT; colset AxAMT = product COLA * AMT;</pre>
Definition	Process <b>B</b> consumes Object <b>A</b> when it is at State <b>s1</b> .		Same as the above consumption link except the following: Place A_s1: the s1 state of class A. A token on A_s1: an instance of class A (i.e., object A) at state s1	Same as the above consumption link except the following: Place A_s0: the s0 state of class A. A token on A_s0: the instance of class A (i.e., object A) at state s with amount attribute.



Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

Enabling and Transforming Procedural Links				
Name	Symbol	OPL	CPN(A)	CPN(B)
Result Link		<b>B</b> yields <b>A</b> .	<pre>colset cola = unit; var oe: cola; val cr = 5;</pre>	<pre>colset cola = unit; var oe: cola; val cr = 5; var amt: AMT; closet AMT = INT; colset AxAMT = product COLA * AMT;</pre>
Definition	Process <b>B</b> creates Object <b>A</b> .		Same as the consumption link except that the direction of the arc is reversed.	Same as the consumption link except that the direction of the arc is reversed.
State-Specified Result Link		<b>B</b> yields <b>s1</b> <b>A</b> .	<pre>colset COLA = unit; var oe: COLA; val cr = 5;</pre>	<pre>colset COLA = unit; var oe: COLA; val cr = 5; var amt: AMT; closet AMT = INT; colset AxAMT = product COLA * AMT;</pre>
Definition	Process <b>B</b> creates Object <b>A</b> at State <b>s1</b> .		Same as the state-specified consumption link except that the direction of the arc is reversed.	Same as the state-specified consumption link except that the direction of the arc is reversed.
Input-Output Link Pair		<b>B</b> changes <b>A</b> from <b>s1</b> to <b>s2</b> .	<pre>colset COLA = unit; var oe: COLA;</pre>	
Definition	Process <b>B</b> changes the state of Object <b>A</b> from State <b>s1</b> to State <b>s2</b> .		Same as the state-specified consumption link except the following: Place <b>A_s1</b> : the <b>s1</b> state of class <b>A</b> . Place <b>A_s2</b> : the <b>s2</b> state of class <b>A</b> . <b>A</b> token on <b>A_s1</b> (or <b>A_s2</b> ): an instance of class <b>A</b> (i.e., object <b>A</b> ) at state <b>s1</b> (or <b>s2</b> ).	
Effect Link		<b>B</b> affects <b>A</b> .	Same as the input-output link pair.	

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

Enabling and Transforming Procedural Links				
Name	Symbol	OPL	CPN(A)	CPN(B)
Definition	Process <b>B</b> changes the state of Object <b>A</b> ; the details of the effect may be added at a lower level.		It will always be replaced with input-output link pair after the hierarchical decomposition of place in CPN.	
Event, Condition, And Invocation Procedural Links				
Instrument Event Link		<b>A</b> triggers <b>B</b> . <b>B</b> requires <b>A</b> .		
Definition	Existence or generation of object <b>A</b> will attempt to trigger process <b>B</b> once. Execution will proceed if the triggering failed.		Place A: class A (existence). A token on A: an instance of A. Color set COLA: set of possible instance values of class A. Arc variable oe: the color set to be bound.	
State-Specified Instrument Event Link		<b>A</b> triggers <b>B</b> . when it enters <b>s1</b> . <b>B</b> requires <b>s1</b> <b>A</b> .		
Definition	Entering state <b>s1</b> will attempt to trigger the process once. Execution will proceed if the triggering failed.		Place A_s1: the s1 state of class A. A token on A_s1: an instance of class A (i.e., object A) at state s1 Color set COLA: set of possible instance values of class A. Arc variable os: the color set to be bound.	
Consumption Event Link		<b>A</b> triggers <b>B</b> . <b>B</b> consumes <b>A</b> .		
Definition	Existence or generation of object <b>A</b> will attempt to trigger process <b>B</b> once. If <b>B</b> is triggered, it will consume <b>A</b> . Execution will proceed if the triggering failed.		Same as consumption link. Same as consumption link.	

Table 5.3. Syntax and Semantics of OPM and its Mapping to CPN (cont.)

Event, Condition, And Invocation Procedural Links				
Name	Symbol	OPL	CPN(A)	CPN(B)
State-Specified Consumption Event Link		<b>A</b> triggers <b>B</b> when it enters <b>s2</b> . <b>B</b> consumes <b>s2A</b> .		
Definition	Entering state <b>s2</b> will attempt to trigger the process once. If <b>B</b> is triggered, it will consume <b>A</b> . Execution will proceed if the triggering failed.		Same as the mapping for state-specified consumption link.	Same as the mapping for state-specified consumption link.
Condition Link		<b>B</b> occurs if <b>A</b> exists.		<pre>colset COLA = unit; var oe: COLA;</pre>
Definition	Existence of object <b>A</b> is a condition to the execution of <b>B</b> . If object <b>A</b> does not exist, then process <b>B</b> is skipped and regular system flow continues.		Same as the mapping for instrument event link	
State-Specified Condition Link		<b>B</b> occurs if <b>A</b> is <b>s1</b> .		<pre>colset COLA = unit; var os: COLA;</pre>
Definition	Existence of object <b>A</b> at state <b>s2</b> is a condition to the execution of <b>B</b> . If object <b>A</b> does not exist, then process <b>B</b> is skipped and regular system flow continues.		Same as the mapping for state-specified instrument event link	
Invocation Link		<b>B</b> invokes <b>C</b> .		
Definition	Execution will proceed if the triggering failed (due to failure to fulfill one or more of the conditions in the precondition set).		Add place Completion_Trigger_Event between transition <b>B</b> and <b>C</b> to signal the end of the former and the triggering of the later. Color set EVENT: a class of system level message object.	

As an executable language, CPN is much more restrictive than OPM in terms of expressing execution semantics. For the execution semantics of the enhanced OPM model to be effectively captured by CPN, some styling rules should be applied when developing the OPM model. Before introducing the styling rules, a theorem regarding the equivalence of state and attribute needs to be established first.

**Theorem:** An attribute and a set of states can be inter-exchanged when modeling. The set of states of an object can be modeled as an attribute known as a state attribute here. In doing so, the set of states becomes the set of possible values for the attribute and vice versa.

Based on this theorem, the following styling rules for developing the OPM/H can be applied:

1. The developer can model a thing either as a state or as an attribute according to the needs of expressing the execution semantics. For example,

(1) If an object with states needs to connect to a process with a procedural link, it is better to model the set of states as a state attribute. Furthermore, if these set of states being replaced by the state attribute were connected to other process(es) using procedural link(s), These links will be redirected to the newly created state attribute. Accordingly, appropriate link conditions should be set so that these links are only active upon a particular value (corresponding to the state that the link originally connected to) of this newly created state attribute.

(2) An attribute object with states is not recommended. Such objects is usually created when an object has more than one set of overlapping states (i.e., the object can simultaneously be at more than one state, each of which come from a state set). In such case, the normal solution would be to group these states into groups and creating an attribute object to contain each group of states. However, such attribute object will have problem mapping to CPN because a token representing the object cannot be at more than one places (corresponding to the overlapped states) simultaneously. Therefore, the recommended solution is to keep only one group of states and model the other groups of states as attributes in the same way as the one presented in example 1 above. Alternatively, the designer can redefine these states and create a new set of states what is the cross product the states from each group.

2. Each enabling / transforming procedural link or event / condition procedural link connected to a zoomed-in OPM process must also connect to an OPM process enclosed in the zoomed OPM process.

3. It is highly recommended to take advantage of the flexibility of token definition in CPN. For example, it is highly recommended to identify alternative components only by a set of attributes (so as to model alternatives using tokens) rather than identify them by a group of elements organized in a certain structure. It is easy much easier to create an alternative object by changing the values of its attributes than creating a group of elements with a different structure.

**5.1.4. The Roles of CPN in Architecture Modeling and Analyses.** As discussed in Section 5.1.3., in the holistic modeling approach proposed here, the formal system model is specified by OPM and the mapping to CPN is conducted only when needed. However, CPN is very useful in many cases. A significant advantage of CPN is that the same model for system modeling can also be used to check the logical or functional correctness of a system and for performance analysis. There are a large number of analysis methods and software tools developed for Petri net models [191], [49], [192]. These methods share a lot in common but may differ in the type of Petri net supported. The discussion here focuses on the analysis methods for CPN. Many algorithms and their software implementations are developed for analyzing CPN. Such facilities include support for collecting data during simulations, for generating different kinds of performance-related output, and for running multiple simulation replications [171]. Note that there is a distinction between modeling the behavior of a system and monitoring the behavior of a model. Therefore, for model analysis purpose, auxiliary CPN constructs are allowed to be added to the original CPN model without affecting the behavior of the model. The roles that CPN plays in architecture modeling and analysis in the search-based architecture development framework include simulation, performance analysis, and system verification and validation. The details are discussed as follows:

Simulation of a CPN model allows user to examine the enabling of transitions and flow of tokens step wise or fully automatically. Such token flow information can be used to examine the behavior of the model, e.g., check whether the system behavior as modeled is expected, or derive performance related information. With the aid of software

tools it is also possible record the simulation process in the form of trace history (sequence of fired transitions and their bindings) and state history. For example, the simulation report [170] of the CPN Tools provides such information.

Since stochasticity is almost always involved in a CPN model and one simulation run can only generate one occurrence sequence out of many possible ones, the result obtained from one simulation run is usually not enough to reflect the true performance measures. Therefore, many of the software tools for simulating Petri net model support batch simulation, i.e., running multiple independent simulations automatically. Data can be automatically collected and saved during each simulation. A proper formulated simulation scheme allows conducting experiments on the system behavior as modeled give certain test cases of scenarios. Such experiments can be used for example to evaluate and estimate the performance measures, to compare different system configurations, to choose appropriate values for parameters of certain system components, to derive certain system properties for performance analysis purpose, or to obtain confidence intervals [193].

CPN models and their simulations contain detailed quantitative information about the performance of a system, such as throughput, processing time, queue lengths, and resource utilization, which can be extracted to support the investigation and discovery of structural and dynamic system properties [171]. The size, complexity, and time concept for CPN prohibit the generation and solution of analytical models from CPN models [171]. Therefore, performance analysis using CPN must rely on extracting from simulation the information needed for deriving performance measures of the system being modeled. The major source of such information is contained in the token values and number of tokens at some particular places of the model, the state of the system as marked by the entire set of tokens as well as from the events that occur (fired transitions and their bindings) during simulations [171]. There are a variety of ways to extract such information from the simulation of a CPN model. A simple way to do so is to add report places [170] to the CPN model. Such places collect historical information about the simulation runs without influencing the simulation [170]. Software tools like the CPN tools also support an advanced way for collecting data called monitors. A monitor is a mechanism that is used to observe, inspect, control or modify a simulation of a CPN

without having to modify the model [171]. A monitor can examine both the states of the model and the events that occur during a simulation [171]. The CPN tools provides a set of stand monitors such as simulation breakpoint monitor (place contents monitor and transition enabled monitor), data collector monitor, count transition occurrences monitor, list length monitor, and write-in-file monitor. It also supports user-defined monitors. Details regarding these monitors can be found at [171], [172]. Advanced CPN software like the CPN Tools also supports the generation of various kinds of simulation output such as log files, statistical reports, and scripts for plotting data values [194].

In another hand, certain attributes can be included in token values to encode required information for deriving performance metrics. For example, time attributes can be included in token values to record time-related information such as the cumulated time that a token spent at a certain place. CPN can be viewed as information processing system with operands being tokens, the value of which can be changed by expressions specified by arc inscriptions or code segments associated with transitions. Therefore, very rich information can be encoded in token values.

With data extracted, various performance measures can then be computed. What information to be extracted from the simulation and how to compute performance measures is problem dependent.

In some cases, simulation may not be the only way to compute certain performance measures; other methods, such mathematical equations, may also be used. However, it is possible that developing such a mathematical model might be much more difficult than constructing a CPN model and then deriving the performance metric using simulation. In such cases, simulation might be a better alternative for performance analysis, especially when creating and simulating a CPN model add no additional efforts (if they are also needed for computing other performance measures) while developing a mathematical model does. Furthermore, simulation based performance analysis is more robust. Changes in a CPN model directly result in changes in behavior and thereby changes in the simulation result. On the other hand, if a CPN model is changed, the mathematical model developed before might not be valid anymore and thus need to be redeveloped.

Another branch of formal analysis methods for Petri is concerned using generating and solving analytical models, such as continuous-time Markov chains, for performance analysis [192]. Although analytical models can provide exact solutions regarding the performance of a model they are subject to the state explosion problem [171].

Rigorous validation and verification of system specifications requires executable models. The use of CPN model and simulation adds an additional level of verification and validation. The rationale is that an architecture will not be fully operational until all components and their interconnections are properly specified and all terminology, definition, and data exchange syntax are consistent. Therefore, by generating a CPN model from the OPM model and simulating it, the developed system model can be verified in the architecture development phase. Consequently, the designers might go back and forth several times between OPM and CPN when developing the architecture model. Some available techniques for model verification and validation are discussed briefly as follows:

Validation can be achieved in many cases by simply observing the simulation result and check, for example, whether the CPN terminates at the desired state (for terminating systems [194]), reaches the right steady-state (for non-terminating systems [194]), gets the expected tokens at certain places. The logic correctness can be examined by testing each step of the simulation to ensure that the model follows the desired logic. The behavior of the system such as precedence relations amongst events, concurrent operations, appropriate synchronization, freedom from deadlock, repetitive activities, and mutual exclusion of shared resources [195] can be observed directly from the simulation. However, it is often beyond the capability of human beings to observe the details of a simulation by watching the enabled transitions and markings at each simulation step. The information extraction techniques mentioned in the performance analysis can be used to examine the behavior more efficiently. More information and details regarding these techniques can be found at [170–172], [186], [196], [197]. Particularly, when all conditions and events of a system are specified correctly in a CPN model, the simulation of the model should proceed with an expected sequence of state transitions. The system design can thus be verified by comparing the behavior as modeled with the desired



behavior. If the comparison shows a match, the model can be verified and validated. If the match is insufficient, then either the architecture model must be modified to better represent the system, or the system architecture must be reconfigured to better satisfy the requirements.

On the other hand, model behavior validation also leads to result validation, thereby result in increased confidence in the performance measured obtained. This is another advantage of using simulation to calculate performance measures over mathematical equations because mathematical equations developed for calculate performances measures also need to be validated.

Furthermore, dynamic properties characterize the behavior of a CPN and are often rather difficult to verify. Some most used dynamic properties are introduced in [186]. For example boundedness properties (the number of tokens can exist at a particular place), home properties (markings or sets of markings to which it is always possible to return), liveness properties (a set of binding elements  $X$  remains active), and fairness properties (how often the different binding elements occur), to name a few. More details about these properties can be found at [186]. A much more complete set of dynamic properties of a CPN can be found in Chap. 4 of [197]. More formal analysis methods that can be used to prove dynamic properties include state space analysis (or occurrence graphs, which illustrate all reachable markings) and place invariants (to construct equations which are satisfied for all reachable marking) [186], [197].

Simulations can only cover a finite number of execution sequences of a CPN model out of potentially many possibilities. Formal verification of system behavior requires examining all possible states. The state space analysis provides such capability. A full state space can be expressed by a directed graph with a node for each reachable marking and an arc for each occurring binding element [186]. However, such graph can be too large to construct even for a small CPN. This is a major drawback of the state space analysis called state space explosion [170] and it make state space analysis of limited usage in some cases. A number of methods have been developed to alleviate the state explosion problem as indicated in [170], [198], [199].

CPN model and its simulation can also help in identifying missing specifications and requirements during the architecture development phase because an incomplete

model is not executable. Missing requirements, in this context, are functions or capabilities not yet specified but needed, without which, the system cannot generate the expected behavior or performance.

Note that not all of these techniques discussed above need to be applied during the architecture search process when using the search-based architecture development framework. It is more appropriate to carry out some detailed analyses after limited number of near-optimum solutions are obtained by the search process and when designer need to choose one final solution out of those alternatives.

## **5.2. ARCHITECTURE GENERATION**

By following the methods described in Section 5.1.3, a generative class model for the system of interest can be developed. Such generative class model describes a collection of models rather than a single instance. Accordingly, an architecture alternative generation mechanism is needed. Such mechanism should support the generation of all instance models that coverer the entire design space as defined by the generative class model. The architecture generation mechanism proposed here includes both a set of architecture alternative generation operations that apply to various levels of model constructs and an automatic generation mechanism that enumerate all possible instances covered by the design space.

**5.2.1. Architecture Alternative Generation Operations.** The generation of architecture alternative is guided by the design space as specified by the generative class model. Elements in an architecture model can be divided into variable part and common part. The common part is shared by all architecture alternatives. The variable part differs from architecture to architecture. The architecture alternative generation is only concerned with generating variable part so it is also known as architecture variant generation. Each set of generated variable elements is then combined with the constant part to form a complete architecture alternative. Architecture alternative generation operations (or variant generation operations in short hereafter) work on three levels. The most fundamental level operation applies to a single element. Structural generation operations work on a set of related elements with different types. The system level

operation forms a complete variable part based on the generated variable elements and eventually combined with the common part to form a complete architecture alternative.

**5.2.1.1 Generate element instances.** The most fundamental variant generation operation is concerned with a single model element that can be abstract as an object from the object-oriented sense. An element can be any OPM/H construct (i.e., elements in  $Sys_k$  as defined in Section 5.1.3.1) since every OPM construct is an instance of the corresponding class in the metamodel of OPM/H. This operation is fundamental because it is used in all other variant generation operations proposed here. There are two steps of the operation:

Step 1: generate all possible instances of an element (class) according to the constraints of its properties. Particularly, generate all possible values of a property for an element according to the constraints of that property. Then generate the whole set of instances for the element using the cross product of the generated property values but eliminating those invalid combinations according to the constraints.

Step 2: generate a set of such element instances (i.e., duplicate the generated instances) according to the participation cardinality constraints.

**5.2.1.2 Generate structural variants.** The second level variant generation operations include a set of primary operations, side-effect handling, and advanced group operations. There are two primary operations. One adds/removes/modifies links between distinct entities (objects, processes, or states) in the system model (Operation 1). The other changes the set of entities (objects, processes, or states) in the system by either adding entities to or removing entities from the system (Operation 2). These operations can be applied, in turn, with Operation 1 preceding Operation 2 in each cycle if both changes are needed to create a variant. The procedures of these two operations are as follows:

**5.2.1.2.1 Add/remove/modify links – operation 1.** The following conditions are given:

(1) The system is currently specified by

$$Sys_k = \{O_k, P_k, S_k, R_k, M_{k,0}\}$$

where the subscript  $k$  indicates the configuration of the model is after its  $k^{\text{th}}$  change.

(2) The set of links to be removed from the system is  $L_k^r$

(3) The set of links to be added to the system is  $L_k^a$

After the  $(k+1)^{\text{th}}$  change, the system will be specified by

$$Sys_{k+1} = \{O_{k+1}, P_{k+1}, S_{k+1}, R_{k+1}, M_{k+1,0}\}$$

such that:

$$(1) L_{k+1} = L_k - L_k^r \cup L_k^a$$

This equation shows the changes of the links in the system when some links are added and/or removed. Modifying a link is equivalent to removing a link and then adding a link with the same Source and Destination properties but different values for other properties. Note that a link is an instance of the link class from the metamodel of OPM/H as defined in Section 5.1.3.1. Therefore, along with adding a link, both the properties of the link and their associated constraints should also be specified at the same time. Similarly, removing a link also removes the properties, along with their constraints, from the link.

$$(2) E_{k+1} = E_k - E_k^i \cup E_k^u$$

where  $E \in \{O, P, S\}$  is an entity (object, process, or state) and  $E_k^i \subset E_k$  is a subset of entities that are isolated from the system because all of their links with other entities ( $E_k - E_k^i$ ) are removed during the change.  $E_k^u \subset E_k^i$  is the subset of isolated entities to be reconnected to the system. The  $E_k^i$  can be expressed as:

$$\forall E_{k,i}^i \in E_k^i: [L_{k,j}(\text{Source}) \neq E_{k,i}^i \wedge L_{k,j}(\text{Destination}) \neq E_{k,i}^i, j = 1, 2, \dots, I_{l,k}]$$

where,  $L_{k,j}(\text{Source})$  and  $L_{k,j}(\text{Destination})$  are the values of the Source property and Destination property of link  $L_{k,j}$ , respectively.

$$(3) I_{k+1} = I_k - I_k^i + I_k^u$$

where  $I \in \{I_o, I_p, I_{si}\}$  is the number of either objects, processes or states in the system and  $I_k^i$  is the number of isolated entities and  $I_k^u$  is the number of entities to be reconnected to (i.e., kept in) the system.

(4)  $M_{k+1,0} = M_{k,0} - M_{k,0}^i + M_{k,0}^u$ , where  $M_{k,s}^i$  is the marking on the isolated objects or states (i.e.,  $O_k^i$  or  $S_k^i$ ) and  $M_{k,s}^u$  is the marking on the objects or states to be reconnect to (i.e., kept in) the system (i.e.,  $O_k^u$  or  $S_k^u$ ).

**5.2.1.2.2 Add/remove/modify entities - operation 2.** Here entities include object processes, and states. The following conditions are given:

(1) The system is currently specified by

$$Sys_k = \{O_k, P_k, S_k, R_k, M_{k,0}\}$$

(2) The subset of entities (objects, processes, and states) to be removed from the system is

$$E_k^{re} = (E_{k,i}^{re}, i = 1, 2, \dots, I_k^{re}) \subseteq E_k$$

where  $E \in \{O, P, S\}$  is an entity (object, process, or state) and  $E_{k,i}^{re}$  is the  $i^{\text{th}}$  entity to be removed and  $I_k^{re}$  is the total number of entities to be removed. Accordingly, the links with entities in  $E_k^{re}$  as either Source or Destination should also be removed. These links to be removed can be expressed as follows:

$$L_k^{re} = (L_{k,j}^{re}, j = 1, 2, \dots, I_{l,k}, L_{k,j}(\text{Source}) \in E_k^{re} \vee L_{k,j}(\text{Destination}) \in E_k^{re})$$

(3) The subset of entities to be added to the system is

$$E_k^{ae} = (E_{k,i}^{ae}, i = 1, 2, \dots, I_k^{ae}) \not\subseteq E_k$$

Where  $E \in \{O, P, S\}$  is an entity (object, process, or state) and  $E_{k,i}^{ae}$  is the  $i^{\text{th}}$  entity to be added and  $I_k^{ae}$  is the total number of entities to be added. Along with adding entities, links connecting these entities to either the existing entities or the newly added entities can be added as well. These links are denoted as  $L_k^{ae}$ .

After the  $(k + 1)^{\text{th}}$  change, the system will be specified by

$$Sys_{k+1} = \{O_{k+1}, P_{k+1}, S_{k+1}, R_{k+1}, M_{k+1,0}\}$$

such that:

$$(1) E_{k+1} = E - E_k^{re} \cup E_k^{ae}$$

This equation shows the effects of both removing and adding entities. Modifying an entity is equivalent to removing an entity and then adding an entity with the same values for the “Name” property but different values for other properties. Note that an entity (object, process, or state) is an instance of the corresponding class from the metamodel of OPM/H as defined in Section 5.1.3.1. Therefore, along with adding an entity, both the properties of the entity and their associated constraints should also be specified. Similarly, removing an entity also removes the properties, along with their constraints, of the entity.

$$(2) M_{k+1,0} = M_{k,0} - M_{k,0}^{re} + M_{k,0}^{ae}$$

where  $M_{k,s}^{re}$  is the marking of either objects or states to be removed from the system (i.e.,  $O_k^{re}$  or  $S_k^{re}$ ) and  $M_{k,s}^{ae}$  is the marking of the objects to be added to the system (i.e.,  $O_k^{ae}$  or  $S_k^{ae}$ ).

$$(3) I_{k+1} = I_k - I_k^{re} + I_k^{ae}$$

where  $I_k^{re}$  and  $I_k^{ae}$  are the numbers of entities to be removed and added, respectively.

$$(4) L_{k+1} = L_k - L_k^{re} \cup L_k^{ae}$$

This equation shows the effects of both removing the links with entities in  $E_k^{re}$  as either Source or Destination values and adding links along with adding entities.

**5.2.1.2.3 Side effects handling.** The variant generation operations introduced above indicate that the order of implementing the operations defined in Operation 1 and Operation 2 matters. Both isolated objects and dangling links need to be cleaned up to prevent side effects. Additional rules also apply when removing objects connected by structural relations. The related scenarios are handled by applying Operation 1 and Operation 2 in an appropriate order. These rules, and the methods to handle related scenarios, are summarized as follows.

(1) Removing the source (or root, parent) object also removes its destination (or leaf, child) objects for an object connected with a group of objects using aggregation-participation links, exhibition-characterization links, or classification-instantiation links. Therefore, if any of the child objects are to be preserved, their corresponding links with the parent object must be removed before the parent object is removed.

(2) Removing a source (or root, parent) object also removes the attributes, structure, procedure, and state inheritance [37] of all destination (or leaf, child) objects for objects connected to it by generalization-specialization links.

**5.2.1.2.4 Advanced operations.** Some advanced variant generation operations can be constructed using the primary operation (i.e., Operation 1 and 2) defined above. With the primary operation, the system can be expanded or shrunk horizontally by adding or removing entities or links. In contrast, the advanced operations are concerned with connecting things to a root thing to achieve vertical scalability (i.e., either refinement or its reverse, aggregation). Such advanced operations are summarized as follows:

(1) *Object decomposition*. Object decomposition is achieved by adding a group of entities (possibly linked) and connecting them with the chosen root object using aggregation-participation links. Additionally, appropriate links between these new entities and the existing ones can be added. This is the scenario for adding a sub-system.

(2) *Process decomposition*. Process decomposition is achieved by adding a group of entities (possibly linked) and connecting them with the process to be decomposed using exhibition-characterization links. Then redirect (via remove and add) the existing links from connecting the process to connecting appropriate entities just added.

(3) *Aggregation*. Aggregation is achieved by adding a new thing (as root, parent or source) and connecting it with related things in the system using aggregation-participation links. This is the scenario for grouping existing system components to create a new subsystem.

(4) *De-Aggregation*. De-Aggregation is the reverse process to the Aggregation operation. It is achieved by first removing the aggregation-participation links between the root (or source, parent) object and its leaf (or destination, child) objects. The root (or source, parent) object is then removed.

(5) *Breakout*, i.e., replacing a single thing with a set of things. Breakout is achieved by both removing and adding things. Additionally these newly added things can be connected to the existing entities using appropriate links. This is the scenario for using a set of components to achieve the same functionality as the one achieved by a single component.

(6) *Merge*, i.e., replacing a set of things with a single thing. This is the reverse process to the Breakout operation. It is also achieved by both removing and adding things and possibly followed by adding links. This is the scenario for using one component to achieve the same functionality as the one achieved by a set of components.

**5.2.1.3 Generate full architecture alternative.** In order to generate the entire variable part of an architecture alternative, the above defined variant generation operations should be applied to each applicable dimension of the design space as specified by the generative class model. The entire variable part of an architecture alternative can then be generated by applying the step 1 of the fundamental variant generation operation defined in Section 5.2.2.1 to the entire variable part. In this case the

entire variable part can be viewed as a class. Accordingly, the dimensions of the design space can be viewed as the properties of the class. Finally, the generated variable part is combined with the common part to form a complete architecture alternative.

**5.2.2. Automatic Generation of All Architecture Alternatives.** An automatic mechanism that can enumerate an entire set of architecture alternatives, in a systematic way, is always desired, especially when the design space is either very large or very complicated. In the research of automated analysis of feature model, several operations of analysis on feature models have been proposed. These operations can be utilized for both generating architecture alternatives and analyzing a generative class model since an OPM/H model contains feature model information. Among these operations, there is one known as “all products” (or “all valid configurations”, “list of products”), which is defined for generating all variants of a feature model. Particularly, this operation takes a feature model as input and returns all of the products represented by the feature model. The “product” in this context is the complete set of features to be selected [14]. Various implementations of these feature model analysis operations based on a variety of paradigms have been proposed as summarized in [14]. However, tool support of these analysis operations is still inadequate. The work presented in [184] is the only one, found so far, that supports the analysis of extended feature models (i.e. including feature attributes). In [184], feature model analysis operations are implemented by translating a feature model into a Constraint Satisfaction Problem using a set of mapping rules. An implementation framework known as FAMA (FeAture Model Analyser) is presented in [200]. FAMA integrates some of the most commonly used logic representations and solvers proposed in the literature into one comprehensive tool suite. It is claimed to be the first tool integrating different solvers for the automated analyses of feature models. The extended feature model, however, supported by FAMA implementation includes feature attributes only (i.e., no support for complex constraints among attributes or features). For example FAMA struggles to address the input link to a feature decorated by either XOR or OR join. If such links are connected to a subfeature, a duplicated feature violation will result. If such links are “requires” links, it is not supported by the implementation. Due to the complexity of architecture modeling, an extended feature model with both feature attributes and constraints (among attributes or features) are, in many cases, needed.



Alternatively, various ad-hoc approaches for automatic generation of all architecture alternatives can be developed using CPN. Such approaches can be based on the idea that tokens of CPN can be used to record the trace of states or transition along the path that tokens travel during execution. One of such approach is proposed here. The purpose of this approach is to explore process related alternatives, for example, exploring alternative execution paths or determining whether to use an optional object according to the processes to be included in the system. The details of this method are as follows:

- The OPM processes, objects, or states are mapped to CPN transitions or places the same way as the mapping methods presented in Section 5.1.3.2.
- Execution path and required objects can be recorded in token values, which are set as list types. For each transition a token travels through, add values of all input tokens and the name of the transition to the tokens sending out from the transition. A token stops traveling at a place with no outgoing arcs.
- The “requires” or “excludes” constraints are encoded in the guard of related CPN transitions. The expression of a transition’s guard can be used to check the existence of certain values in the input tokens of the transition. For example if a process needs to exclude something, the guard can be set to false given that the input token of the transition contains a value corresponding to the thing to be excluded.

Such an approach requires the identification of an initial CPN place, where all variations originate or equivalent to the root node of the corresponding feature model. Additionally, there must also be a limited set of end places with no outgoing arcs (corresponding to leaf nodes of the tree-structured feature model). All of the generated alternatives (represented by tokens) can be collected at these end places after a simulation run of the CPN model. The collection of these tokens represents the alternatives discovered. Case study 2 provides an example of applying the above suggested approach.

This section identified the need of the holistic modeling and proposed combining OPM, CPN and feature model to achieve such holistic modeling. The architecture alternative generation mechanism was also developed based on the proposed modeling approaches. The search-based architecture development process requires automating the alternative generation, architecture evaluation, and optimization process. Therefore, a

software implementation of the proposed approaches is needed. Such implementation should integrate the development of holistic system models, the generation of architecture alternatives, the calculation of performance metrics, and the search for optimum solutions into one coherent process. Such implementation is presented in next section.

## 6. GENERIC IMPLEMENTATION

This section presents the software implementation of the proposed approaches. This implementation covers the holistic system model development, part of the alternative generation, the model simulation, the performance analysis, and the optimization process. The overall architecture of the programs developed is first presented, followed by a summary of the workflow of related activities. This workflow integrates the programs developed and related activities into one coherent problem solving process. The design rationale and implementation strategies of some major code modules are also provided.

The entire implementation of the proposed approaches and their application in solving the first sample problem, the design of reconfigurable manufacturing system, is written in Python 2.7.3. Python is chosen as the programming language because there are a huge number of open source libraries available in Python. This implementation uses two of them: the SNAKES package [187], for its CPN support, and the Inspyred package [201], for its GA support. SNAKES is a general Petri net library implemented in Python. It provides the necessary components to create, edit, and execute many sorts of Petri nets. It also supports state-space construction. The Inspyred library contains a set of modulus for implementing various types of evolutionary computations and swarm intelligence. The library separates problem-specific computation from algorithm-specific computation thus making it easy for users to integrate GA computation into their own code. Nevertheless, extensive modifications to these libraries are made to achieve the capabilities required for the implementation in this research.

### 6.1. PACKAGE ARCHITECTURE

The overall program implementation strategy can be visualized as a layered architecture as shown in Figure 6.1. The top layer is the user interface. Modules in this layer allow user to specify the input information (such as part, machine and processing information for the RMS problem), system models, analysis models, control parameters of the genetic algorithms and the overall process, and output processing and archiving. The bottom layer contains the components for alternative generation, chromosome

encoding, candidate simulation and evaluation. The middle layer calls the services provided by the facilities in the bottom layer and organize them into a coherence search process in searching for good alternatives. Note that, in Figure 6.1, the lighter shaded blocks at the bottom layer are from the external libraries. However, amount them, the components denoted in bold text are heavily modified for this research. The rest are developed from scratch for this research.

User Interface				
Problem specific input (e.g., Part, machine and processing information)	Parameter settings for overall process, GA, and output processing & Archiving	Analysis model building	CPN model specification in ABCD language	
Search Process				
Alternative generation and chromosome encoding (e.g., RMS configurations)	GA computation	Objective functions	OPM Core	ABCD Parser, <b>simulator</b> , PN markup language translator, visualizer
	Variation operator ( <b>crossover</b> and <b>mutation</b> ) and other components			CPN Core

Figure 6.1. Components of the Software Implementation

The *snakes.nets* module as the core element implementing the Petri net in the SNAKES library only provides the capability to execute a one-step firing of a selected transition with a selected binding. The full simulation capability is provided by the ABCD simulator in the *abcd* plugin which also provides a simple Graphical User Interface (GUI). The graphical ABCD simulator requires specifying a Petri net model

using the ABCD (Asynchronous Box Calculus with Data) language whose semantics is given in terms of CPN. The syntax of ABCD is a mix between Python and a process algebra. The *abcd* utility provides the parser to create a computational model of Petri net using the *snakes.nets* module from the textual Petri net model specified in ABCD. However, there is very little or no documentation regarding this ABCD language. The semantics and syntax of ABCD can only be inferred from two files used by the ABCD parser: *snakes/lang/abcd/abcd.pgen*, which contains the concrete grammar, and *snakes/lang/abcd/abcd.asdl*, which provides the abstract syntax. Due to the limited information and knowledge regarding the ABCD language, the code to transform an OPM/H model to a CPN model has not been developed yet.

Based on the search-based architecture development framework introduced in Section 4.1, using the multi-objective genetic algorithm as the optimization model, the workflow among various modules of the program implementation and activities of the designer is illustrated in Figure 6.2 using OPM notations. Such workflow is a concrete implementation of the search-based architecture development framework presented in Section 4.1.2 with GA as the search algorithms. The designer needs to involve in this problem-solving process through five activities as represented by the five blue shaded OPM processes in Figure 6.2. These activities are (1) developing a problem-specific data preprocessing module to handle input data according to the need of the system model developed for the system of interest, (2) developing the system model, (3) developing analysis models to compute various performance measures needed to assess the models, (4) developing the optimization model to conduct the search for optimum solutions, and (5) developing a decision model to choose one final solution out of a set of non-dominant solutions obtained from the optimization model. These activities are carried out according to the design requirements. A preprocessing process is needed to transform raw data into a format that the system model can use. Such pre-processing is problem specific. As indicated in Figure 6.2, each of these activities results in certain kind of models. Once these models and parameters are set, the search process can proceed in an automated way. The developed OPM/H class model also contains the specification to build a CPN model and the specification of design space in the form of feature model. A set of OPM instance models, along with the mapped CPN models, can then generated by the alternative

generation process. Such instance models are transformed to chromosome representations according to the encoding scheme developed in the process of developing the optimization model. Then the GA-based search process can proceed. The calculation of performance metrics needs to invoke the analysis models, which may involve the simulation of the corresponding CPN models. The search process stops when a user specified termination criterion is met. A set of non-dominant solutions can be obtained after the search process. Selected results can be saved to files besides displaying on the screen. The user can then use the developed decision model to select a final solution. This selection process can optionally be supported by more detailed analyses.

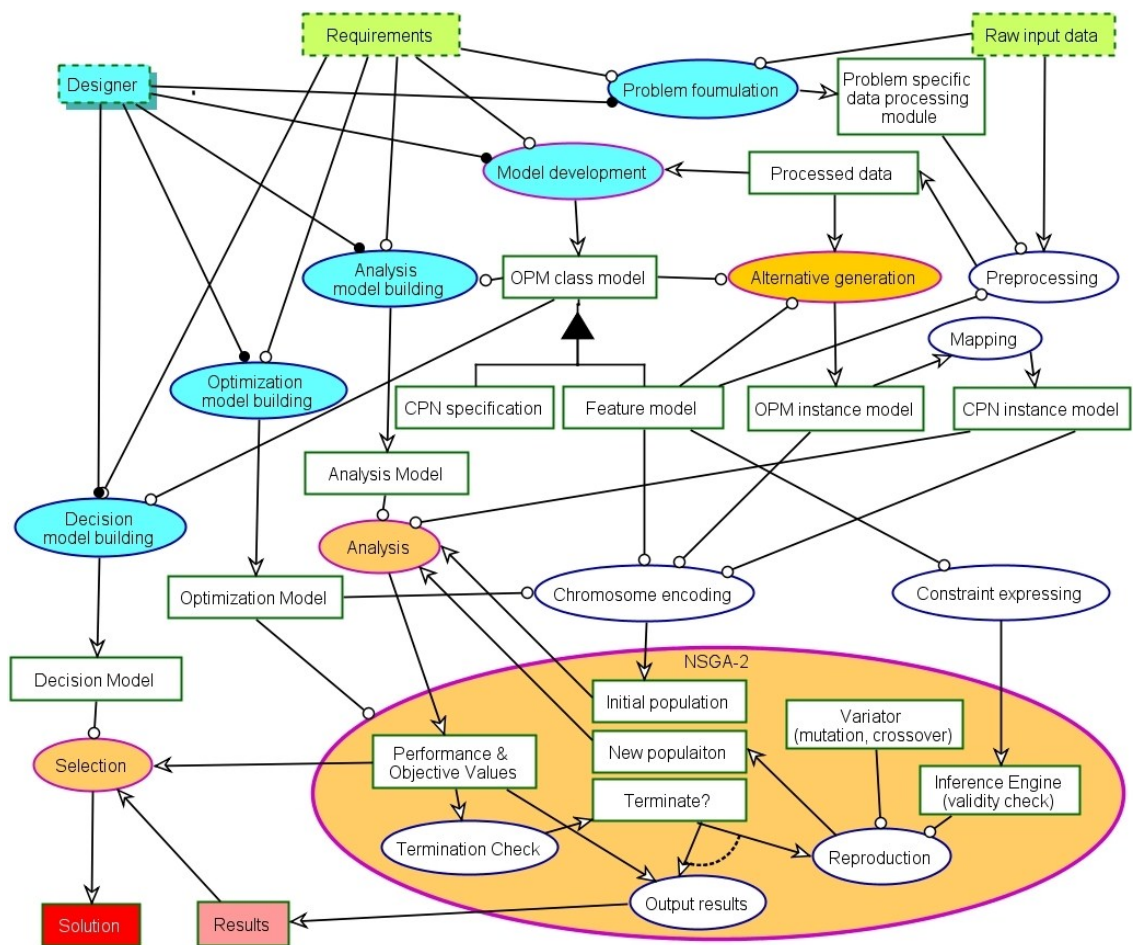


Figure 6.2. Workflow of the Implementation of the Search-Based Architecture Development Framework

## 6.2. MODULES

The program implementation consists of a set of Python modules. Some of them are developed from scratch; others are modified based on existing open-source libraries. Some major modules, along with their design rationale and implementation strategies, are summarized as follows (the italics in the parentheses following each module is the name of the corresponding Python module in the program package):

OPM/H module (*e\_opm*): This module provides the classes for both building and editing an OPM/H model. The class functions for adding, deleting, and modifying basic OPM/H constructs are also used to achieve the basic architecture generation operations.

CPN module (*snakes.nets*): This is the main Petri net module provided by the SNAKE library. However it is heavily modified to achieve the capabilities required in this research. The major modifications include:

(1) Support for timed CPN. The time semantics of the Petri net adopted here is the same as the one used in the CPN Tools [202]. Such semantics utilizes timed token and simulated clock to implement the time concepts in CPN. A timed token is a regular token attached with a number, called the time stamp. The simulated clock is a counter (globally available within an executing model) whose current value is the current abstract simulation time. A timed token is not available for any purpose unless the clock time is greater than or equal to the token's time stamp. When there are no enabled transitions, but there would be if the clock had a greater value, the simulator increments the clock by the minimum amount necessary to enable at least one transition. Therefore, the time stamp of a token can be interpreted as the time since when (in terms of simulated time) the token is available. The units of simulated time do not inherently represent any particular absolute time unit but can be interpreted as real time according to the subject being modeled [202]. Simulated time is sometimes referred to as model time [202].

Particularly, the timed token here is implemented as a Python tuple with the last element in the tuple being the time stamp. Such time stamp is a string type constructed by proceeding an integer representing the time with an “@”. For example, a regular token “10” with a time stamp of value “5” become “(10, '@5’)” when represented as a timed token. An interpreter is inserted into the *snakes.nets* module to translate such notations. The time information contained in the time stamp will be extracted.

In order to evaluate a time delay expression, a “`sys_time`” keyword is allowed to be used in an ABCD model. This keyword will be interpreted as the current simulated time within the `snakes.nets` module (particularly, within function `binding` of class `Expression` of the `nets` module).

(2) Allowing a transition to send empty tokens (i.e., do nothing,) to its output places. Therefore, the keyword “None”, which is a special Python data type frequently used to represent the absence of a value, is allowed in an expression of an ABCD model. Such capability makes it much easier to develop arc annotations (or arch inscriptions) in some cases.

ABCD simulator module (`simulngui` replacing `snakes.utils.abcd.simul`). The ABCD simulator provided by the SNAKES library only supports simulating the firing a single selected transition with a selected enabled binding using the ABCD simulator GUI (Figure 6.3). Details regarding this simulation GUI can be found at [203]. The optimization process in the search-based architecture development may invoke the simulation and require the simulation to proceed automatically until desired results are returned. Therefore, several enhancements are made to the original ABCD simulator and a new module called `simulngui` is created to replace the one. These enhancements and modifications to the original module are briefly summarized as follows:

(1) Adding an option to disable GUI. The users are given the option to disable ABCD simulator GUI to save computational time. The simulation is invoked automatically by the optimization model when a candidate needs to be evaluated. The simulation will be conducted numerous times in the entire search process conducted by GA. Therefore there is no need to show the GUI for each simulation in such case.

(2) Adding an option to do multi-step automatic simulation. The user can specify the maximum number of steps a simulation is to be executed. The simulator randomly chooses an enabled binding from a randomly chosen enabled transition and fires that transition. The simulation will stop when either there is no enabled transition or the maximum number of simulation steps has been reached. Such enhancement does not impair the original functionalities of the simulator. The user can still chose a binding from the list of enabled ones to fire a transition and observe the change of system state after firing that transition if the ABCD simulator GUI is turned on.



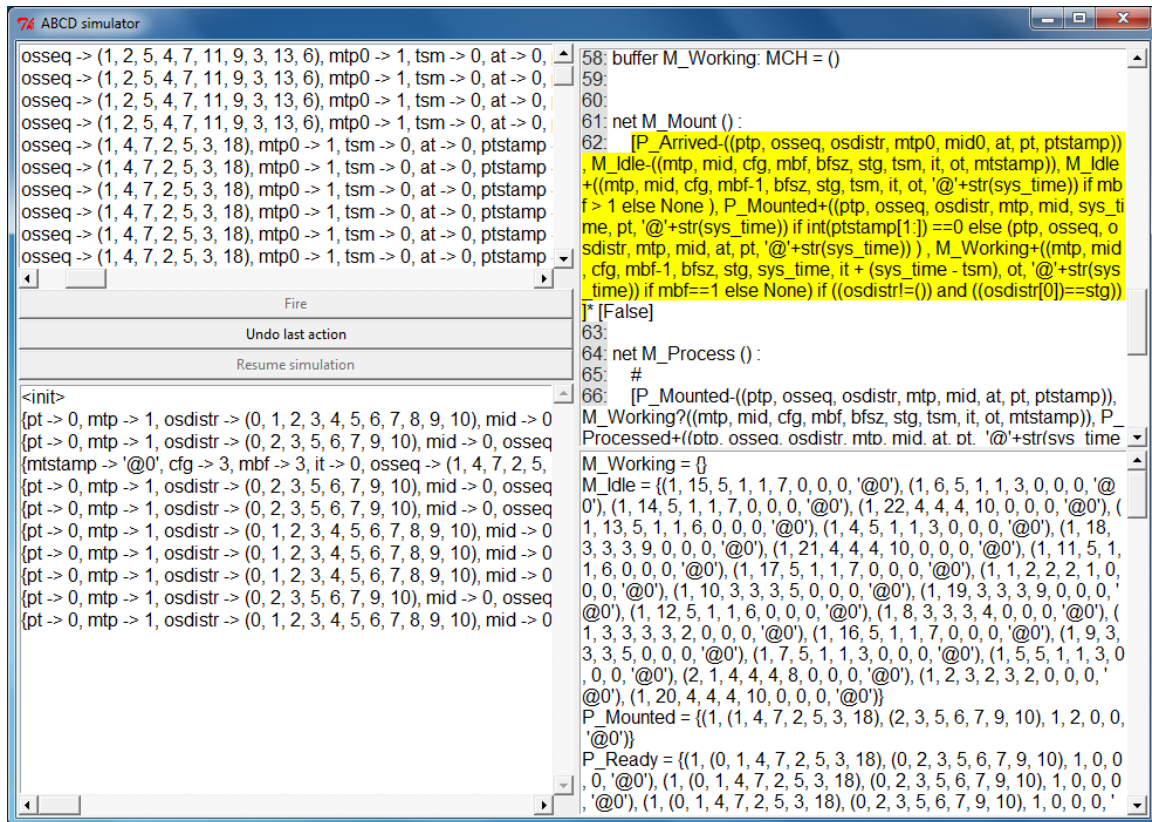


Figure 6.3. Illustration of the GUI of the ABCD Simulator

(3) Adding resuming capability. The *Resume simulation* button of the ABCD simulator GUI is redefined to include the multi-step simulation capability. Particularly, if after running an  $N$ -step automatic simulation, there is still enabled transitions, click this button will run another  $N$ -step automatic simulation.

(4) Storing state and trace histories. The marking of each simulation step can be stored in the state history, which is written in an out file named `filenameStatHistory.txt`, where “filename” is the file name of the ABCD model. The fired transition and its associated fired binding of each simulation step can also be stored in the trace history, which is written in an out file named `filenameTraceHistory.txt`, where “filename” is the file name of the ABCD model too. The state and trace history can also optionally be shown on screen during the simulation.

(5) Retrieving the simulation result. A function is added to the `Simulator` class of the `simulngui` module so that the simulation result can be retrieved in the form of final markings be external programs.

Compiler and simulation control module (`abcd_build_simu` replacing `snakes.utils.abcd.main`): The original main module of the `abcd` utility provided by the `SNAKES` library is designed to take a command line input, which contains the file name of a ABCD model, a set of options and parameters for those options, and to provide services according to the options. Such services include simulating a Petri net model, drawing a Petri net model and saving it as `.PNG` file using the `Graphviz` plugin [204] for Python, saving a Petri net model into one represented by the Petri net markup language, etc. All these services need to first have the input ABCD model compiled using the `parser` module. The result is a computational model of the Petri net model created using the `snakes.nets` module. The modification was made on the original `snakes.utils.abcd.main` module and saved as a new module named `abcd_build_simul`. This new module takes function arguments as input instead of from the command line. The simulation result, in the form of final markings, can also be returned as an augment return. Such modifications make it possible for other Python functions to call the services provided in this module within a Python thread instead of through command lines.

Variation operator module of the Inspyred library (`inspired.ec.variators.mutators` and `inspired.ec.variators.crossovers`). Candidates generated by any crossover (or mutation) operator provided in the `crossovers` (or `mutators`) module of the `Inspyred` library are subject to a validity check. Such check is added to the decorating functions of both crossover operators and mutations operators. If a candidate generated by the crossover operator is not valid, then redo the crossover operation (using a different pair of parents) until the validity check is passed. Similar procedure is gone through for the mutation operation too. The actual validity is checked by a function in an external module, i.e., problem-specific module. The result is then returned to the decorating functions of either crossover operators or mutation operators. This repairing mechanism makes sure that only valid candidates are evaluated and kept in

the population. This is necessary because evaluating a candidate might cost a lot of computation time and resources, especially when simulation is used.

Main module. The main module is the top level module through which the user controls the problem solving process. It, therefore, is highly customized. The major functionalities provided by this module include loading input data and base CPN model, choosing the optimization algorithm to be used and setting related parameters and options, executing the search process, showing and plotting results, and saving results to archive files.

Problem specific modules. The tasks needed to solve a problem varies from problem to problem. The most common functionalities to be supported include: (1) data preprocessing function that transforms raw data into the format required by the system model, (2) chromosome encoding (create chromosomes from the system models) and decoding (convert chromosomes into machine/human interpretable format for recreating system models) function, (3) candidate (chromosome) generation function, (4) analysis model development and candidate assessment function, (5) validity check function for generated candidates. Note that alternative generation is usually associated with candidate (chromosome) generation through chromosome encoding and decoding process.

This section presented the software implementation of the proposed approach. Such implementation is generic except for the data pre-processing part which must be problem-specific. Such implementation is applied to the design of RMS to demonstrate the usage of the proposed approach in solving real-world architecture design problem. The implementation details and test results are presented in the next section. As suggested in the workflow depicted in Figure 6.2, for solving a different problem, the user needs to develop a problem-specific data preprocessing module and also needs to develop the system model, the analysis models, the decision model and the optimization model according to the problem to be solved. The user then can control all the activities using the main module by setting options and parameter values.

The Python code developed for this implementation is enclosed in the attached CD. The contents of the files included are listed in Table C1.

## 7. APPLICATION DEMONSTRATIONS

This section uses two examples to demonstrate the application of the proposed approach and the developed software implementation. These sample projects are the configuration of RMS and the architecture design of a manned lunar landing system for the Apollo program (retrospective).

A full implementation of the proposed approaches is presented on the first example problem along with the test results. Such implementation is generic meaning that the code is capable of solving similar RMS configuration problem. Only some of the assignment expressions in the data input module needs to be updated according to the new raw input data. In the second example problem, the focuses are architectural model development and alternative generation. No optimization is actually conducted due to lack of data.

### 7.1. RECONFIGURABLE MANUFACTURING SYSTEM

The operation life of an RMS consists of more than one DP, each of which would have a specific duration and a corresponding demand scenario. The RMS is configured according to each demand scenario. The demand scenario under consideration here is characterized by multiple products with mid-to-large production volumes. For this scenario, the flow-line configuration proposed in [205] and used in [55], [206] is adopted here. Such an RMS is comprised of a set of stages each of which contains multiple identical stations/machines arranged in parallel with identical operation assignments.

Generally, the number of feasible configurations for a given DP is significantly large in an RMS. Therefore, a method is needed to find RMS configurations that are not only capable of meeting functional and capacity requirements of each DP but also have low cost, good performance and desired “ilities”. Therefore the RMS configuration is a constrained, multi-objective optimization problem. In addition, unlike conventional flow-line optimization that pursues the optimal solution, for RMSs, the goal is to find a set of solutions which include the optimal solution and near optimal solutions [55]. One reason is that the optimal configuration for the current DP may not be the best one considering the cost of reconfiguring previous configuration to the current one. Another reason is that

other system objectives and criteria (e.g., quality, convertibility, scalability) besides cost should also be considered in the selection of the best configuration [207], [208]. For the example problem to be solved here, only one DP and two quantitative objects, cost and production rate, are considered. The goal is to find a set of near-optimal solutions to be used in more in-depth analyses.

**7.1.1. Problem Definition.** In order to facilitate the benchmark comparison this dissertation adopted the case study used in [55]. The same problem definition is used except that multi-objective optimization is assumed in this dissertation. A second objective, maximizing production rate (or equivalently minimizing unit production time), is added in addition to the minimizing capital cost objective. This dissertation only provides a brief summary of the problem and some key data for clarity whereas some addition and modifications to the problem definition are described in detail. Readers are encouraged to refer to the original paper [55] for detailed problem definitions and data structures used to describe the problem. Related input data for designing the configuration the RMS are also extracted and presented in Appendix A.

Youssef and H. ElMaraghy [55] define the following core concepts to be used in describing an RMS:

An operation cluster setup (*OS*) is a set of one or more operation clusters (*OCs*) that can be performed together on a specific machine with a specific configuration. An operation cluster (*OC*) is a set of operations (*OPs*), which are always machined together with a specific order due to different types of constraints such as logical or datum tolerance constraints.  $MC_{ij}$  stands for machine configuration  $j$  corresponding to machine/station  $i$ . Only one feasible machine configuration (*MC*) can be assigned to a machine/station ( $M$ ) in a selected configuration.

Figure 7.1 shows an example of a selected configuration in a specific configuration period capable of producing two different types of parts within a part family. In Figure 7.1, there are two rows of *OSs* each representing the OS assignments to different stages for one of the two part types to be produced and the zeros mean that the stage is not used for that specific part type [55].

The input parameters and information assumed to be available include:

(1) Demand scenario, which specify the types of product to be produced by the RMS and their demand rates along with the configuration period [55].

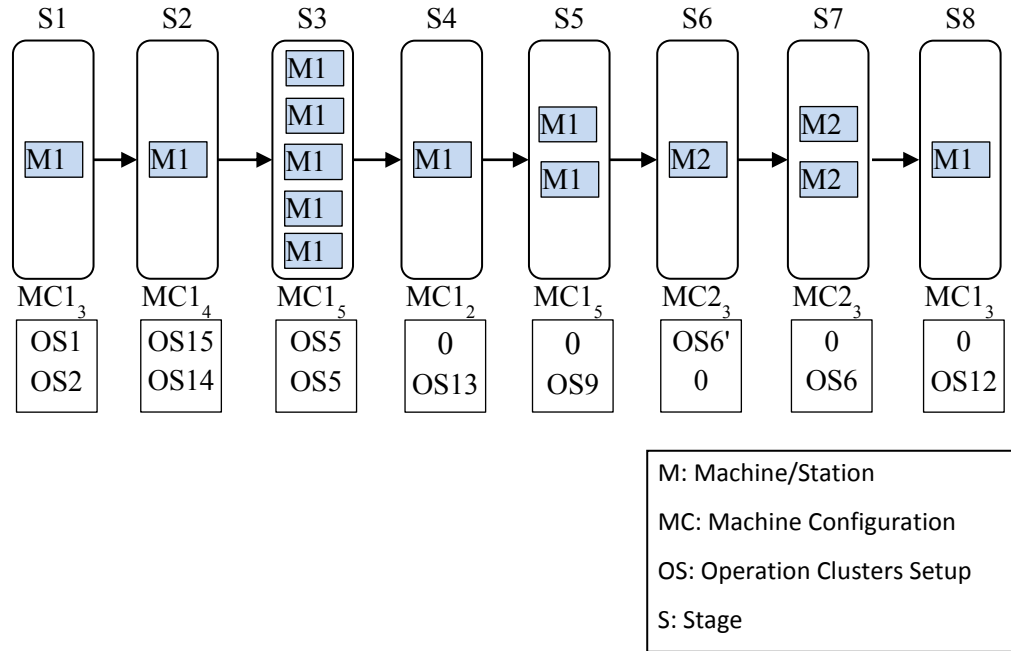


Figure 7.1. Example of a Selected RMS Configuration ([55])

(2) Parts processing information (*OPs*, *OCs*, *OSs* and *PGs*). *OPs* must be accompanied by operations precedence graphs (*PGs*) that define sequential constraints between the different *OPs* and subsequently between different *OCs* [55].

(3) Machines/stations (*Ms*) information: types of machine/station available for use in the system, each of which may have a set of machine configurations (*MCs*) and cost [55].

(4) Feasibility and operation time for each M–MC–OS combination.

Output and decision variables: A solution of the architecture design includes determining the following decision variables: Number of stages (*NS*) to be used in the system, Machine types (*Ms*), their configurations (*MCs*), and numbers of parallel machines for each stage, and the OS assigned to the machines in each stage for each part.

Objective functions: Two objectives are considered here. The details are as follows:

1) Minimize the capital cost of the configuration (the computation of present value as did in the original paper is omitted in this paper for simplicity)

2) Minimize the average unit production time (equivalent to maximize production rate)

Constraints: The constraints for this problem include:

(1) Space limitations. The space allocated to the flow-line is constrained by the length and width available. Such constraints are simply expressed as the maximum number of stage locations (*NSL*), which reflects the length, and the maximum number of parallel machines/stations allowed within a stage (*MMS*), which reflects the width. For other shapes of floor layout, a mapping function can be developed to transform the *NSL* and *MMS* into space related features. Therefore, the basic idea introduced here is still applicable.

(2) Investment limitation: The total initial investment in the configuration cannot exceed the maximum allowable values [55].

(3) Precedence and non-overlap constraints. This provides the full information that

(4) Capacity constraint: the configuration should have sufficient capacity to satisfy the required demand rate for all parts [55].

The description of other implicit constraints such as functionality constraints and decision variable domain constraints are omitted here.

As in [55], the parts to be produced in such an RMS are the ANC-101 and ANC-90, which belong to the same products family. Figure 7.2 illustrates these two parts and their features. The detailed input information, including machining processing information, operation data, operation precedence graph, operation cluster definition for each part, available machine information, and time and production rate information, is provided in Appendix B, which is extracted from Appendix A of [55].

During a configuration period, the production rate requirement for this RMS is 120 parts/hour for ANC-90 (Part A) and 180 parts/hour for ANC-101 (Part B), respectively. Both parts are to be produced simultaneously on the RMS. The maximum number of stages allowed is 10. The maximum number of parallel machines per stage is 5. The maximum allowable budget for initial investment is 30 million US Dollars. All these settings are the same as that in [55].

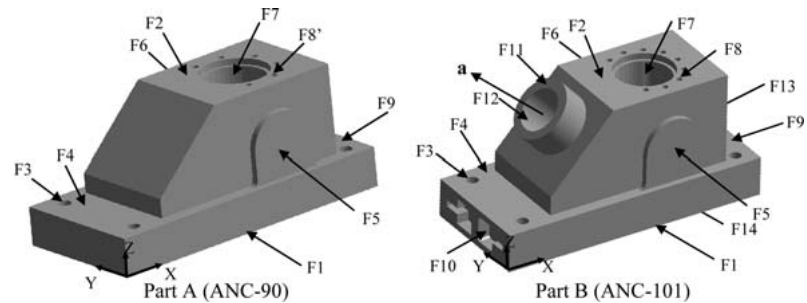


Figure 7.2. Part to be Produced by the RMS ([56])

**7.1.2. Building a Holistic System Model for the RMS.** Following the proposed holistic modeling approach, a generative class model is first developed using OPM/H as shown in Figures 7.3 and 7.4. Figure 7.3 shows a high-level overview of the RMS while Figure 7.4 shows the zoomed-in manufacturing process. The execution semantics of this OPM/H model can be precisely specified using CPN as shown in Figure 7.5, which is developed using CPN Tools (Shorthand notations of the OPM constructs are used). The same CPN model specified the by the ABCD language is shown in Figure 7.6. Since all stages of the RMS share the same structure, only one representation is needed in such a generative class model. Information regarding the configuration of each particular stage is reflected in the instance values (or token values) of both *Machine* object and *Part* object, which are the only variable elements in this system. These variable elements are alternatives from a feature model perspective. Note that the variable elements in this particular model only involve objects, no processes or links. An OPM/H model with only objects as variable elements makes it much easier to generate architecture alternatives. Modeling a system in such a way is encouraged when applying the search-based architecture development proposed in this research. The reason will be explained later.

A *Machine* object is described by 10 attributes as illustrated in Figure 7.4. Their details are explained in Table 7.1. The number of machines in each stage is reflected by the number of machine instances created for each stage. Note that attributes 7 to 10 are attributes describing the dynamic aspects of the *Machine* object, which would not normally present in a model with static information only. Attribute 6, *cost*, has no impact on the dynamic of the RMS system and is, therefore, not mapped to CPN.



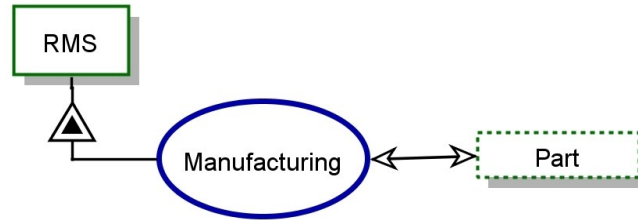


Figure 7.3. OPM/H Model for a RMS - Overview

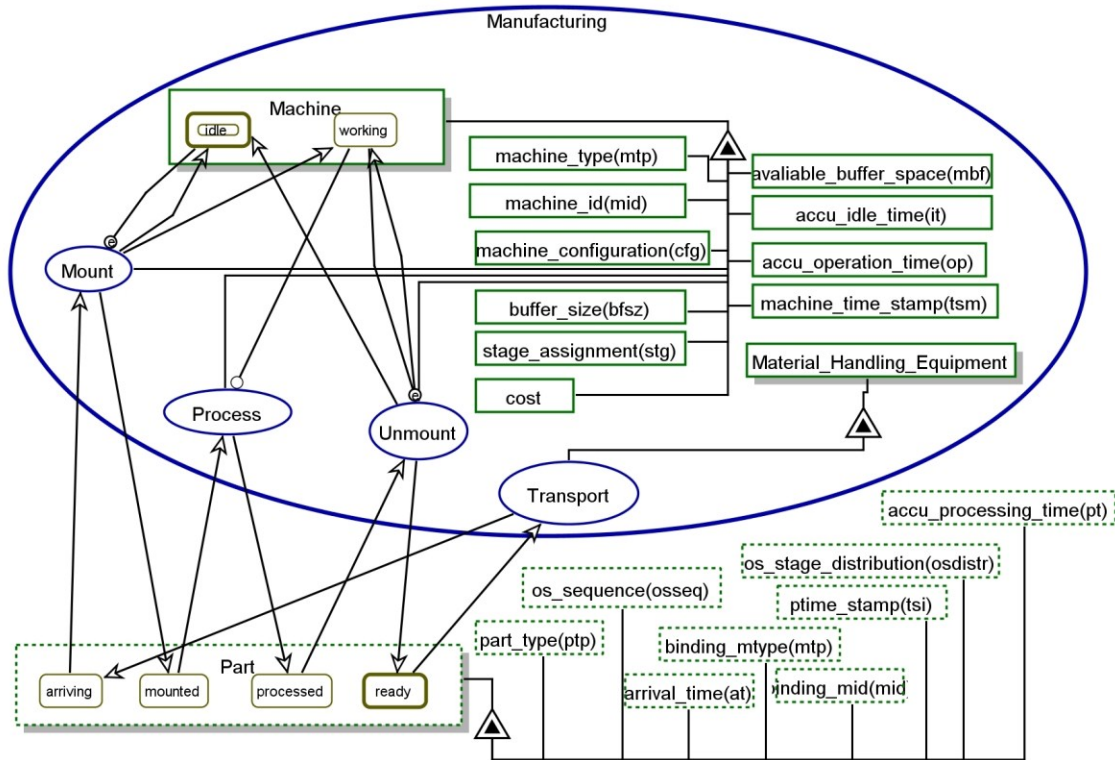


Figure 7.4. OPM/H Model for a RMS – Zoom-in into Manufacturing Process

A `Part` object is described by 8 attributes as shown in Figure 7.4. Their details are explained in Table 7.2. Again, attributes 2 to 8 are dynamic attributes, the values of which keep changing along with the change of the dynamic of the system.

Allowable alternatives for variable elements are specified in the initial value filed of the respective element in the OPM/H model, which are added through the property

sheet of the respective element using the OPCAT tool. As shown in Figure 7.7 (a), a function denoted by “@M\_IDLE\_INIT@” is used to specify the initial value of the machine object. Such function will be implemented by the alternative generation module of the Python program to generate appropriate instance values as the search process proceeds. A set of initial instances for objects in the RMS model is visible on the mapped CPN model in the form of initial markings on the place P\_ready and M\_idle, respectively. The extended information contained in the OPM/H model for specifying the CPN is also set at the property sheet. Such information includes arc annotation (or inscription) (Figure 7.7 (b)) and guard conditions (Figure 7.7 (c)). The added attributes for design space specification is not obvious on Figure 7.3 or 7.4 either. An example that shows the specification of the range of the stage\_assignment(stg) attribute is shown in Figure 7.7 (d). Nevertheless most of these types of information are visible or inferable from the corresponding CPN (Figure 7.5) model though.

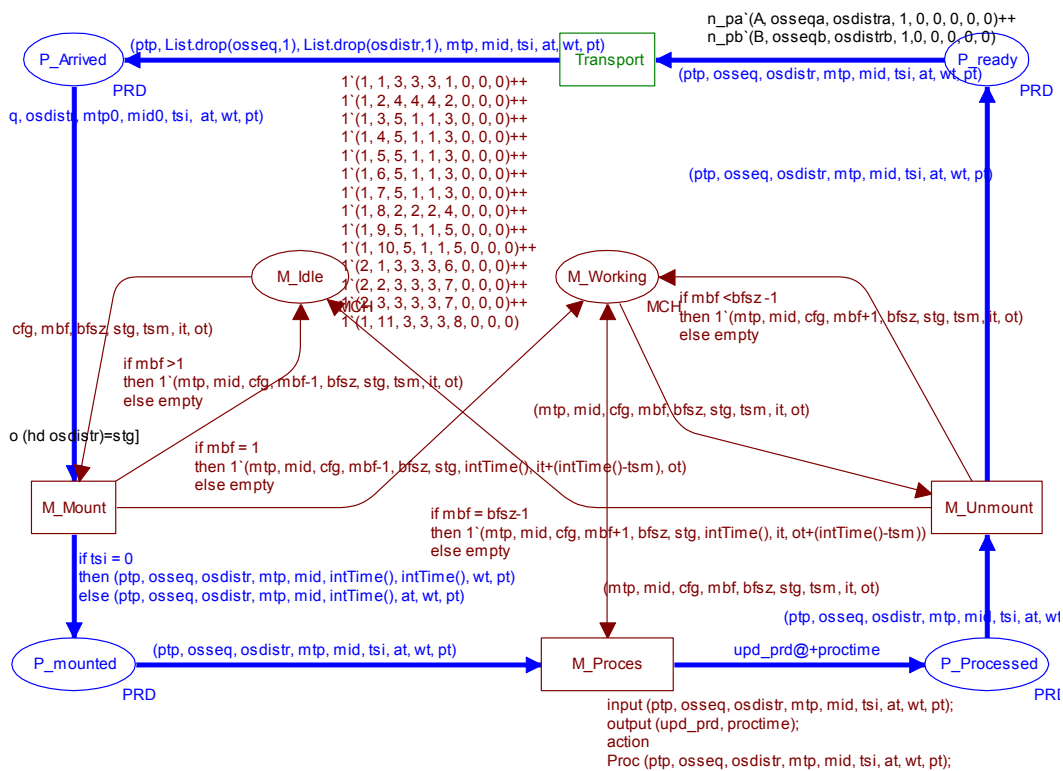


Figure 7.5. CPN Model for the RMS

```

# The last element (type C{str}) of PRD and MCH is time stamp
typedef PRD : int*tuple*tuple*int*int*int*int*str
typedef MCH : int*int*int*int*int*int*int*int*int*str
const p_ready_init = @P_READY_INIT@
const m_idle_init = @M_IDLE_INIT@
const osmachinetime = {1:{1:30 , 2:20 , 3: 30 , 4:20 , 5:60 , 6:120 , 18:90 , 7:18 ,
      8:20 , 9:40 , 10:18 , 11:24 , 12:60 , 13:30 , 14:40 , 15:60 , 16:60 , 17:90},
      2:{3: 30 , 6:120, 18:90}}
buffer P_Arrived : PRD = ()
buffer P_Ready: PRD = p_ready_init
buffer P_Mounted: PRD = ()
buffer P_Processed: PRD = ()
buffer M_Idle: MCH = m_idle_init
buffer M_Working: MCH = ()
net M_Mount () :
    [P_Arrived-((ptp, osseq, osdistr, mtp0, mid0, at, pt, ptstamp)), M_Idle-((mtp,
mid, cfg, mbf, bfsz, stg, tsm, it, ot, mtstamp)), M_Idle+((mtp, mid, cfg, mbf-1,
bfsz, stg, tsm, it, ot, '@'+str(sys_time)) if mbf > 1 else None ), P_Mounted+((ptp,
osseq, osdistr, mtp, mid, sys_time, pt, '@'+str(sys_time)) if int(ptstamp[1:]) ==0
else (ptp, osseq, osdistr, mtp, mid, at, pt, '@'+str(sys_time)) ) , M_Working+((mtp,
mid, cfg, mbf-1, bfsz, stg, sys_time, it + (sys_time - tsm), ot, '@'+str(sys_time))
if mbf==1 else None) if ((osdistr!=()) and ((osdistr[0])==stg))* [False]
net M_Process () :
    [P_Mounted-((ptp, osseq, osdistr, mtp, mid, at, pt, ptstamp)), M_Working?((mtp,
mid, cfg, mbf, bfsz, stg, tsm, it, ot, mtstamp)), P_Processed+((ptp, osseq, osdistr,
mtp, mid, at, pt, '@'+str(sys_time + osmachinetime[mtp][osseq[0]])) )] * [False]
net M_Unmount () :
    [P_Processed-((ptp, osseq, osdistr, mtp, mid, at, pt, ptstamp)), M_Working-((mtp,
mid, cfg, mbf, bfsz, stg, tsm, it, ot, mtstamp)), M_Working+((mtp, mid, cfg, mbf+1,
bfsz, stg, tsm, it, ot, '@'+str(sys_time)) if mbf < (bfsz -1) else None),
P_Ready+((ptp, osseq, osdistr, mtp, mid, at, pt + osmachinetime[mtp][osseq[0]],
'@'+str(sys_time))), M_Idle+((mtp, mid, cfg, mbf+1, bfsz, stg, sys_time, it, ot +
(sys_time - tsm), '@'+str(sys_time)) if mbf == (bfsz -1) else None))* [False]
net Transport () :
    [P_Ready-((ptp, osseq, osdistr, mtp, mid, at, pt, ptstamp)), P_Arrived+((ptp,
osseq[1:], osdistr[1:], mtp, mid, at, pt, '@'+str(sys_time)))] * [False]
# main process with one instance of each net
M_Mount() | M_Process() | M_Unmount() | Transport()

```

Figure 7.6. CPN Model for the RMS Specified in the ABCD Language

Table 7.1. Attributes of the Machine Object in the OPM/H Model

<i>No.</i>	<i>Attribute</i>	<i>Description</i>	<i>Value type</i>	<i>Possible values</i>
1	machine_type (mtp) :	Machine types	int	1 or 2
2	machine_id (mid)	A unique id for each type of machine	int	[1, maximum number of available machines for each machine type)
3	machine_configuration (cfg)	Machine configuration id	int	[1, 5] for machine type 1, [1, 4] for machine type 2
4	buffer_size (bfsz)	Buffer capacity, i.e., the number of part that can be processed simultaneously on a machine. It equals to the number of spindles of a machine in this particular example	int	[1, 4]
5	stage_assignment (stg)	The stage that the machine is installed	int	[1, 10]
6	cost	Cost of the machine	int	Refer to Table A5 for the set of possible values
7	available_buffer_space (mbf)	Buffer space left	int	[0 to buffer size]
8	accumulated_idle_time (it) and	Accumulated idle time	int	[0, )
9	accumulated_operation_time (op)	Accumulated operation time	int	[0, )
10	machine_time_stamp (tsm)	Time stamp of the machine	int	[0, )

Table 7.2. Attributes of the Part Object in the OPM/H Model

No.	Attribute	Description	Value type	Possible values
1	part_type (pt p)	Part type	int	7 or 11
2	os_sequence (osseq)	The sequence of operation cluster setups to be processed for the part	int list	Computed according to the algorithm in [55]
3	os_stage_distribution (osdistr)	The distribution of the sequence of operation cluster setups among available stages	int list	Computed according to the algorithm in [55]
4	Binding_mtype (mtp)	The machine type that the part is mounted to	int	Same as attribute 1 in Table 7.1
5	Binding_mid (mid)	The machine id that the part is mounted to	int	Same as attribute 2 in Table 7.1
6	arrival_time (at)	The time that the part is first mounted	int	[0, )
7	accu_processing_time (pt)	Accumulated processing time of the part	int	[0, )
8	ptime_stamp (tsi)	Time stamp of the part	int	[0, )

The CPN model is worth a closer look. Initially, tokens representing parts are all at the place `P_ready` simulating that they are ready to be moved to the next stage (which may be the first stage) and tokens representing machines are all at the place `M_Idle` simulating the fact that all machines are available before production begins. A token representing a part (or a part token in-short, hereafter) is moved from the place `P_ready` to the place `P_Arrived` when the transition `MHE_Transport` fires simulating that the material handling equipment moves a part from a stage where the part has just been processed to the next stage where the part should be processed according to

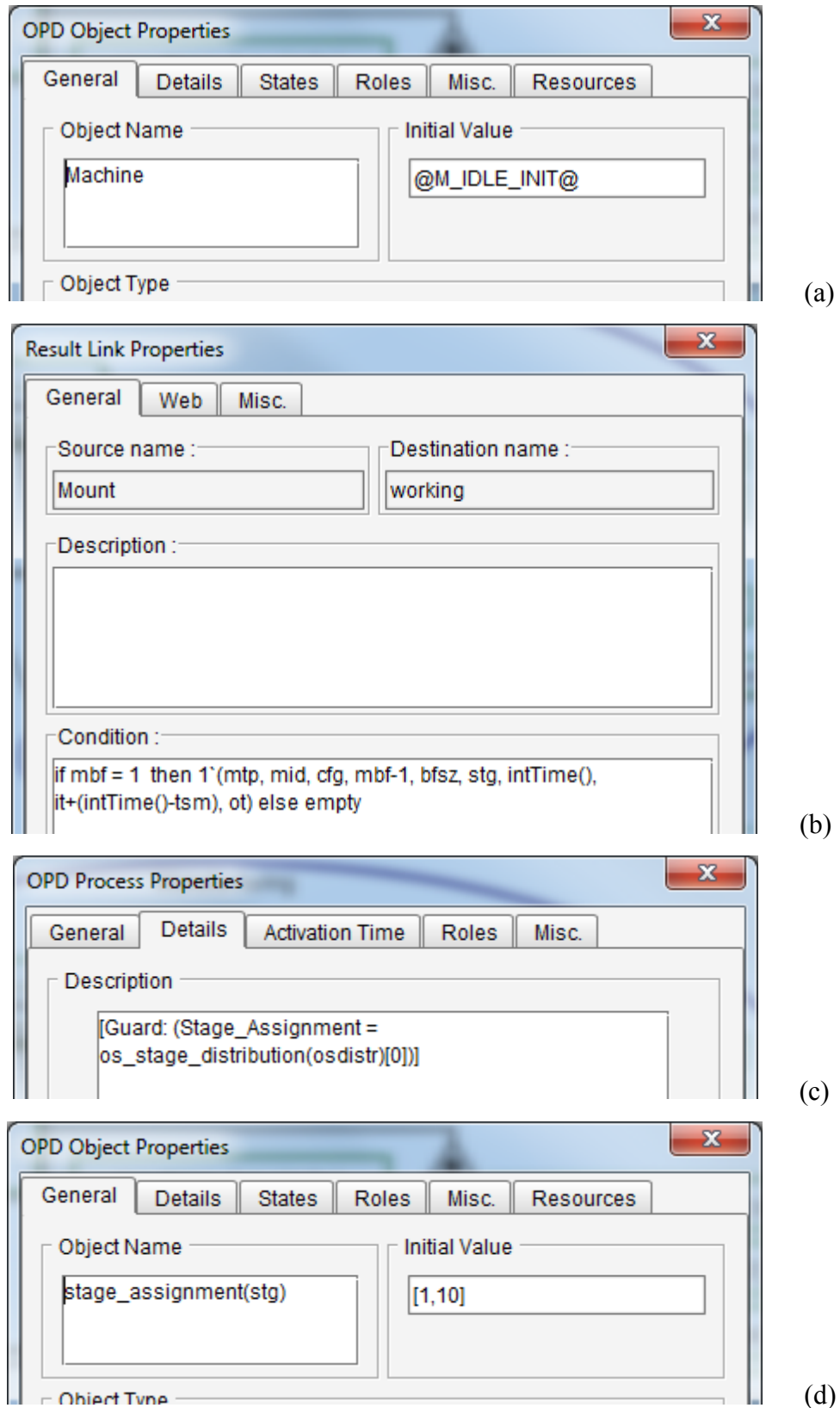


Figure 7.7. Examples of Information Set at the Property Sheet of OPCAT

the OS assignment. As indicated in the input and out arc inscriptions of the transition `MHE_Transport`, when this transition fires, the input token's `osseq` and `osdistr` attribute both have their head values of their respective list removed thus having the information regarding the remaining OSs and their corresponding stage assignments updated. Within each stage, a part goes through the `M_Mount`, `M_Process`, `M_Unmount`, and `MHE_Transport` processes and iterates like this for all stages that the part should be processed. A part is finished when the corresponding part token reaches the place `P_Arrived` and when its `osseq`, and `osdistr` list are both empty, signaling no further processing is needed.

Although there is only one set of transitions in the CPN model, they can still represent actions of all stages of the RMS, thus allowing the modeling of concurrent behavior, as long as bindings can be concurrently enabled. For example, the set of tokens on the place `P_mounted` represent that multiple parts can simultaneously be at the mounted state, each of which may belong to a different stage. The transition `M_Process` can concurrently enabled for all tokens on the place `P_Ready`. The firing of a transition takes no time and there can be at most one transition being fired at each simulation step according to the CPN semantics. Hence whether transitions fires sequentially or concurrently makes no difference in the resultant system states. The result of using multiple `M_Process` transitions firing sequentially is the same as that of using just one `M_Process` transition firing multiple times. Where a part is mounted is reflected by which machine it is bound to as suggested by the value of the corresponding part token's `mtp` and `mid` attributes. These attribute values are determined each time the transition `M_Mount` is fired.

The transition `M_Mount` has a guard inscription. Hence it can only fire when the next stage that a part needs to go (as suggested by the head value of the part token's `osdistr` list attribute) matches the value of the stage attribute (`stg`) of a machine token. Each time the transition `M_Mount` fires, the matching machine token's available buffer (represent by `mbf`) is decreased by 1. A machine token is moved from the place `M_Idle` to the place `M_Working` when its buffer is full simulating the situation that a machine is fully loaded (thus not available for mounting any more) and begins to process

parts. Hence, the `bfsz` attribute of a machine token simulates the product bunch that a machine can simultaneously handle. The transition `M_Unmount` has a reverse effect as that of the transition `M_Mount`. Which machine's buffer is reduced when the transition `M_Unmount` fires depends on the value matching of the `mtp` and `mid` attributes of a part token and a machine token.

A time delay is added after the transition `M_Process` is fired (by function `Proc` in the model shown in Figure 7.5 or by output expression in the model shown in Figure 7.5) representing the time needed to process a part. This is the only transition in this CPN model that changes the time stamp of a token. The time needed for other processes is omitted to simplify the problem. The idle time (`it`) and operation time (`ot`) attributes of a machine token keep tracking the accumulated idle time and operation time, respectively, of the machine represented by the token. These attributes, therefore, can be used to measure the resource utilization. The final value of a part token's time stamp minus the arrival time of the token (represented by the value of the token's `at` attribute) represents the total time that the part is in the system. The total machine time needed for processing a part is fixed for each part type as determined by the sum of the standard machine time corresponding to the set of OSs assigned for the part. The difference between a part's total time in system and its total processing time is the time that a part spent in waiting. The smaller this time is the more efficient the RMS system is.

The generative class model like the one presented in Figures 7.3 and 7.4 can simplify the problem representation and alternative generation by grouping a set of variable elements into one representation. This is achieved by identifying variable elements as object attributes and encoding structural information into attribute values as much as possible. The rationale of such approach is that, given the proposed architecture alternative method, it is much easier to create object instances, even with complicated attributes, than to create a structure (i.e., a set of interconnected objects and, possibly, processes). For example, the production stage could have been modeled as an object with machines and OS as its attributes. Accordingly, the alternative way of representing the RMS is presented in Figure 7.8 through Figure 7.10. The problem with such RMS models is that there must be a representation for each individual stage in the OPM model. Consequently, each RMS configuration with a different OS sequence distribution will



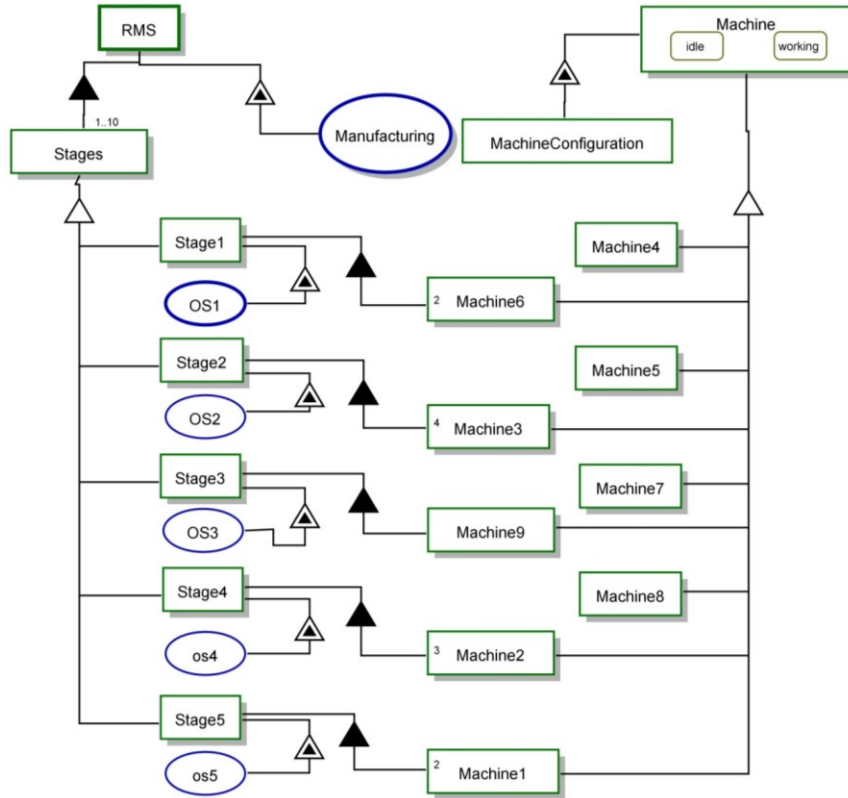


Figure 7.8. An Alternative Way to Model the RMS - Un-fold RMS

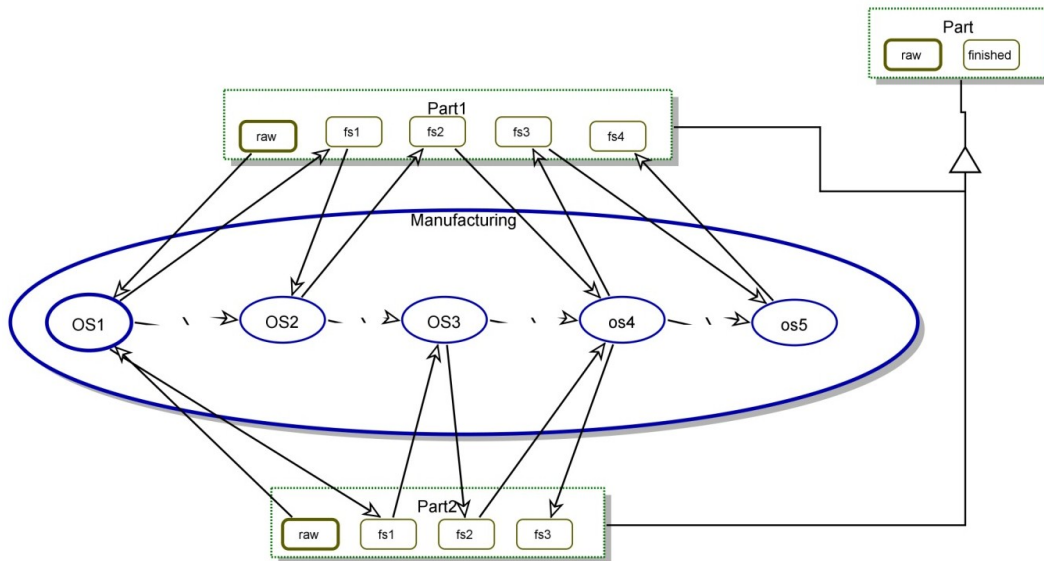


Figure 7.9. An Alternative Way to Model the RMS - Zoom-in into Manufacturing Process

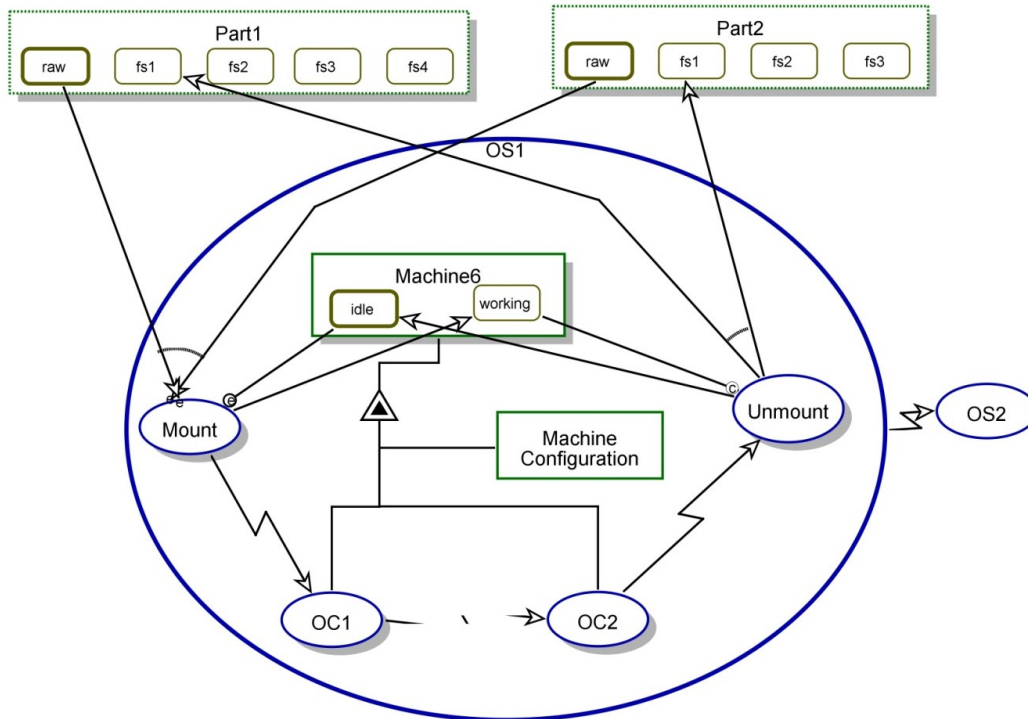


Figure 7.10. An Alternative Way to Model the RMS - Zoom-in into OS1 Process

need a unique structure to be expressed by the OPM model. Generating architecture alternatives for such kind of system model requires executing a lot of variant generation operations, making solving the problem rather difficult. Since a production stage is a virtual concept, by merging stage information to the attributes of machines (*mstg*) and parts (*osseq* and *osdistr*). Both the problem representation and alternative generation can be greatly simplified.

From the OPM/H model presented in Figures 7.3 and 7.4, the dimensions of the design space of the RMS configuration can be expressed explicitly. The main dimensions of the design space are (*Machine* × *Part*). The sub-dimensions of *Machine* are (*machine\_type*(*mtp*) × *machine\_configuration*(*cfg*) × *stage\_assignment*(*stg*) × *number\_of\_machine*). The sub dimensions of *Part* is (*part\_type*(*ptp*) × *os\_sequence*(*osseq*) × *os\_stage\_distribution*(*osdistr*)). Transit (or dynamic) attributes of an object only make sense when the system is running and therefore should not be counted

in the dimensions of the system design space. The transit attributes for the machine object in the OPM/H model presented in Figure 7.4 include `available_buffer_space(mbf)`, `accu_idle_time(it)`, `accu_operation_time(op)`, and `machine_time_stamp(tsm)`. For the part object, the transit attributes include `arrival_time(at)`, `binding_mid(mid)`, `binding_mtype(mtp)`, `arrival_time(at)`, `accu_processing_time(pt)`, and `ptime_stamp(tsi)`.

The constraints between attributes of objects within the OPM/H model presented in Figures 7.3 and 7.4 have not been captured. For example the information in the operation precedence graph shown in Figure A1 cannot be captured by such model. In order to capture such information, the extended (or advanced) feature model [209–213] concepts must be implemented. Such advanced feature models not only allow features to have attributes, which can have domain and values, and but can also capture the complex relationships and constraints among features and feature attributes.

**7.1.3. Building Analysis Models.** Two quantitative objects are considered here. One of them is obtained from mathematical equation; the other is derived from simulation. The details are as follows:

1) The capital cost of the configuration is computed from the following equation:

$$CC = \sum_{s=1}^{NS} (n_s \times CMC_{M_s}(c_s)) \quad (7.1)$$

where  $CC$  is the capital cost of the configuration,  $n_s$  is the number of machines in stage  $s$ ,  $CMC_{M_s}(c_s)$  is the cost of machine  $M$  at stage  $s$  when configured at configuration  $c$ , and  $NS$  is the number of stages in the system.

2) The average unit production time and production rate is derived from the information obtained from the simulation of the CPN model and computed according to the following equations:

$$PT_u = T_{sys} / NP_f \quad (7.2)$$

$$PR = 3600 / PT_u \quad (7.3)$$

Where  $PT_u$  is unit production time (seconds),  $PR$  is production rate (parts/hour),  $T_{sys}$  is the model time of the CPN model when the simulation is end, and  $NP_f$  is the number of parts finished, which is the number of tokens at the place `P_Arrived` with their `osseq` and `osdistr` list both empty when the simulation is end.

**7.1.4. Building Optimization Models.** As concluded in [214] and referenced in [55], a special case of this optimization problem with fixed machine configurations, fixed order of operations and no consideration of capacity requirements was proven to be NP-hard. Therefore the GA, as a meta-heuristic global optimization algorithm, is good for solving this problem. Since this problem is a multi-objective optimization problem, the search algorithm adopted here is the non-dominated sorting genetic algorithm II (NSGA-II) [215]. NSGA-II is a modified version of the NSGA, a popular non-domination based genetic algorithm for multi-objective optimization. NSGA-II reduces the computational complexity of NSGA, incorporates elitism and requires no sharing parameter to be chosen *a priori* [215].

As suggested in Section 5.1.3.2, the chromosome encoding of the RMS model only needs to capture the variable elements of the RMS model. According to the design space analysis conducted at Section 7.1.2, the information regarding these variable elements can be summarized as: (1) the OS sequence for each part type, (2) the distribution of OS sequence over the available production stages, (3) the selected machine type and its configuration and multiplicity for each stage. A chromosome can be constructed accordingly as shown in Figure 7.11. Each element of the chromosome can be a real number representing a selected value of the respective design variables. This chromosome encoding scheme coincides with the one proposed in [55]. In order to facilitate the comparison of the proposed approaches in this research with the work done in [55], the exactly same real encoded (ranging from 0 and 1) chromosome is used in this dissertation. A more intuitive string representation of a configuration solution can be developed as suggested in [55] (Figure 7.12). Such string representation starts with one element representing the number of stage, followed by  $NS$  number of segments (i.e., groups of elements), each of which represents a configuration of a stage. Within each segment, there is a list of elements representing the machine, the machine configuration, the number of machines, and the OS assignment for each part, respectively. Therefore the total length of each segment is  $(3 + NP)$ , where  $NP$  is the number of part.

**7.1.5. Development of Problem-Specific Modules in Python.** Three problem-specific Python modules have been developed for solving the RMS problem. The `RMS_DataPcs` module processes the raw input data in order to generate all kinds of

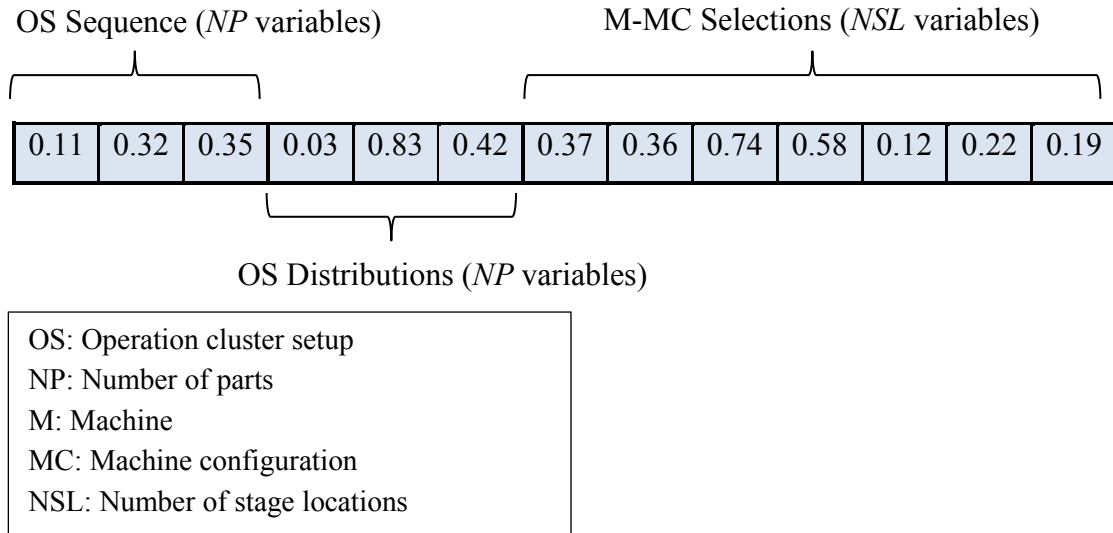


Figure 7.11. Chromosome Encoding of the Design Variables for Solving the RMS Problem ([55])

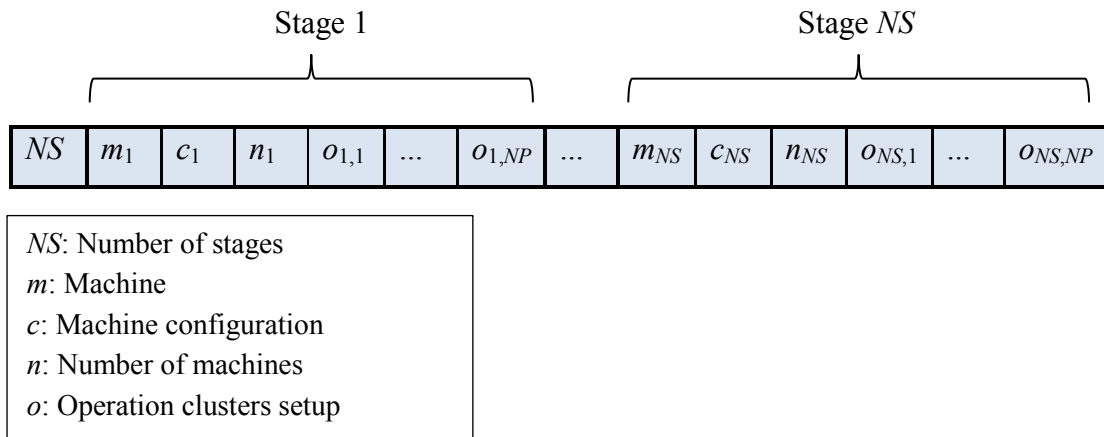


Figure 7.12. String Representation of a Solution ([55])

data required by both the system model and the alternative generation. It contains an `Rms` class along with some other classes and functions needed by the `Rms` class. The `Rms` class contains the data model for the RMS problem and the functions needed to compute and generate various kinds of data. The `RMS_data_provider` module is responsible

for loading the raw input data for constructing the RMS data mode. It runs on top of the `RMS_DataPcs` module and uses the services provided by it to create an `rms` object of the `Rms` class to contain all data regarding the RMS required by other programs. The `RMS_GA_problem` module encompasses several functions to support the running of the main GA program. It contains an `Rms_dgn` class, which provides the generator (for generating candidates) and evaluator (for computing objective functions) functions needed by the main GA program. The `Rms_dgn` class also contains a function to decode a chromosome into a string representation of the configuration solution in the format illustrated in Figure 7.12, which is both human and machine readable. It also contains a function to construct a CPN model using the decoded chromosome (i.e., string representation). The simulation of a CPN model is invoked by the evaluator function. A CPN simulation is end when there are no more enabled binding. The simulation result is then returned in the form of final markings. The evaluator then use the information derived from the final marking to compute one of the objectives, the unit production time. The other objective, system cost, is computed using the machine cost information stored in the `rms` object of the `Rms` class according to the machine information contained in the decoded chromosome. More objectives can be used by defining more objective functions within the `Rms_dgn` class and adding them to the evaluator function. As suggested in Section 7.1.2, the simulation of a CPN model can provide several performance metrics, which can also be used to construct objective functions.

The CPN model for the RMS was initialized with 24 tokens for part A and 36 tokens for part B. The ratio between these two numbers is in proportion to the production rate requirements of these two types of parts. The parameters used in the GA are summarized in Table 7.3.

**7.1.6. Results and Discussion.** The results (Pareto-front or population) can be plotted after the GA finishes running. Figure 7.13 shows the Pareto-front obtained from an optimization run, which contains 5 non-dominant solutions. The user can select one of them as the final solution based on more detailed analyses. This research is only intended to provide such reduced solution space. The string representations of these five solutions are provided in Figure 7.14. One of them is illustrated graphically on Figure 7.15. Figure 7.16 demonstrates the convergence curve of the GA for the two objectives.

Table 7.3. Parameters Used in the GA

<i>Parameter</i>	<i>Value</i>
GA algorithm	NSGA-II
Population size	80
Number of generations	80
Crossover operator	<i>blend_crossover</i> ( <i>blx_alpha</i> : 0.1, <i>blx_points</i> : 1, 2 or 5 to 14), <i>simulated_binary_crossover</i> ( <i>sbx_distribution_index</i> : 5), <i>heuristic_crossover</i> , <i>arithmetic_crossover</i>
<i>crossover_rate</i>	0.9
Mutation operator	<i>gaussian_mutation</i> ( <i>gaussian_stdev</i> : 0.3) nonuniform_mutation
<i>mutation_rate</i>	0.1
<i>selector</i>	<i>tournament_selection</i> ( <i>tournament_size</i> : 5)
<i>replacer</i>	<i>nsga_replacement</i>
<i>terminator</i>	generation_termination ( <i>max_generations</i> : Number of generations), evaluation_termination ( <i>max_evaluations</i> : 5000), time_termination ( <i>max_time</i> : 72,0,0)
<i>archiver</i>	best_archiver
<i>observer</i>	file_observer

\* The key words used in the *Inspired* package are in italic

For this RMS problem, with 60 (24+60) part tokens initialized, each simulation run of a CPN model took approximately 1 minutes to finish (by using an Intel CORE i5 computer with 4 Gb RAM). It included both the simulation time and the time it took to parse the CPN model (Every instance of the CPN model, specified by the ABCD language, has to be parsed by the ABCD parser and then be built using the modified *nets* module in the current implementation). With a CPN initialized with the same number of part tokens, the CPN Tools took less than 3 seconds to finish the simulation. Therefore, the current code is not efficient enough and there should be a large space for improvement.

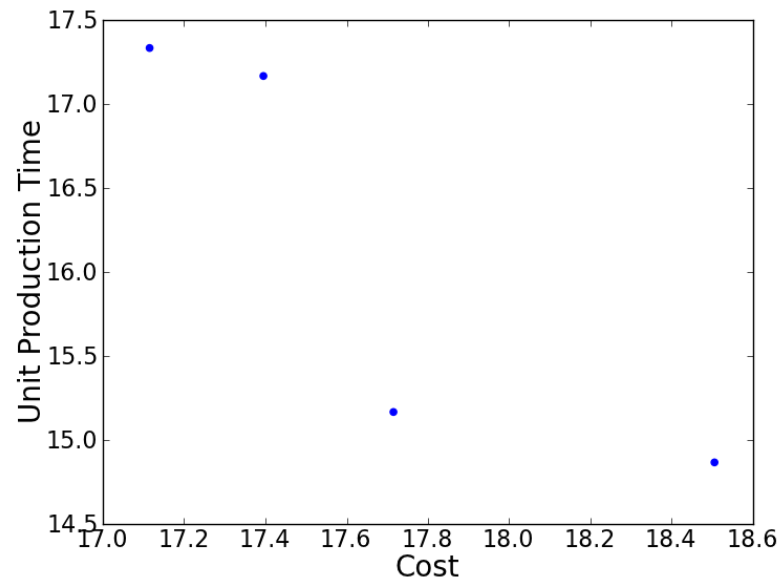


Figure 7.13. The Pareto-front of the Solutions Found Using GA for the RMS Problem

Alternative 1 (17.12, 17.33)								Alternative 2 (17.46, 17.18)							
S	1	2	3	4	5	6	7	S	1	2	3	4	5	6	7
M	1	1	1	1	1	1	2	M	1	1	1	1	1	2	1
MC	2	2	5	5	3	5	3	MC	1	3	5	2	5	4	2
NMS	2	1	3	5	1	2	3	NMS	3	2	3	1	4	3	1
OS <sub>A</sub>	1	15	0	5	0	0	18	OS <sub>A</sub>	1	15	0	0	5	18	0
OS <sub>B</sub>	1	0	16	5	12	9	6	OS <sub>B</sub>	1	15	5	13	9	6	11
Alternative 3 (17.72, 15.16)								Alternative 4 (18.51, 14.86)							
S	1	2	3	4	5	6	7	S	1	2	3	4	5	6	7
M	1	1	1	1	1	1	2	M	1	1	1	1	1	1	2
MC	2	1	5	5	5	3	3	MC	2	1	5	5	5	4	4
NMS	1	1	5	2	5	1	3	NMS	1	1	5	2	5	1	3
OS <sub>A</sub>	0	1	15	0	5	0	18	OS <sub>A</sub>	0	1	15	0	5	0	18
OS <sub>B</sub>	1	0	16	9	5	12	6	OS <sub>B</sub>	1	0	16	9	5	12	6

Figure 7.14. Near-Optimal Solutions in the Pareto-front



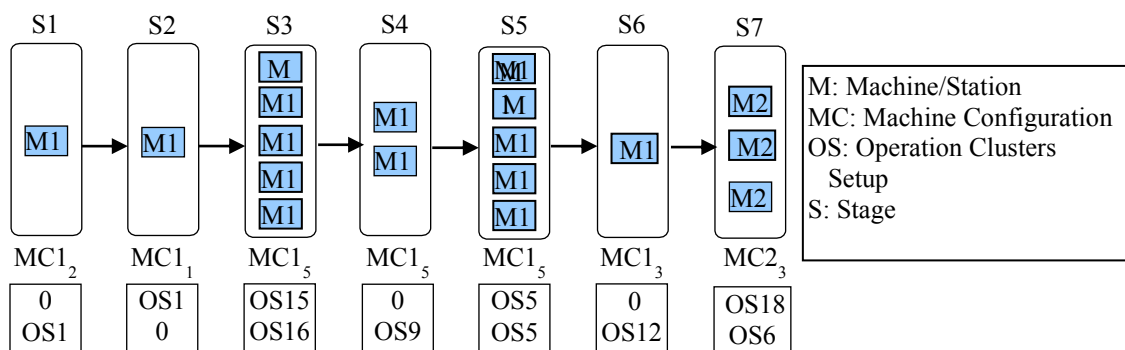


Figure 7.15. Illustration of One of the Near-Optimal Solution

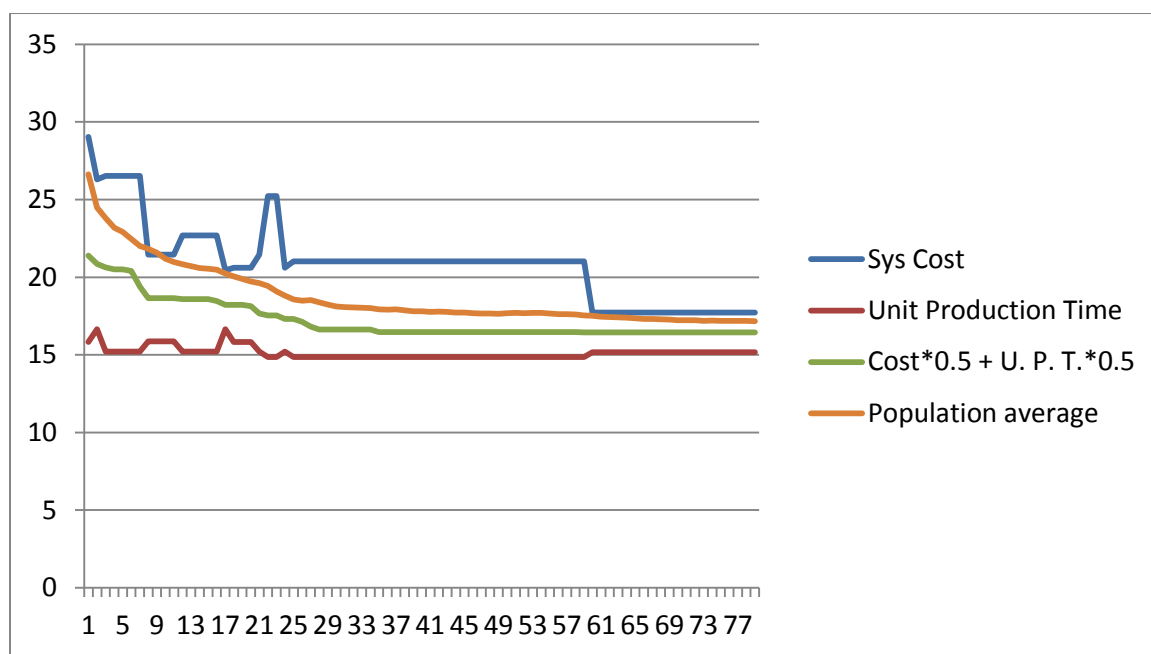


Figure 7.16. Convergence Curve of the NSGA-II in Solving the RMS Configuration

For example, eliminating the need of the ABCD parser can at least save the time to parse and build the CPN model. Currently it takes approximately 140 hours to run the GA for 80 generations with a population size of 80. Increasing the population size and number of generations can yield better solutions as implied by the work in [55] where the population size is 100 and number of generations is 150.

There are a number of factors affecting the computation of the performance metrics. A near-optimal architecture is needed for such analyses in order for the result to make sense. Hence the optimum solution present in [55] is used to facilitate the comparison. Its configuration has already been shown in Figure 7.1.

As indicated in equation (7.2) and (7.3), the computation of the unit production time and the production rate has not eliminated the impact of the ramp-up period. Since the system is not fully loaded during the ramp-up period, the results computed from equation (7.2) or (7.3) do not truly reflect the unit production time or the production rate. Leaving out the first few finished parts from the computation can reduce or remove the impact of the ramp-up period but will require more part tokens to be used in the CPN simulation and, therefore, has not been implemented in the result shown above. The impact of the ramp-up period on the computation of the unit production time or the production rate can also be reduced by using more part tokens in the CPN simulation. The more part tokens used, the less impact the ramp-up will have, and the closer the equation (7.2) and (7.3) will be to the true values. The impact of increasing part tokens on the computation of the unit production time and the production rate is demonstrated in Table 7.4. The simulations used the architecture presented in Figure 7.1 (i.e., the optimal one in [55]) initialized with 60, 90, 300 and 600 part tokens in each simulation run, respectively.

Using more part tokens in the CPN simulation itself can improve the accuracy of the computed results because the variance will be reduced as the sample size increase. Similarly, running the simulation multiple times can also result in better accuracy but the performance margin is small because the randomness does not play a big role in this problem setting. Table 7.5 shows the results obtained from 10 CPN simulation runs using the same architecture with 60 part tokens in each run. As can be seen that the standard deviation is really small and therefore the accuracy from one simulation run should be acceptable. There is another factor that can make the production rate computed from equation (7.3) lower than it should be. Some machines in the system can handle multiple parts simultaneously. In the current implementation, such machines require all needed parts to be mounted before it can begin processing. If the number of part tokens used in the simulation is too small, it happens that some parts are not able to form complete batch

Table 7.4. Impact of the Number of Part Tokens Used in the CPN Model on the Computation of the Unit Production Time and the Production Rate

# finished part	60	84	300	600
Finish time	1290	1630	4730	9050
Unit production time	21.50	19.40	15.77	15.08
Production rate	167.44	185.52	228.33	238.67

Table 7.5. Statistics from 10 CPN Simulations

Exp. #	1	2	3	4	5	6	7	8	9	10	Mean	STD	STD /Mean
Finish time	1270	1230	1290	1230	1310	1210	1310	1290	1310	1270	1272	37.059	0.029
# finished part	60	60	60	60	60	60	60	60	60	60	60	0.000	0.000
Unit production time	21.17	20.5	21.5	20.5	21.83	20.17	21.83	21.5	21.83	21.17	21.2	0.618	0.029
Production rate	170.1	175.6	167.4	175.6	164.9	178.5	164.9	167.4	164.9	170.1	169.9	5.022	0.030

and eventually cannot have all the required OSs processed. Such tokens do consume some processing time and machine resources but are left out from equation (7.2 and 7.3). Hence the result obtained from equation (7.3) is lower (or higher in equation (7.2)) than it should be.

By comparing the solutions obtained using the approach proposed in this research (Figures 7.13 and 7.14) with the one developed in [55] (as shown in Table 7.5), it can be seen that the solutions obtained in this research has lower unit production time (i.e., higher production rate) but higher cost (\$17.12 million obtained here vs. \$ 13.92 million obtained in [55]) than the best solution obtained in [55]. However, the simulation results presented in Table 7.4 show that the best solution obtained in [55] could not actually satisfy the production rate requirements, which is 300 (180 + 120) parts per hour. This conclusion holds even when running the simulation with 600 part tokens, where impact of the ramp-up period should be very small. A closer examination of the final marking obtained from the CPN simulation of an RMS configuration can explain why the calculated production rate is lower than expected. Table 7.6 and 7.7 present summaries of the final markings, along with some calculations, obtained from a CPN simulation

with 60 part tokens using the best configuration obtained in [55] (Figure 7.1). The resource utilization rate of each machine shown in Table 7.6 suggests that most of the machines were not fully utilized during the simulation period. Therefore, the production rate of the entire production line is usually lower than the capacity of any of its stage. Furthermore, Table 7.7 indicates that the accumulated waiting time of a part is not zero. Such observation happens to apply to all parts being processed by the RMS as shown in Table 7.7. The machine time of each stage is usually not the same. The number of machine in each stage is a discrete number. Therefore, the production rate of each stage may not match each other exactly. Accordingly a part has to spend some time in waiting between stages. For such reasons, a design with each stage satisfying the minimum production rate requirements usually won't be able to satisfy the production rate requirements as far as the entire production line is concerned. From this example, it can be concluded that the scale of the waiting time spent by parts and the effective production rate of the production line cannot be accurately assessed without using simulations like the one provided by CPN.

Table 7.6. Final Marking on the Place  $M\_Idle$  Obtained from One Simulation Run of the CPN Model for the RMS

	<i>mtp</i>	<i>mid</i>	<i>cfg</i>	<i>mbf</i>	<i>bfsz</i>	<i>stg</i>	<i>tsm</i>	<i>it</i>	<i>ot</i>	<i>Time stamp</i>	<i>Resource Utilization</i>
1	1	1	3	3	3	1	600	0	600	600	100%
2	1	2	4	4	4	2	940	60	880	940	94%
3	1	3	5	1	1	3	980	260	720	980	73%
4	1	4	5	1	1	3	920	200	720	920	78%
5	1	5	5	1	1	3	1000	280	720	1000	72%
6	1	6	5	1	1	3	1000	220	780	1000	78%
7	1	7	5	1	1	3	980	320	660	980	67%
8	1	8	2	2	2	4	1010	470	540	1010	53%
9	1	9	5	1	1	5	1070	350	720	1070	67%
10	1	10	5	1	1	5	1070	350	720	1070	67%
11	1	11	3	3	3	8	1270	550	720	1270	57%
12	2	1	3	3	3	6	1090	370	720	1090	66%
13	2	2	3	3	3	7	1190	470	720	1190	61%
14	2	3	3	3	3	7	1150	430	720	1150	63%

Table 7.7. Final Marking on the Place  $P_{Arrived}$  Obtained from One Simulation Run of the CPN Model for the RMS

#	ptp	at	pt	finish time	wt	Time in System	#	ptp	at	pt	finish time	wt	Time in System
1	11	0	380	450	70	450	31	7	300	240	700	160	400
2	11	0	380	450	70	450	32	7	300	240	790	250	490
3	11	0	380	450	70	450	33	11	300	380	730	50	430
4	11	30	380	510	100	480	34	7	330	240	700	130	370
5	11	30	380	510	100	480	35	7	330	240	790	220	460
6	11	30	380	510	100	480	36	11	330	380	890	180	560
7	7	60	240	430	130	370	37	7	360	240	610	10	250
8	7	60	240	430	130	370	38	11	360	380	810	70	450
9	11	60	380	570	130	510	39	11	360	380	950	210	590
10	11	90	380	570	100	480	40	7	390	240	1090	460	700
11	11	90	380	570	100	480	41	11	390	380	810	40	420
12	11	90	380	890	420	800	42	11	390	380	1270	500	880
13	7	120	240	520	160	400	43	7	420	240	700	40	280
14	7	120	240	520	160	400	44	11	420	380	950	150	530
15	11	120	380	630	130	510	45	11	420	380	1270	470	850
16	7	150	240	430	40	280	46	7	450	240	910	220	460
17	11	150	380	630	100	480	47	7	450	240	1090	400	640
18	11	150	380	730	200	580	48	11	450	380	1270	440	820
19	7	180	240	1000	580	820	49	11	480	380	950	90	470
20	11	180	380	630	70	450	50	11	480	380	1050	190	570
21	11	180	380	810	250	630	51	11	480	380	1210	350	730
22	7	210	240	610	160	400	52	7	510	240	790	40	280
23	7	210	240	1000	550	790	53	11	510	380	1110	220	600
24	11	210	380	730	140	520	54	11	510	380	1210	320	700
25	7	240	240	520	40	280	55	7	540	240	1000	220	460
26	7	240	240	610	130	370	56	7	540	240	1090	310	550
27	7	240	240	910	430	670	57	11	540	380	1110	190	570
28	7	270	240	910	400	640	58	11	570	380	1050	100	480
29	11	270	380	890	240	620	59	11	570	380	1110	160	540
30	11	270	380	1050	400	780	60	11	570	380	1210	260	640
Mean:											202.5	526.5	
Standard deviation											144.6	152.6	

## 7.2. THE APOLLO PROGRAM (RETROSPECTIVE)

The Apollo program was a benchmark problem in the discipline of systems engineering and has been very well studied. To further demonstrate the application of the proposed approach, a retrospective study of the manned lunar landing system architecture design for the Apollo program is made here. The actual architecture design for such a system is very complicated involving many design factors. Since the purpose is demonstration, only very limited design aspects are considered in this study. Even though, the information needed to support the architecture reasoning task is not fully available. Moreover, a solution based on such a scaled down problem, with only limited aspects considered, may not agree with the one obtained in the real-world scenario. Therefore, rather than trying to find a design solution, this study focuses on demonstrating how to use the proposed modeling approach to develop a holistic architectural model that supports design space specifications and alternative generations (with structural difference between alternatives).

**7.2.1. Problem Definition and Analysis.** The primary objective of the Apollo program is to accomplish the initial manned lunar landing and return of a United States citizen before the end of the 60s decade. Such objective include three sub-goals: manned lunar landing, crew return, and a one decade time limit [216–218].

An architecture development can start from analyzing the initial, final, and critical mission states that the system need to achieve and then find the means to achieve the transitions between these states. Achieving lunar landing implies conquer the distance obstacle. Therefore, the positions of the lunar landing system can be modeled as critical states to be considered in the design process. For the Apollo mission, the initial state of the lunar landing system is the Earth launching site, the final state is the Earth landing site, and the critical mission state in between is the moon surface. The trajectory of the lunar landing system describes the trace of the intermediate states between the initial state and the critical mission state and between the critical mission state and the final state.

In a continuous space, such trajectories are infinite. For the initial architecture design phase, a precise trajectory is not necessary. Hence the description of trajectory can be simplified by identifying it as discrete design space. As stated in [1], Frazolli [219] developed an approach to quantizes the description of continuous dynamic systems into a

set of motion primitives. This approach describes the motion of an object using two motion primitives. One is repeatable motions, which are motions at constant speeds or constant accelerations. The other is finite time motions, which are other (non-constant) motion speeds and accelerations. Using such method, the entire trajectory for the Apollo mission can be described using Figure 7.17 as suggested in [1]:

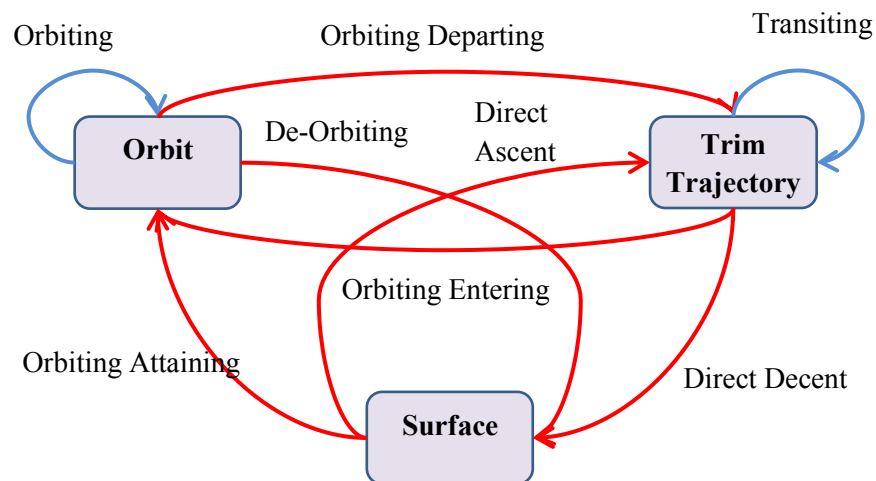


Figure 7.17. Discrete-Space Representation of the Trajectory of the Manned Lunar Landing System ([1])

Different motion trajectories may require lunar landing system to have different operation sequences during the journey, which in turn require the support of different set of equipment and different system configurations. A selection of the trajectory and operational sequence is called a mode in the Apollo program [216], [217]. The choice of mode affects not only design requirements for many system elements but also the schedule and program risks. Therefore the mode selection is regarded as the most important design factors according to many studies and history records [217], [218], [220].

A mode includes both a launch vehicle capability and a required set of maneuvers [218]. According to the initial Apollo program studies [218], the major modes considered for the initial manned lunar missions are:

1. Lunar Orbit Rendezvous (LOR) using the C-5 launch vehicle and the present *Apollo* Command Module.
  - a. 1-day stay-time on the moon with 24-hour contingency
  - b. 7-day stay-time on the moon
2. Earth Orbit Rendezvous (EOR) using the C-5 launch vehicle and the present *Apollo* Command Module.
3. Direct Flight (DF / Liquid *Nova*) using the Liquid *Nova* or C-8 launch vehicle and the present *Apollo* Command Module.
  - a. 8 F-1, 9 J-2, 1 J-2 [C-8 (9)]
  - b. 8 F-1, 5 J-2, 1 J-2 [C-8 (5)]
  - c. 8 F-1, 2 M-1, 1 J-2 [*Nova*]
4. Direct Flight (DF/C-5) using the C-5 launch vehicle and a smaller, modified Command Module.
5. Direct Flight (DF / Solid *Nova*) using the Solid *Nova* launch vehicle and the present *Apollo* Command Module.

A manned lunar landing system is comprised of a launch vehicle and a spacecraft. The launch vehicle is responsible for escape the Earth gravity. The spacecraft is responsible for the moon orbit entering and the remaining flying task plus the landing mission. The major design factor for selecting launch vehicle is its payload. Depending on the mission mode, the spacecraft can have different configuration and accordingly require different propulsion system. The lunar landing module should further consider parameters such as weights, size, mission duration, crew capacity.

**7.2.2. Architecture Modeling.** Based on the above analyses, a primary system architecture can be developed for the manned lunar landing system by identifying the system elements required by various mission operations. Such an architecture can be modeled using the OPM/H as shown in Figures 7.16. The overall manned lunar landing system is modeled as an OPM object, with a set of states corresponding to the progress of the mission. Among those states, are the initial state, Earth launch site, the final state, Earth landing site, the critical miss states, Moon landing site and Moon launching site, and a set of states describing the trajectory of the system in between. As presented in Section 7.2.1, the repeatable motions of the trajectory are



modeled as the states of the Manned lunar landing system object whereas the finite time motions of the trajectory are modeled as a set of OPM processes that change the states of the Manned lunar landing system object. Note that some of the finite time motions, i.e., the midcourse maneuvers and the orbiting maneuvers have been omitted in the model shown in Figure 7.18 for simplicity of representation. These two types of maneuvers maintain the state of the manned lunar landing system instead of making it transit to another state. Hence it is not that important for the problem considered here. Moreover, the system should have the capability to abort the mission at any state if necessary. Such aborting maneuvers are omitted too for simplicity. For each of the maneuver (or operation) modeled as an OPM process, one or more system elements needed to support it are identified and connected to it using OPM instrument links. For example, the process `MissionPerforming` needs either Command Module (CM) or Lunar Excursion Vehicle (LEV). Therefore, two OPM objects representing them (CM and LEV) are connected to the process `MissionPerforming` using OPM instrument links. These two links are also joined by an OPM XOR representing that exactly one of them is needed.

In the original design, the EOR and DF modes also use a Lunar Touchdown Module (LTDM) for executing the midcourse maneuvers and providing the Lunar Braking Module (LBM) thrust vector control. In this simplified architecture model, the functionality of the LTDM is combined with that of the LBM and only one representation, the LBM, is present on the model. The LOR mode can also use a two stage Service Module (SM) and a LEV, which is composed of a Lunar Excursion Module (LEM) and two fully-staged propulsion systems. The simplified architecture model present here makes no distinction between the two stages of the SM and uses a LEV to represent both the LEM and its propulsion systems.

**7.2.3. Design Space Analysis.** The architectural model shown in Figure 7.18 is a generative class model that can capture the design space. All design alternatives can be generated based on such model. One dimension of the design space is the mode. All possible maneuvers (modeled as OPM process) and system states are present on the OPM model (except for those left out intentionally as explained in Section 7.2.2). A possible mode is a sequence of interconnected maneuvers and system states interleaving each

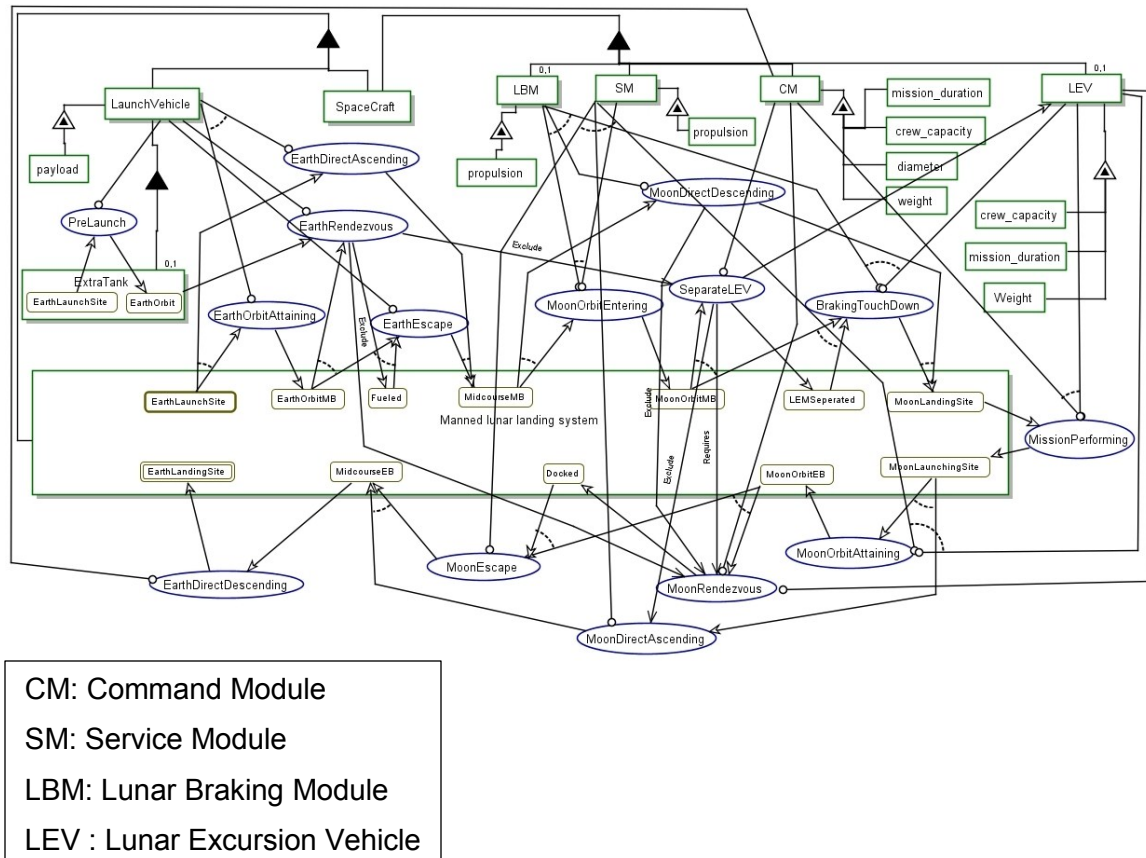


Figure 7.18. OPM/H Class Model Representing the Architecture of the Manned Lunar Landing System Represented

other along the sequence. When multiple processes originating from or joining the same state using the OPM result/consumption links, the relationships between these links should be specified (the relationship is “AND” by default). For example, there are three maneuver-state sequences between the state EarthLaunchSite and the state MidcourseMB. They are (1) (EarthOrbitAttaining - EarthOrbitMB - EarthRendezvous - Fueled - EarthEscape), (2) (EarthDirectAscending) and (3) (EarthOrbitAttaining - EarthOrbitMB - EarthEscape). Several OPM XOR relations were used to connect related result/consumption links. Such usage of the OPM XOR relations suggests that exactly one of these sequences should present in a particular architecture alternative. The above

sequence (2) and (3) are mostly equivalent to each other so the sequence 3 is excluded in the discussions below.

A mode outlines a profile of the Apollo mission and thus will determine a particular spacecraft configuration. The total weight of the spacecraft, along with the trajectory, will in turn determine the launch vehicle to be used. The major concerns in selecting the launch vehicle are the payload and the trajectory requirement. Table 7.8 summarizes three major modes captured by the architecture model and the corresponding spacecraft configuration. It can be inferred from Table 7.8 that the both CM and SM are mandatory while both LBM and LEV are optional. Such mandatory/optional features have been specified using the participation cardinality on the OPM/H architectural model shown in Figure 7.18. The information in Table 7.8 further proves that the mission mode is the most critical factor in the architecture design of the lunar landing system as it has a significant impact on the requirements of other system elements.

For each module within the spacecraft, there are also a set of parameters to be determined such as the propulsion system (which is related to thrust and mass fraction), mission duration, crew capacity, weight, and size. Depending on these parameters, there are a number of alternatives available for each of these modules. With such information, the dimensions of the design space of the manned lunar landing system and their domains can be briefly summarized in Table 7.9, where the data is obtained from [218]:

Unlike the architecture model developed for the first application demonstration (Figures 7.3 and 7.4), The OPM/H architecture model shown in Figure 7.18 contains both structural variations (modes and system configurations) and object variations (the rest). Even for this small design space, it is still error-prone for humans to discover all possible modes (trajectories) and system configurations. Therefore, a CPN model is developed, for design space exploration, using the approach proposed in Section 5.2.2. Such CPN model is presented in Figure 7.19 (note that shorthand notations have been used for place names and transition names on the CPN model to save space). The place `A_EarthLauS` is the initial place where all variations originate and is given one list type initial token, `[A_EarthLauS]`. After a simulation run, tokens representing the architecture alternatives discovered can be collected at the place `A_EarthlanS`, which is the only end place in this model. The purpose of this CPN model is to assist the discovery of all

Table 7.8. Major Modes of the Manned Lunar Landing System and the Corresponding Spacecraft Configuration

<i>Mode</i>	<i>Sequence of maneuvers</i>	<i>Sequence of states</i>	<i>Spacecraft configuration</i>
EOR	Earth orbit attaining – Earth rendezvous – Earth escape – Moon direct descending – Mission performing – Moon direct ascending – Earth direct descending	Earth launch site – Earth orbit Moon bound – fueled – Midcourse Moon bound – Moon landing site – Moon launching site – Midcourse Earth bound – Earth landing site	LBM, CM,SM
LOR	Earth direct ascending – Moon orbit entering – Separating LEM – Moon braking touch down – Mission performing – Moon orbit attaining – Moon rendezvous – Moon escape – Earth direct descending	Earth launch site – Midcourse Moon bound – Moon orbit Moon bound – LEM separated – Moon landing site – Moon launching site – Moon orbit Earth bound – Docked – Midcourse Earth bound – Earth landing site	SM, CM LEV
DF	Earth direct ascending – Moon direct descending – Mission performing – Moon direct ascending – Earth direct descending	Earth launch site – Midcourse Moon bound – Moon landing site – Moon launching site – Midcourse Earth bound – Earth landing site	LBM, CM,SM

structural variants so the values of each dimension of the design space are not included in this CPN model. In another word, this model is only intended to discover all possible combinations of the OPM processes, objects and links in the OPM/H model shown in Figure 7.18. The final marking at place  $A\_EarthlanS$ , i.e., the token values representing the discovered architecture alternatives are summarized in Table 7.10. In

Table 7.9. Dimensions of the Design Space of the Manned Lunar Landing System

Dimension		Domain
Mode		LOR, EOR, DF
Launch vehicle		C-5, C-8, Liquid <i>Nova</i> , Solid <i>Nova</i>
	Extra Tank	True, False
LBM (propulsion)		Pressured-fed hypergolic, Pressured-fed LOX/LH <sub>2</sub> , Pump-fed LOX/LH <sub>2</sub>
SM (propulsion)		Pressured-fed hypergolic, Pressured-fed LOX/LH <sub>2</sub> , Pump-fed LOX/LH <sub>2</sub>
CM	Mission duration	2day, 7day
	Crew capacity	2 men, 3 men
	Weight (including SM)	11,228 lbs (NAA. 154 in.), 9,148 lbs (STL. 154 in.), 8,400 lbs (STL. 138 in.), 6,728 lbs (AMES. 138 in.)
	diameter	138 inch, 154 inch
LEV	weight	5,475 lbs (Chance-Vought), 3,143 lbs (Manned S/C Center), 5,330 lbs (Grumman-RCA), 5,568 lbs (Martin),

this table, an architecture alternative is shown as state-maneuver sequences along with the required system components (configuration) to support these maneuverers. The cells shaded with the same color share the same mode (state-maneuver sequences) but having different system configurations. An instance model representing a discovered architecture alternative (corresponding to alternative 12 in Table 7.10) is shown in Figure 7.20, which is a LOR system configuration.

The system models (including OPM/H models developed using OPCAT and CPN models developed using CPN Tools) for both the RMS and the Apollo examples, the Python code developed for the RMS example, and a set of sample output archive files for the RMS example are presented in the attached CD as summarized in Appendix C. The statistics obtained from one run of the NSGA-II for the RMS example are also shown in Appendix B.



Table 7.10 Summary of Token Values at the Place A\_Earthlans Representing the Architecture Alternatives Discovered (cont.)

No.	<i>Alternatives</i>
4	1`[SCSM,A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,EEscape,A_MidCMB,SCSM,MOrbitEntering,A_MOrbitMB,SCLBM,Braking,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,MDirectAsc,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
5	1`[SCSM,A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,EEscape,A_MidCMB,SCLBM,MOrbitEntering,A_MOrbitMB,SCLBM,Braking,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,MDirectAsc,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
6	1`[SCSM,A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,EEscape,A_MidCMB,SCLBM,MDirectDescend,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,MDirectAsc,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
7	1`[SCSM,A_EarthLauS,LV,EdirectAsc,A_MidCMB,SCSM,MOrbitEntering,A_MOrbitMB,SCLBM,Braking,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,MDirectAsc,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
8	1`[SCSM,A_EarthLauS,LV,EdirectAsc,A_MidCMB,SCLBM,MOrbitEntering,A_MOrbitMB,SCLBM,Braking,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,MDirectAsc,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
9	1`[SCSM,A_EarthLauS,LV,EdirectAsc,A_MidCMB,SCLBM,MDirectDescend,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,MDirectAsc,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
110	1`[A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,EEscape,A_MidCMB,SCSM,MOrbitEntering,A_MOrbitMB,SCCM,SCLEV,SeperateLEV,A_LEV-Sep,SCLEV,Braking,A_MoonLanS,SCLEV,MissionPerf,A_MoonLauS,SCLEV,MOrbitAttain,A_MOrbitEB,SCCM,SCLEV,MRendez,A_Docked,SCSM,MEscape,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
11	1`[A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,EEscape,A_MidCMB,SCLBM,MOrbitEntering,A_MOrbitMB,SCCM,SCLEV,SeperateLEV,A_LEV-Sep,SCLEV,Braking,A_MoonLanS,SCLEV,MissionPerf,A_MoonLauS,SCLEV,MOrbitAttain,A_MOrbitEB,SCCM,SCLEV,MRendez,A_Docked,SCSM,MEscape,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
12	1`[A_EarthLauS,LV,EdirectAsc,A_MidCMB,SCSM,MOrbitEntering,A_MOrbitMB,SCCM,SCLEV,SeperateLEV,A_LEV-Sep,SCLEV,Braking,A_MoonLanS,SCLEV,MissionPerf,A_MoonLauS,SCLEV,MOrbitAttain,A_MOrbitEB,SCCM,SCLEV,MRendez,A_Docked,SCSM,MEscape,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
13	1`[A_EarthLauS,LV,EdirectAsc,A_MidCMB,SCLBM,MOrbitEntering,A_MOrbitMB,SCCM,SCLEV,SeperateLEV,A_LEV-Sep,SCLEV,Braking,A_MoonLanS,SCLEV,MissionPerf,A_MoonLauS,SCLEV,MOrbitAttain,A_MOrbitEB,SCCM,SCLEV,MRendez,A_Docked,SCSM,MEscape,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]
14	1`[A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,T_Earth,PreLaunch,T_EOrbit,ERendez,A_Fuled,LV,EEscape,A_MidCMB,SCSM,MOrbitEntering,A_MOrbitMB,SCLBM,Braking,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,SCSM,MOrbitAttain,A_MOrbitEB,SCSM,MEscape,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++
15	1`[A_EarthLauS,LV,EOrbitAttn,A_EOrbitMB,LV,T_Earth,PreLaunch,T_EOrbit,ERendez,A_Fuled,LV,EEscape,A_MidCMB,SCLBM,MOrbitEntering,A_MOrbitMB,SCLBM,Braking,A_MoonLanS,SCCM,MissionPerf,A_MoonLauS,SCSM,MOrbitAttain,A_MOrbitEB,SCSM,MEscape,A_MidCEB,SCCM,EDirectDescend,A_EarthLanS]++





## 8. CONCLUSION AND FUTURE WORK

This section first compares the proposed approach with other approaches in solving similar problems and then discusses the strengths, limitation, implementation concerns, and scalability of the proposed approach. The conclusions can then safely be drawn. This section also provides some insights into further development of the proposed approach and directions for future research.

### 8.1. DISCUSSION

**8.1.1. Comparisons with other Approaches for Solving Similar Problems.** For solving the RMS configuration problem, a number of approaches have been discussed in Section 3.3. All these approaches developed some problem-specific models particularly for RMS, which cannot (or are very hard to) be generalized and applied to other systems. Moreover, all these approaches can only take into account very limited aspects in the objective space and limited factors and design variables in the design space due to the lack of a comprehensive (holistic) model. For example, the approaches proposed in [55], [57] used capital cost as the only objective for optimization. Their modeling approaches can only capture some static (or structural) aspects of the system and thus cannot support the assessment of many of the critical performance metrics that are associated with the behavioral aspects of the system (e.g., production rate, processing time, and resource utilization). Petri net based approaches [59–62], on the other hand, cannot capture, and thus cannot be used to assess, pure static (or structural) aspects of the system of interest.

An optimization covering limited dimensions of the objective space while ignoring other, potentially critical, objectives tends to be biased. This is the common drawbacks of traditional optimization approaches that use no comprehensive system model. For example, with capital cost as the only optimization goal, the resulting system might use more dedicated machines, which, although cheap, may not have good modularity or convertibility.

Moreover, optima are often obtained at somewhere near the boundaries in an optimization. Therefore, if there is a change in the boundary or it was poorly estimated, the optimization results obtained there might be invalid. For example, an optimization

towards minimizing capital cost only, may yield solutions that have either too many stages or too many machines in a stage, yet still satisfying the space constraints. Such solutions may leave no room for adding machines (e.g., when demand rate is to be increased) or adding stages (e.g., when a similar type of part with more features are to be added to the part family of the RMS). For scenarios, such as shorter demand periods, diversified products to be produced, or frequently changing demand rate, the capital cost objective is not as important as features such as modularity, convertibility, and scalability. In such cases, it is very important for the system model to capture more design information and to support the assessment of more objectives. The simple problem-specific models proposed in literature are not adequate for such purposes.

When multiple objectives need to be considered, the usual solution is to develop multiple models for the system of interest, each of which being used to optimize certain aspect(s) of the system. The problem with such approaches is that multiple designs will be produced from these optimizations. Each of the design has certain objective(s) optimized. Integration of these designs into one final design not only needs extra efforts but also will almost certainly compromise some objective(s). The optimality of the integrated design is not guaranteed. The system needs to be reevaluated before the performance of the final design can be known.

The holistic modeling approach, along with the search-based architecture development framework, proposed in this research allows more information to be captured in a single holistic model, which also supports CPN-based analyses and verification/validation. Such a model approach enables multiple performance objectives to be optimized and maintained using one integrated system model.

**8.1.2. Strengths and Weaknesses.** In the proposed holistic modeling approach, the OPM, CPN and feature model are used in a complementary way. Together, they offer a full-featured system modeling language. The OPM provides both object-oriented and process-oriented modeling capabilities. Object-oriented modeling is one of the most popular modeling paradigms that can capture a variety of systems, at various levels of abstraction, from various types of perspective. Process-oriented modeling adds to the flexibility of modeling by allowing defining processes independently of objects. This feature makes it possible to specify a system model that leaves the implementation of

some of its processes to be specified by later design cycles. This is particularly useful in the software architecture design. For example, an interface is often defined as abstract type with methods defined only. A class having all the methods defined by an interface is said to implement that interface.

The CPN provides the simulation and model analysis capabilities. The simulation capability is an indispensable means to derive certain performance metrics and to conduct behavior analyses. There exists also a large collection of analysis methods and tools developed for CPN. Such analysis methods and tools not only support detailed architecture analyses, but can also be used to verify and/or validate the model. Such integration of a system model with an analysis model not only avoids the loss of fidelity during model transformation but also eliminates the need to develop a new analysis model when the system model changes.

A holistic model also provides a common foundation to integrate various design activities. By using a holistic model, various design aspects and knowledge from multiple domains can be integrated and represented in one single system model that can be used in multiple design activities. Such integration thus eliminates both the need to transform models between design activities and the efforts to maintain model consistency. “Without a holistic modeling approach, the cost of model construction and the effort required to integrate various system models may present critical concerns to be reflected in the resulting system” [1].

However, there are still some limitations in applying the proposed approach. Some of these limitations and constraints are identified and summarized as follows:

(1) Limitation imposed by the model expressiveness. The standard OPM is not effective in capturing mathematical relationships between entities. With the extension of CPN, it is possible to incorporate programming languages, and therefore mathematical computations, into the modeling. However, many mathematical relations between entities have to be constructed on the basis of state-transition-based structure, which may not be intuitive in some applications and may have limited expressiveness.

Zeigler [221] proposed a categorization scheme that distinguishes formal simulation models into five dimensions i.e., (1) continuous time – discrete time, (2) continuous state – discrete state, (3) deterministic – non-deterministic, (4) autonomous –

non-autonomous, and (5) time invariant – time varying. According to these dimensions, the proposed modeling approach is not capable of capturing fully continuous time and continuous state [27], [70], [159], [160].

(2) Limitation imposed by available analysis, evaluation, or optimization models. Deriving accurate performance measures from architectural model is much more difficult than that from other well-studied problems due to several reasons, such as ambiguity, multiple domains, limited information or knowledge, limited resources or capability for conducting experiments. Heuristic-based optimization cannot guarantee optimum solutions. Analytical models, although very powerful, come at the expense of limited applicability, as many real-world systems are too complex for analytical modeling or evaluation or their solutions are too complex and demand immense computation [222].

(3) Fidelity and computational efficiency. Fidelity issues exist in both architecture models and analysis models. Low-fidelity models might have adverse impact on the final results. The degree of fidelity necessary to guarantee good solutions is difficult to estimate in most cases. The estimation and control of model fidelity are challenging and are not addressed in this research. High fidelity models often demand more computational resources. A trade-off between complexity and fidelity has to be made sometimes.

(4) Accuracy and error control. Errors propagate once generated and it is hard to control the propagation. Errors must be estimated and controlled within tolerable range. Otherwise, the analysis results may not be credible or viable. Accuracy and error control issues are not addressed in this research.

(5) Design space might be incompletely specified. Design options are either identified explicitly by designers or be specified implicitly by constraints. Due to limited knowledge and experience of designers, there is the possibility of overlooking some part of the design space.

In addition, there are some additional concerns to be considered in order to prevent bias or unfaithful results when applying the proposed approach. The proposed modeling approach suffers the same risks as most modeling approaches, such as inappropriate specified requirements, unexpected interactions among the constituent components of the system, low fidelity of architecture models or analysis models,

uncontrolled error propagation, and uncertainty. Many of the limitations identified above also transform to risks in applying the proposed approach. Since the proposed approach incorporates optimization process into the architecture development process, it also suffers the issue of sensitive to boundaries, which is a common issue of most optimization algorithms. Optimization algorithms in general tend to find optimum solutions at the boundary of certain constraints. If such constraints are not accurately specified or subject to uncertainty, the optimum solution obtained might be invalid.

In addition, the architecture generation mechanism currently proposed still provides only rudimentary functionality. Generation of structural variants that are specified by complicated constraints may need some additional problem-specific programming.

**8.1.3. Scalability of the Proposed Approaches.** The proposed approach consists of several components. Hence the discussion of scalability won't be complete without examine each individual components.

(1) Scalability of the search-based architecture development framework. Such a framework, as presented in Section 4.1, is domain independent and problem neutral. It should be able to apply to a broad range of systems at various levels of abstraction.

(2) Scalability of the modeling approach. The scalability of the modeling approach is determined by the expressiveness of the modeling languages adopted and the modeling paradigm assumed by the modeling languages. The proposed modeling approach combines the capabilities of OPM, CPN, and the feature model. It, therefore, roughly has the expressive power that equals to the sum of the capabilities of these three individual modeling languages. The modeling units (objects, processes, links, states, transitions, features, etc.) of these modeling languages are very primitive with little assumption to the entities being modeled. Therefore they can be applied to a broad range of abstract concepts. These individual languages have been proved to have the capability to support the modeling of a huge variety of systems at various levels of abstraction for many types of architectures (including functional architecture, system architecture, and physical architecture). For example, in terms of model resolution, the OPM provides three refinement/abstraction mechanisms as discussed in Section 3.2.2. These

mechanisms enable OPM to recursively specify a system to any desired level of detail without losing legibility and comprehension of the complete system [40].

Object-oriented modeling has been proved to have the capability to model a huge variety of systems and at a broad level of abstraction. The concepts of class and object can be used to model any abstract concepts at any level of abstraction as discussed in Section 3.1 and Section 4.2. The power of the object-oriented paradigm can simply be demonstrated by the fact that UML can be defined by itself, which has been discussed in Section 4.2.2.

As discussed in last section, the OPM modeling is also process-oriented. Specifying a process without identifying the object responsible for it also allows modeling of functional architecture intuitively. As an example, refer to the RMS model developed in Section 7.1.2. Figure 7.3 shows a top level abstraction of the RMS. It can be viewed as a functional architecture that describes the main function of a RMS, i.e., in this case, transforming a raw work piece into a finished product. Such function can be achieved through the interactions of the constituent components of the system and their behaviors. Figure 7.4, elaborating the manufacturing process, can be viewed as a system architecture.

(3) Scalability of the architecture generation mechanism. The architecture generation mechanism proposed in Section 5.2 is modeling language dependent, not problem specific. Therefore, its scalability is the same as the modeling languages that the architecture generation mechanism is based on.

(4) Scalability of the modeling process. The modeling process proposed in Section 4.2.3 is based on the OOA/D and domain engineering, both of which have been proved to have the capability to model a huge variety of systems at various levels of abstractions. Such a modeling process supports hierarchical development and design cycles where each lower level becomes more detailed and refined as the design progresses. This modeling process also includes functional and behavioral mapping, which provides a mechanism to connect models either at different levels of abstraction or at different design stages, for example mapping functional architecture to system architecture and mapping system architecture to physical architecture. These two mechanisms, i.e., mapping models at different design stages and decomposing models

within each stage, allows the hierarchal reduction of ambiguity. The hierarchal reduction of ambiguity has also been addressed in Section 4.2.3. However, the integration of system models across different design stages has not been well addressed yet. Since different design stages (or levels) use different levels of abstraction, thus different design sets and different system models, an explicit mapping between these models should be further studied and developed.

(5) Scalability of the architecture assessment process. The proposed search-based architecture development framework identifies three sub processes within the architecture assessment process (Figure 4.2), i.e., architecture analysis process, architecture selection process, and architecture optimization process. This research discussed some applicable techniques to each of these three processes but has not developed any of such techniques. The claim is that the architect can apply any analysis methods to derive the performance metrics as needed and the architectural model should provide the necessary input to the analysis models. The scalability of the architecture assessment process depends primarily on the chosen analysis methods, along with the available information provided by the system model and the available knowledge regarding the system of interest. For example, the state space analysis cannot scale well to large and complex models.

## **8.2. CONCLUSIONS**

The development of a generative class model and the generation of all instance models enable architectural models to be used as design alternatives in various search algorithms with the aim of discovering optimum architecture designs. Then the concepts and knowledge encoded in architectural models can be processed automatically through computation, thus saving the architect from discovering and evaluating large number of alternatives. As such, an architecture development problem can be converted to a search search-based optimization problem. The search-based architecture development framework implements this idea by integrating architecture modeling, alternative generation, and architecture assessment into a coherence process. Such an architecture development process allows vast design space to be explored before commitment to more detailed design, thus reducing time, cost, and risks of the project and improving design quality.

The proposed modeling approach combines the full features of OPM, CPN and feature models. Therefore, its expressiveness is the sum of these individual languages. More specifically, the proposed modeling approach supports both object-oriented and process-oriented paradigm as provided by OPM. Such OPM is supplemented by CPN for execution semantics. So it has state-transition-based execution semantics supporting discrete-event system simulation. The incorporation of CPN into the architecture modeling also allows the developed system model to be also used as a analysis model. A large collection of analysis methods and tools developed for CPN can be utilized for strong model analysis, verification, and validation. Such OPM is also extended to support the feature model concepts so it can model a collection of systems. In summary, such modeling approach not only can model a broad range of systems at various levels of abstraction but also can support the needs of search-based architecture development by providing both comprehensive information needed for architecture reasoning and the design space specification needed for architecture alternative generation.

The other components in the proposed approach, including the search-based architecture development framework, the architecture alternative generation mechanism, and the suggested implementation architecture assessment, are all domain independent and problem neutral. Therefore, the entire approach set is generic and should be able applied to a broad range of systems that can be specified using conceptual models with either object-oriented or process-oriented paradigm. Still, a large number of case studies are needed to further examine the capabilities of the proposed approach.

Architectures can arise within a variety of scenarios [223]. These include the deliberate design of a system from scratch, the evolution of a design from previous designs, the expansion of smaller systems, or the exploration of form and behavior requirements. The proposed architecture development approach can facility both incremental design through hierarchical refinement and adapting an existing architecture to new or changing design needs.

### **8.3. FUTURE WORK**

The architecture generation mechanism proposed in Section 5.2 need to be further researched to allow fully automatic generation of all types of architecture alternatives



with little or no need of problem-specific programming. One possible way to achieve such capabilities is to develop a mapping from OPM class model directly to a suitable logical representation, such as the propositional logic, the constraint programming, and the description logic. The strengths of logical representations are their support of computational implementation and their capabilities to process complexity dependencies between features. With such logical representations, it is possible to use the off-the-self solvers to generate all possible architecture alternatives and perform other automatic model analyses. Alternatively, a parser that translates an OPM class model to a feature model can be developed so that the current tool support and analysis methods of the feature model can be utilized. Furthermore, since a design space represented by the feature model is a tree-like structure, generative algorithms in conjunction with tree-based algorithms might be useful in discovering all possible architecture alternatives. In addition, algorithms need to be developed to prove the completeness (i.e., covering the entire design space) of the generated alternatives (closure).

All design variables can be identified through design space analysis using the feature model information. Algorithms can be developed to conduct automatic design space analysis. Since a chromosome needs only to capture these design variables. A unified chromosome representation scheme can be developed to automate the process from OPM/H class model development to chromosome encoding. With the fully automated alternative generation and chromosome encoding, a fully automated search-based architecture development process can be achieved.

In order to capture complex design space, more advanced feature model concepts should be incorporated and implemented. For example the full support of feature attribute [209–213] is needed. According to [14], the attribute of a feature should consist at least of a name, a domain and a value. The example studied in Section 7.1 also reveals the need for capturing complex relationships and constraints between features and feature attributes [14], [209], [213].

The ABCD language, as a higher level language for specifying a Petri net, makes writing a Petri net specification easier by hiding programming implementation details. However, it also creates an extra level of formality that is not necessary in the search-based architecture development process since a CPN model should ideally be constructed

directly from the OPM/H model. Moreover, using the ABCD language for specifying CPN models adds another layer of computation between the CPN model specification and the computational model building. In the case of producing a large number of model instances, like in the proposed the search-based architecture development process, such a layer of computation wastes a lot of computation resources in compiling the ABCD specification. Especially, when model instances share a large portion in common, rebuilding the entire CPN computational model for each instance of ABCD specification is not necessary. Therefore the use of ABCD for CPN specification should be removed from the implementation of the search-based architecture development framework.

In the current implementation of the modeling approach, the simulation of CPN models depends on the ABCD language layer so the transformation from OPM/H model to CPN model is not an automatic process yet. A fully automatically transformation can be developed once the dependency on the ABCD language layer is removed.

Furthermore, a hybrid OPM-CPN modeling language can be developed and implemented to fully integrate the execution semantics and simulation capability of CPN into the OPM modeling such that an OPM/H model can be executed directly. In addition, inclusion of BBN can also be considered in this hybrid modeling language so that uncertainty can be modeled and managed effectively.

As discussed in Section 8.12. Model fidelity affects the accuracy of the performance metrics derived from the system model. The estimation and control of model fidelity should be further studied. Moreover, the error propagation issue and the uncertainty management should also be studied. Furthermore, since architecture models are special types of design alternatives, methods need to be developed to support the sensitivity analysis in the context of optimization algorithms used for architecture optimization.

In addition, the support of traceability analysis based on the proposed architecture development framework deserves further study. The estimation of the impact of changes in architecture to performance metrics can offer several benefits. For example, (1) it provides designers a better understanding of the system of interest, (2) it provides designers the insights into the relative importance of certain part of the system to certain performance metrics, (3) it can point out the direction of possible architecture

improvement, (4) it allows incremental development, in which case, only partial update of the system is desired. The last point is especially important when updating an existing system where the system architecture is expected to keep relative stable and only partial improvement is expected (or can be afforded).

Besides the genetic algorithms, other meta-heuristic search algorithms also deserve a try. As discussed in Section 4.3.3, each of these optimization algorithms has its own merit. Depending on the problem to be solved and the data available, some optimization algorithms may perform better than others.

On the other hand, with the incorporation of feature model concepts, the holistic modeling approach also facilitates the management of architecture variants, including the variants of subsystems and components. This can in turn facilitate the system family development and management. Such types of application can be further explored.

APPENDIX A.

MACHINE PROCESSING INFORMATION FOR THE RMS DESIGN EXAMPLE

Table A1. Operations data for part ANC-90 ([55])

Feature	Description	Operation	Op. ID	TAD candidates	Tool candidates
F1	Planar surface	Milling	OP1	+Z	C6, C7, C8
F2	Planar surface	Milling	OP2	Z	C6, C7, C8
F3	Four holes arranged as a replicated feature	Drilling	OP3	+Z, Z	C2
F4	A step	Milling	OP4	+X, Z	C6, C7
F5	A protrusion (rib)	Milling	OP5	+Y, Z	C7, C8
F6	A protrusion	Milling	OP6	Y, Z	C7, C8
F7	A compound hole	Drilling	OP7	Z	C2, C3, C4
		Reaming	OP8		C9
		Boring	OP9		C10
F8	Six holes arranged in a replicated feature	Drilling	OP10'	Z	C1
		Tapping	OP11'		C5
F9	A step	Milling	OP12	X, Z	C6, C7

Table A2. Operations Data for Part ANC-101([55])

Feature	Description	Operation	Op. ID	TAD candidates	Tool candidates
F1	Planar surface	Milling	OP1	+Z	C6, C7, C8
F2	Planar surface	Milling	OP2	Z	C6, C7, C8
F3	Four holes arranged as a replicated feature	Drilling	OP3	+Z, Z	C2
F4	A step	Milling	OP4	+X, Z	C6, C7
F5	A protrusion (rib)	Milling	OP5	+Y, Z	C7, C8
F6	A protrusion	Milling	OP6	Y, Z	C7, C8
F7	A compound hole	Drilling	OP7	Z	C2, C3, C4
		Reaming	OP8		C9
		Boring	OP9		C10
F8	Nine holes arranged in a replicated feature	Drilling	OP10	Z	C1
		Tapping	OP11		C5
F9	A step	Milling	OP12	X, Z	C6, C7
F10	Two pockets arranged as a replicated feature	Milling	OP13	+X	C6, C7, C8
F11	A boss	Milling	OP14	a	C7, C8
F12	A compound hole	Drilling	OP15	a	C2, C3, C4
		Reaming	OP16		C9
		Boring	OP17		C10
F13	A pocket	Milling	OP18	X	C7, C8
F14	A compound hole	Reaming	OP19	+Z	C9
		Boring	OP20		C10

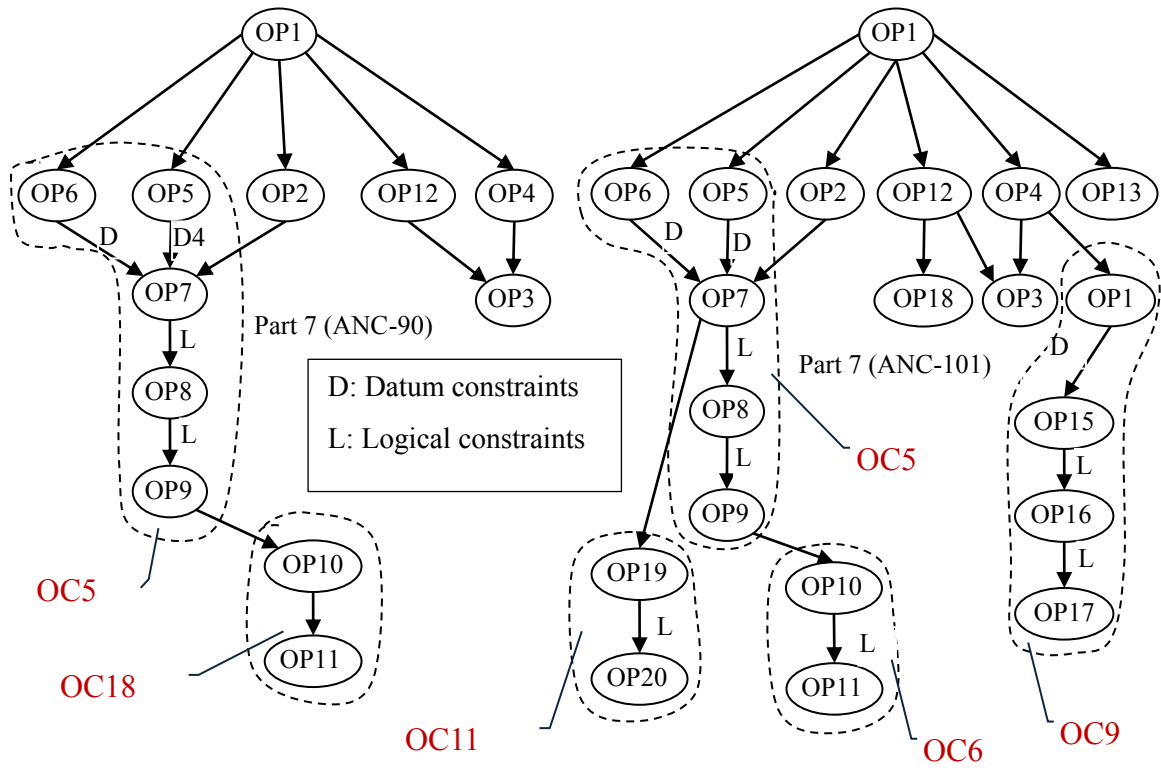


Figure A1. Operation Precedence Graph for the Two Parts ([55])

Table A3. Operation Cluster Definitions for Part ANC-90 ([55])

<i>Operation cluster</i>	<i>Operations</i>
OC1	[OP1]
OC2	[OP2]
OC3	[OP3]
OC4	[OP4]
OC5	[OP5, OP6, OP7, OP8, OP9]
OC60	[OP10', OP11']
OC7	[OP12]

Table A4. Operation Cluster Definitions for Part ANC-101 ([55])

<i>Operation cluster</i>	<i>Operations</i>
OC1	[OP1]
OC2	[OP2]
OC3	[OP3]
OC4	[OP4]
OC5	[OP5, OP6, OP7, OP8, OP9]
OC6	[OP10, OP11]
OC7	[OP12]
OC8	[OP13]
OC9	[OP14, OP15, OP16, OP17]
OC10	[OP18]
OC11	[OP19, OP20]

Table A5. Available/Obtainable Resources Description and Cost ([55])

Machine (M)		Machine configuration (MC)		Initial cost
Code	Description	Code	Description	(in 1000 USD)
M1	Reconfigurable horizontal milling machine	MC1 <sub>1</sub>	Three-axis with one spindle	860
		MC1 <sub>2</sub>	Three-axis with two spindles	1140
		MC1 <sub>3</sub>	Three-axis with three spindles	1420
		MC1 <sub>4</sub>	Three-axis with four spindles	1700
		MC1 <sub>5</sub>	Four-axis with one spindle	1010
M2	Reconfigurable drilling press	MC2 <sub>1</sub>	One spindle	385
		MC2 <sub>2</sub>	Two spindles	555
		MC2 <sub>3</sub>	Three spindles	725
		MC2 <sub>4</sub>	Four spindles	895



Table A6. Time and Production Rate Information for Different M-MC-OS Combinations ([55])

Operation cluster setup (OS)	Standard time in seconds (production rate in parts/h)											
	M1						M2					
Code	Operation clusters (OCs)	MC11	MC12	MC13	MC14	MC15	MC21	MC22	MC23	MC24		
OS1	[OC1]	30	30	30	30 (480)	30	X	X	X	X		
OS2	[OC2]	20	20	20	20 (720)	20	X	X	X	X		
OS3	[OC3]	30	30	30	30 (480)	30	30	30	30	30 (480)		
OS4	[OC4]	20	20	20	20 (720)	20	X	X	X	X		
OS5	[OC5]	X	X	X	X	60 (60)	X	X	X	X		
OS6	[OC6]	120	120	120	120 (120)	120	120	120	120	120		
OS18	[OC60]	90 (40)	90 (80)	90	90 (160)	90 (40)	90 (40)	90	90	90 (160)		
OS7	[OC7]	18	18	18	18 (800)	18	X	X	X	X		
OS8	[OC8]	X	X	X	X	20	X	X	X	X		
OS9	[OC9]	X	X	X	X	40 (90)	X	X	X	X		
OS10	[OC10]	X	X	X	X	18	X	X	X	X		
OS11	[OC11]	24	24	24	24 (600)	24	X	X	X	X		
OS12	[OC3, OC11]	60 (60)	60	60	60 (240)	60 (60)	X	X	X	X		
OS13	[OC8, OC10]	30	30	30	30 (480)	30	X	X	X	X		
OS14	[OC2, OC4, OC7]	40 (90)	40	40	40 (360)	40 (90)	X	X	X	X		
OS15	[OC2, OC3, OC4, OC7]	60 (60)	60	60	60 (240)	60 (60)	X	X	X	X		
OS16	[OC2, OC4, OC7, OC8, OC10]	X	X	X	X	60 (60)	X	X	X	X		
OS17	[OC2, OC3, OC4, OC7, OC8, OC10]	X	X	X	X	90 (40)	X	X	X	X		

APPENDIX B.  
SELECTED RESULTS OF THE RMS DESIGN EXAMPLE

Table B1. Statistics of the Results from Running the NSGA-II for the RMS Configuration Problem

<i>Gen.</i>	<i>System Cost – Best Individual</i>	<i>Unit Production Time – Best Individual</i>	<i>Population Median</i>	<i>Population Average</i>	<i>Population Standard Deviation</i>
0	27.39	18.04	21.03	27.20	10.27
1	22.62	17.88	21.17	24.98	8.18
2	26.44	17.04	20.70	24.06	7.56
3	26.44	17.04	21.16	23.69	7.22
4	22.19	16.10	20.41	23.58	7.43
5	22.19	16.10	20.41	23.34	7.15
6	22.19	16.10	20.41	22.91	6.81
7	22.19	16.10	20.27	22.89	6.95
8	22.19	16.10	19.01	22.93	7.12
9	22.19	16.10	19.02	22.87	7.14
10	22.19	16.10	19.29	22.76	7.04
11	22.19	16.10	19.01	22.46	6.95
12	22.19	16.10	19.48	22.09	6.69
13	22.19	16.10	19.48	21.81	6.45
14	22.19	16.10	19.33	21.47	6.01
15	22.19	16.10	19.33	21.24	5.82
16	22.19	16.10	19.33	21.31	6.00
17	22.19	16.10	19.33	21.27	6.03
18	22.19	16.10	19.33	20.94	5.66
19	22.19	16.10	19.10	21.04	5.89
20	22.19	16.10	18.64	20.99	5.87
21	22.19	16.10	18.62	20.87	5.78
22	22.19	16.10	18.49	20.85	5.79
23	22.19	16.10	18.40	20.86	5.80
24	22.19	16.10	18.40	20.84	5.81
25	22.19	16.10	18.40	20.69	5.52
26	22.19	16.10	18.40	20.73	5.58
27	22.19	16.10	18.40	20.34	5.09
28	22.19	16.10	18.40	20.33	5.09
29	22.19	16.10	18.40	20.30	5.08
30	22.19	16.10	18.52	20.30	5.08
31	22.19	16.10	18.52	20.26	5.05
32	22.19	16.10	18.15	20.14	4.93
33	22.19	16.10	18.15	20.13	4.94
34	22.19	16.10	18.15	20.14	5.00
35	20.17	16.20	18.19	19.99	4.83
36	20.17	16.20	18.19	19.96	4.81

Table B1. Statistics of the Results from Running the NSGA-II for the RMS Configuration Problem (cont.)

<i>Gen.</i>	<i>System Cost – Best Individual</i>	<i>Unit Production Time – Best Individual</i>	<i>Population Median</i>	<i>Population Average</i>	<i>Population Standard Deviation</i>
37	20.17	16.20	18.19	19.98	4.84
38	20.17	16.20	18.19	19.94	4.83
39	20.17	16.20	18.19	19.94	4.83
40	20.17	16.20	18.35	19.85	4.71
41	20.17	16.20	18.35	19.72	4.47
42	20.17	16.20	18.35	19.68	4.42
43	20.17	16.20	18.35	19.75	4.54
44	20.17	16.20	18.19	19.68	4.43
45	20.17	16.20	18.17	19.57	4.12
46	20.17	16.20	18.15	19.35	3.56
47	20.17	16.20	18.04	19.39	3.63
48	20.17	16.20	17.94	19.46	3.81
49	20.17	16.20	17.94	19.38	3.65
50	20.17	16.20	17.94	19.32	3.55
51	20.17	16.20	18.14	19.25	3.43
52	20.17	16.20	18.32	19.23	3.41
53	20.17	16.20	18.32	19.20	3.39
54	20.17	16.20	18.24	19.16	3.31
55	20.17	16.20	18.15	19.18	3.36
56	20.17	16.20	18.14	19.17	3.37
57	20.17	16.20	18.13	19.16	3.37
58	20.17	16.20	18.15	19.12	3.34
59	20.17	16.20	18.13	19.12	3.35
60	20.17	16.20	18.13	19.10	3.34
61	20.17	16.20	18.13	18.92	3.01
62	20.17	16.20	18.34	18.89	2.98
63	20.17	16.20	18.42	18.86	2.95
64	20.17	16.20	18.42	18.83	2.92
65	20.17	16.20	18.34	18.80	2.89
66	20.17	16.20	18.34	18.82	2.91
67	20.17	16.20	18.34	18.83	2.92
68	20.17	16.20	18.34	18.82	2.92
69	20.17	16.20	18.34	18.84	2.93
70	20.17	16.20	18.43	18.79	2.89
71	20.17	16.20	18.43	18.71	2.75
72	20.17	16.20	18.43	18.70	2.74
73	20.17	16.20	18.36	18.69	2.75

Table B1. Statistics of the Results from Running the NSGA-II for the RMS Configuration Problem (cont.)

<i>74</i>	<i>20.17</i>	<i>16.20</i>	<i>18.36</i>	<i>18.68</i>	<i>2.74</i>
75	20.17	16.20	18.36	18.66	2.73
76	20.17	16.20	18.43	18.67	2.77
77	20.17	16.20	18.04	18.57	2.63
78	20.17	16.20	18.04	18.57	2.63
79	20.17	16.20	18.04	18.60	2.69
80	20.17	16.20	18.04	18.62	2.71
81	20.17	16.20	18.04	18.65	2.77
82	20.17	16.20	18.04	18.67	2.80
83	20.17	16.20	18.04	18.70	2.84
84	20.17	16.20	18.04	18.66	2.82
85	20.17	16.20	18.00	18.64	2.80
86	20.17	16.20	18.04	18.67	2.84
87	20.17	16.20	18.04	18.77	3.04
88	20.17	16.20	18.04	18.77	3.04
89	20.17	16.20	18.04	18.84	3.19
90	20.17	16.20	18.18	18.81	3.12

APPENDIX C.  
PYTHON CODE, OUTPUT ARCHIVE FILES,  
AND OPM AND CPN MODELS ON CD-ROM

## INTRODUCTION

Included with this dissertation is a CD-ROM, which contains the PYTHON CODE for both the generic implementation of the proposed approaches and the problem specific code for the RMS example (as listed in Table C1), the output archive files after running the program (as listed Table C2), and the system models developed for both the RMS design and the Apollo program (as listed Table C3). Each module of the PYTHON CODE has been developed using PYTHON 2.7.2 for Windows 32 bit. All output archive files are automatically generated by the program in .csv format. The system models for both example problems are developed using both OPCAT and CPN Tools. The contents of the CD-ROM are summarized in Tables C1, C2, and C3.

Table C1. List of Developed Python Code, OPM Models, and CPN Models

<i>Module</i>	<i>Description</i>
RMS_GA.py	Top level module for loading input data and CPN base model, setting the GA parameters, plotting results, and saving archive files.
RMS_GA_proble m.py	Module for formulating the problem to be solved by GA (e.g., chromosome encoding and decoding, alternative generation, candidate assessment, etc.).
RMS_DataPcs.py	Module for preprocessing input data and generating attribute values for design alternative.
RMS_data_provid er.py	Module for specifying part and machine processing data. it creates a rms object of type Rms that contains the processed data and some related functions
nets.py	Modified Petri net module (to replace the original one located at Python27\Lib\site-packages\snakes\net.py).
simulngui.py	Alternative Petri net simulation engine that suppresses the GUI. Otherwise it is the same as the simul.py below
simul.py	Modified Petri net simulation module (to replace the original one located at Python27\Lib\site-packages\snakes\utilits\abcd\simul.py).
abcd_build_simul. py	Build and simulate a Petri net. It is a modified version of the snakes\utilits\abcd\main.py. it use the simulngui.py as the simulation engine
main.py	Modified main module for organizing the tasks of compiling and simulating a Petri net (to replace the original one located at Python27\Lib\site-packages\snakes\utilits\abcd\main.py).
runPN.py	A program that allows user to set Petri net simulation parameters and test run a Petri net simulation
ec.py	Modified evolutionary computation module (to replace the original one located at Python27\Lib\site-packages\inspyred\ec.py).
crossovers.py	Modified crossover operator (to replace the original one located at Python27\Lib\site-packages\inspyred\crossovers.py).
mutators.py	Modified mutation operator module (to replace the original one located at Python27\Lib\site-packages\inspyred\mutators.py).
e_opm.py	OPM/H module for creating and editing OPM/H models.



Table C2. List of Output Archive Files Generated from Running the Program

<i>File name</i>	<i>Description</i>
rms_ec_individuals.csv	The entire individuals (the entire population from all generations) generated and evaluated by one run of the NSGA-II for the RMS example, along with their objective function values.
rms_ec_statistics.csv	Key statistics (worst, best, median, average, and standard deviation) of each generation obtained from running the NSGA-II for the RMS example.
RMS_StatHistory.txt	State history from one simulation run of a CPN model for the RMS example.

Table C3. System Models

<i>Models</i>	<i>Description</i>
RMS.opz	OPM system architecture model of the RMS developed using OPCAT v3.1.
RMS.cpn	CPN model for the RMS developed using CPN Tools v 3.2.2.
RMS.abcd	CPN model for the RMS developed using ABCD language.
Apollo.opz	OPM system architecture model of the manned lunar landing system for the Apollo program example developed using OPCAT v3.1.
Apollo.cpn	CPN model for generating the design space of the manned lunar landing system developed using CPN Tools v 3.2.2.

**BIBLIOGRAPHY**

- [1] H.-Y., Benjamin Koo, “A Meta-Language for Systems Architecting,” Massachusetts Institute of Technology, 2005.
- [2] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and others, “Reformulating software engineering as a search problem,” in *Software, IEE Proceedings-*, 2003, vol. 150, pp. 161–175.
- [3] K. Czarnecki, “Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models,” Ph.D. Dissertation, Technical University of Ilmenau, 1998.
- [4] K. Czarnecki, K. Østerbye, and M. Völter, “Generative Programming,” in *Object-Oriented Technology ECOOP 2002 Workshop Reader*, vol. 2548, J. Hernández and A. Moreira, Eds. Springer Berlin / Heidelberg, 2002, pp. 15–29.
- [5] M. Harman, “The Current State and Future of Search Based Software Engineering,” in *2007 Future of Software Engineering*, Washington, DC, USA, 2007, pp. 342–357.
- [6] L. Relu, “Evolutionary Computing in Search-Based Software Engineering,” Master’s Thesis, Lappeenranta University of Technology, Finland, 2004.
- [7] M. Harman and B. F. Jones, “Search-Based Software Engineering,” *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.
- [8] M. Harman, S. A. Mansouri, and Y. Zhang, *Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications*. 2009.
- [9] G. Dos Reis and J. Järvi, “What is Generic Programming?,” *Library-Centric Software Design (LCSD’05)*, p. 1, 2005.
- [10] D. Musser and A. Stepanov, “Generic Programming,” *Symbolic and Algebraic Computation*, pp. 13–25, 1989.
- [11] J. Dehnert and A. Stepanov, “Fundamentals of Generic Programming,” in *Generic Programming*, vol. 1766, M. Jazayeri, R. Loos, and D. Musser, Eds. Springer Berlin / Heidelberg, 2000, pp. 1–11.
- [12] C. H. Dagli, A. Singh, J. Dauby, and R. Wang, “Smart Systems Architecting: Computational Intelligence Applied to Trade Space Exploration and System Design,” *Systems Research Forum*, vol. 3, no. 2, pp. 101–120, Dec. 2009.

- [13] A. Alfari, “The Evolutionary Design Model (EDM) for the Design of Complex Engineered Systems: Masdar City as a Case Study,” Massachusetts Institute of Technology, 2009.
- [14] D. Benavides, S. Segura, and A. Ruiz-Cortés, “Automated Analysis of Feature Models 20 Years Later: A Literature Review,” *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.
- [15] O. Rähkä, “Applying Genetic Algorithms in Software Architecture Design,” *University of Tampere, Department of Computer Sciences*, 2008.
- [16] M. Amoui, S. Mirarab, S. Ansari, and C. Lucas, “A Genetic Algorithm Approach to Design Evolution Using Design Pattern Transformation,” *International Journal of Information Technology and Intelligent Computing*, vol. 1, no. 1, p. 2, 2006.
- [17] A. Asllani and A. Lari, “Using Genetic Algorithm for Dynamic and Multiple Criteria Web-Site Optimizations,” *European journal of operational research*, vol. 176, no. 3, pp. 1767–1777, 2007.
- [18] D. Doval, S. Mancoridis, and B. S. Mitchell, “Automatic Clustering of Software Systems Using a Genetic Algorithm,” in *Software Technology and Engineering Practice, 1999. STEP’99. Proceedings*, 1999, pp. 73–81.
- [19] M. Harman, R. M. Hierons, and M. Proctor, “A New Representation And Crossover Operator For Search-based Optimization Of Software Modularization,” in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, USA, 2002, pp. 1351–1358.
- [20] V. Le Hanh, K. Akif, Y. Le Traon, and J. M. Jézéque, “Selecting an Efficient OO Integration Testing Strategy: an Experimental Comparison of Actual Strategies,” *ECOOP 2001—Object-Oriented Programming*, pp. 381–401, 2001.
- [21] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, 1st ed. Addison-Wesley Professional, 1999.
- [22] W. G. Griswold and D. Notkin, “Automated Assistance for Program Restructuring,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 2, no. 3, pp. 228–269, 1993.
- [23] T. Mens and T. Tourwé, “A Survey of Software Refactoring,” *Software Engineering, IEEE Transactions on*, vol. 30, no. 2, pp. 126–139, 2004.
- [24] O. Seng, J. Stammel, and D. Burkhart, “Search-based determination of refactorings for improving the class structure of object-oriented systems,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1909–1916.

- [25] M. Ó. Cinnéide, “Towards Automated Design Improvement Through Combinatorial Optimisation,” *IN PROC. WORKSHOP ON DIRECTIONS IN SOFTWARE ENGINEERING ENVIRONMENTS*, 2004.
- [26] E. Visser, “A Survey Of Rewriting Strategies in Program Transformation Systems,” *Electronic Notes in Theoretical Computer Science*, vol. 57, pp. 109–143, 2001.
- [27] H. Van Vliet, H. Van Vliet, and J. Van Vliet, *Software Engineering: Principles and Practice*. Citeseer, 2008.
- [28] “OMG Unified Modeling Language (OMG UML), Infrastructure,” Object Management Group, formal/2011-08-05.
- [29] G. Booch, R. A. Maksimchuk, and M. W. Engle, *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 2007.
- [30] L. Geyer, “Feature Modeling Using Design Spaces,” in *Proceedings of the 1st German Workshop on Software Product Lines. Kaiserslautern: Fraunhofer IESE*, 2000, vol. 3539.
- [31] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. 1990.
- [32] K. Czarnecki, S. Helsen, and U. Eisenecker, “Formalizing Cardinality-Based Feature Models and Their Specialization,” in *Software Process: Improvement and Practice*, 2005, p. 2005.
- [33] “OMG Unified Modeling Language (OMG UML), Superstructure,” Object Management Group, formal/2011-08-05.
- [34] “UML 2.4 Diagrams Overview,” *UML 2.4 Diagrams Overview*. [Online]. Available: <http://www.uml-diagrams.org/uml-24-diagrams.html>.
- [35] G. Engels and L. Groenewegen, “Object-Oriented Modeling: a Roadmap,” in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 103–116.
- [36] OMG, “OMG System Modeling Language (OMG SysML) v1.2,” Object Management Group, Standard formal/2010-06-01, 2010.
- [37] D. Dori, *Object-Process Methodology*, 1st ed. Springer, 2002.
- [38] N. R. Soderborg, E. F. Crawley, and D. Dori, “System Function and Architecture: OPM-Based Definitions and Operational Templates,” *Commun. ACM*, vol. 46, no. 10, pp. 67–72, 2003.

- [39] I. Reinhartz-Berger and D. Dori, "Object-Process Methodology (OPM) vs. UML: a Code Generation Perspective," in *Proceedings of CAiSE'04 Workshops*, 2004, pp. 275–286.
- [40] D. Dori and I. Reinhartz-Berger, "An OPM-Based Metamodel of System Development Process," *Conceptual Modeling-ER 2003*, pp. 105–117, 2003.
- [41] I. Reinhartz-Berger and D. Dori, "A Reflective Metamodel of Object-Process Methodology: the System Modeling Building Blocks," *Business Systems Analysis with Ontologies*, pp. 130–173, 2005.
- [42] D. Dori, I. Reinhartz-Berger, and A. Sturm, "Developing Complex Systems with Object-Process Methodology using OPCAT," *Conceptual Modeling-ER 2003*, pp. 570–572, 2003.
- [43] D. Dori, I. Reinhartz-Berger, and A. Sturm, "OPCAT-a bimodal CASE tool for object-process based system development," in *5th International Conference on Enterprise Information Systems (ICEIS 2003)*, 2003, pp. 286–291.
- [44] D. Wikarski, *Petri Net Tools: a Comparative Study*. Technische Universität Berlin, Fachbereich 13, Informatik, 1997.
- [45] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice Hall, 1981.
- [46] S. Carlsen, "Conceptual modeling and composition of flexible workflow models," Citeseer, 1997.
- [47] C. A. Petri, "Kommunikation mit Automaten," *PhD, University of Bonn, West Germany*, 1962.
- [48] J. L. Peterson, "Petri Net Theory and the Modeling of Systems.," *PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, NJ 07632, 1981, 290*, 1981.
- [49] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use - Volume 1, 2, & 3*, Second Edition, 3 VOLUME SET. Springer Verlag, 1996.
- [50] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel, "Reconfigurable Manufacturing Systems," *CIRP Annals-Manufacturing Technology*, vol. 48, no. 2, pp. 527–540, 1999.
- [51] R. Hill, "File:RMS schematic.gif - Wikipedia, the free encyclopedia," *Reconfigurable Manufacturing System*, 1999. [Online]. Available: [http://en.wikipedia.org/wiki/File:RMS\\_schematic.gif](http://en.wikipedia.org/wiki/File:RMS_schematic.gif). [Accessed: 10-Dec-2011].
- [52] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable Manufacturing Systems: Key to Future Manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.

- [53] H. A. ElMaraghy, “Flexible and Reconfigurable Manufacturing Systems Paradigms,” *International journal of flexible manufacturing systems*, vol. 17, no. 4, pp. 261–276, 2005.
- [54] M. Shpitalni and Y. Koren, “Design of Reconfigurable Manufacturing Systems,” *Journal of Manufacturing Systems*, vol. 29, no. 4, pp. 130–141, Oct. 2010.
- [55] A. Youssef and H. A. ElMaraghy, “Modelling and Optimization of Multiple-Aspect RMS Configurations,” *International journal of production research*, vol. 44, no. 22, pp. 4929–4958, 2006.
- [56] A. M. A. Youssef and H. A. ElMaraghy, “Optimal Configuration Selection for Reconfigurable Manufacturing Systems,” *International Journal of Flexible Manufacturing Systems*, vol. 19, no. 2, pp. 67–106, 2007.
- [57] J. Dou, X. Dai, and Z. Meng, “Graph Theory-Based Approach to Optimize Single-Product Flow-Line Configurations of RMS,” *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 9, pp. 916–931, 2009.
- [58] L. Tang, Y. Koren, D. M. Yip-Hoi, and W. Wang, “Computer-Aided Reconfiguration Planning: An Artificial Intelligence-Based Approach,” *J. Comput. Inf. Sci. Eng.*, vol. 6, no. 3, pp. 230–240, 2006.
- [59] Z. Cai and X. Yan, “Case Study on Reconfiguration Algorithm for Reconfigurable Manufacturing System [J],” *Journal of Computer Aided Design & Computer Graphics*, vol. 2, 2003.
- [60] Z. Meng, J. Li, and X. Dai, “Hierarchical Petri net modelling of reconfigurable manufacturing systems with improved net rewriting systems,” *International Journal of Computer Integrated Manufacturing*, vol. 22, no. 2, pp. 158–177, Feb. 2009.
- [61] L. Zhang and B. Rodrigues, “Modelling Reconfigurable Manufacturing Systems with Coloured Timed Petri Nets,” *International Journal of Production Research*, vol. 47, no. 16, pp. 4569–4591, Aug. 2009.
- [62] L. C. Wang and S. Y. Wu, “Modeling with Colored Timed Object-Oriented Petri Nets for Automated Manufacturing Systems,” *Computers & industrial engineering*, vol. 34, no. 2, pp. 463–480, 1998.
- [63] X. Dai, Z. Meng, and J. Li, “Improved Net Rewriting System-Based Approach to Model Reconfiguration of Reconfigurable Manufacturing Systems,” *The International Journal of Advanced Manufacturing Technology*, vol. 37, no. 11, pp. 1168–1189, Jul. 2008.
- [64] D. M. Buede, *The Engineering Design of Systems: Models and Methods*, 2nd ed. Wiley, 2009.

- [65] W. L. Chapman, A. T. Bahill, and A. W. Wymore, *Engineering Modeling and Design*. CRC Press, 1992.
- [66] A. Singh, “Architecture Value Mapping: Using Fuzzy Cognitive Maps as a Reasoning Mechanism for Multi-criteria Conceptual Design Evaluation,” Ph.D. Dissertation, Missouri University of Science and Technology, 2011.
- [67] Wikipedia contributors, “Software Design Pattern,” *Wikipedia, the free encyclopedia*. Wikimedia Foundation, Inc., 05-Sep-2012.
- [68] E. Zitzler and L. Thiele, “Multiobjective Optimization Using Evolutionary Algorithms—a Comparative Case Study,” in *Parallel Problem Solving from Nature—PPSN V*, 1998, pp. 292–301.
- [69] K. Deb, *Multi-objective Optimization using Evolutionary Algorithms*, vol. 16. Wiley, 2001.
- [70] Wikipedia contributors, “Abstraction (computer science),” *Wikipedia, the free encyclopedia*. Wikimedia Foundation, Inc., 02-Aug-2012.
- [71] “WordNet Search - 3.1.” [Online]. Available: <http://wordnetweb.princeton.edu/perl/webwn?s=abstraction>. [Accessed: 07-Sep-2012].
- [72] L. Floridi and T. M. O. Abstraction, *Levellism and the Method of Abstraction*.
- [73] B. Liskov and S. Zilles, “Programming with Abstract Data Types,” *SIGPLAN Not.*, vol. 9, no. 4, pp. 50–59, Mar. 1974.
- [74] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.
- [75] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, 1st ed. Addison-Wesley Professional, 1999.
- [76] D. F. D’Souza and A. C. Wills, *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*, 1st ed. Addison-Wesley Professional, 1998.
- [77] B. P. Douglass, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley Professional, 1999.
- [78] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: a Use Case Driven Approach*. Addison-Wesley, 1992.
- [79] J. Rumbaugh, *Object-Oriented Analysis and Design With Applications*. Prentice Hall, 1991.

- [80] P. Kruchten, *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2004.
- [81] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley Professional, 2003.
- [82] K. Czarnecki, "Generative Programming: Methods, Techniques, and Applications Tutorial Abstract," in *Software Reuse: Methods, Techniques, and Tools*, vol. 2319, C. Gacek, Ed. Springer Berlin / Heidelberg, 2002, pp. 477–503.
- [83] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*. Addison-Wesley Professional, 1997.
- [84] H. Van Dyke Parunak, R. Savit, and R. Riolo, "Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide," in *Multi-Agent Systems and Agent-Based Simulation*, 1998, pp. 277–283.
- [85] P. Fishwick, *Simulation Model Design and Execution: Building Digital Worlds*, 1st ed. Prentice Hall, 1995.
- [86] J. P. Dauby, "Assessing System Architectures: the Canonical Decomposition Fuzzy Comparative Methodology," Ph.D. Dissertation, Missouri University of Science and Technology, Rolla, MO, USA, 2011.
- [87] R. Kazman, M. Klein, and P. Clements, *ATAM: Method for Architecture Evaluation*. Citeseer, 2000.
- [88] Y. Akao, *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Pr, 2004.
- [89] J. B. ReVelle, J. W. Moran, and C. A. Cox, *The QFD Handbook*, 1st ed. Wiley, 1998.
- [90] T. L. Saaty, *Analytical Hierarchy Process*. Wiley Online Library, 1980.
- [91] T. L. Saaty, *The Analytic Network Process: Decision Making With Dependence and Feedback*, 2nd ed. Rws Pubns, 2001.
- [92] K. P. Yoon and C.-L. Hwang, *Multiple Attribute Decision Making: An Introduction*. SAGE, 1995.
- [93] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of The Art Surveys*, vol. 78. Springer Verlag, 2005.
- [94] J. P. Brans and P. Vincke, "A Preference Ranking Organisation Method:(the PROMETHEE Method for Multiple Criteria Decision-Making)," *Management science*, pp. 647–656, 1985.



- [95] S. Brandt, *Data Analysis: Statistical and Computational Methods for Scientists and Engineers*, 3rd ed. Springer, 1998.
- [96] C. Kahraman, *Fuzzy Multi-Criteria Decision Making: Theory and Applications with Recent Developments*. Springer, 2008.
- [97] T. J. Ross, *Fuzzy Logic with Engineering Applications*, 2nd ed. Wiley, 2004.
- [98] R. M. Axelrod, *The Structure of Decision: The Cognitive Maps of Political Elites; Written Under the Auspices of the Institute of International Studies, University of California (Berkeley) and the Institute of Public Policy Studies, the University of Michigan*. Books on Demand.
- [99] S. Mizuno, Y. Akao, and K. Ishihara, *QFD, the Customer-Driven Approach to Quality Planning and Deployment*. Asian Productivity Organization, 1994.
- [100] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, 1st ed. The MIT Press, 2009.
- [101] J. Pearl, *Causality: Models, Reasoning, and Inference*, vol. 47. Cambridge Univ Press, 2000.
- [102] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [103] J. Pearl, "Decision Making Under Uncertainty," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 89–92, 1996.
- [104] D. Heckerman and others, "A Tutorial on Learning with Bayesian Networks," *Nato Asi Series D Behavioural And Social Sciences*, vol. 89, pp. 301–354, 1998.
- [105] R. Kindermann, J. L. Snell, and A. M. Society, *Markov Random Fields and Their Applications*. American Mathematical Society Providence, RI, 1980.
- [106] J. Pearl, "From Conditional Oughts to Qualitative Decision Theory," in *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*, 1993, pp. 12–20.
- [107] Y. Weiss and W. T. Freeman, "On The Optimality of Solutions of the Max-Product Belief-Propagation Algorithm in Arbitrary Graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 736–744, Feb. 2001.
- [108] S. P. Meyn, R. L. Tweedie, and P. W. Glynn, *Markov Chains and Stochastic Stability*, vol. 2. Cambridge University Press Cambridge, 2009.
- [109] J. D. Sterman, "System Dynamics Modeling: Tools for Learning in a Complex World," *California management review*, vol. 43, no. 4, p. 8, 2001.

- [110] E. O. Doebelin, *System Dynamics: Modeling, Analysis, Simulation, Design*. CRC, 1998.
- [111] J. Sterman and J. D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World with CD-ROM*. McGraw-Hill/Irwin, 2000.
- [112] “What is system dynamics?” [Online]. Available: [http://www.systemdynamics.org/what\\_is\\_system\\_dynamics.html](http://www.systemdynamics.org/what_is_system_dynamics.html). [Accessed: 16-Aug-2012].
- [113] M. J. Radzicki and R. A. Taylor, *U.S. Department of Energy’s Introduction to System Dynamics: A Systems Approach to Understanding Complex Policy Issues*, 1.0 ed. U.S. Department of Energy, 1997.
- [114] R. Axelrod, *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, 1997.
- [115] C. M. Macal and M. J. North, “Tutorial on Agent-Based Modelling and Simulation,” *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010.
- [116] C. M. Macal and M. J. North, “Tutorial on Agent-Based Modeling and Simulation PART 2: How to Model with Agents,” in *Simulation Conference, 2006. WSC 06. Proceedings of the Winter, 2006*, pp. 73–83.
- [117] M. Gries, “Methods for Evaluating and Covering the Design Space during Early Design Development,” *Integration, the VLSI Journal*, vol. 38, no. 2, pp. 131–183, 2004.
- [118] P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design: Modeling and Computation*. Cambridge Univ Pr, 2000.
- [119] F. Kockler, “Systems Engineering Management Guide,” DTIC Document, 1990.
- [120] O. De Weck, “Multiobjective Optimization: History and Promise,” *The Third China-Japan-Korea Joint Symposium on Optimization of Structural and Mechanical Systems*, Oct. 2004.
- [121] R. T. Marler and J. S. Arora, “Survey of Multi-Objective Optimization Methods for Engineering,” *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.
- [122] S. M. Lee, *Goal Programming for Decision Analysis*, 1st ed. Auerbach Publishers, 1972.
- [123] M. Schniederjans, *Goal Programming: Methodology and Applications*, 1st ed. Springer, 1995.
- [124] D. Jones and M. Tamiz, *Practical Goal Programming*. Springer, 2010.

- [125] A. Abraham and L. Jain, “Evolutionary Multiobjective Optimization,” *Evolutionary Multiobjective Optimization*, pp. 1–6, 2005.
- [126] C. Vira and Y. Y. Haimes, *Multiobjective Decision Making: Theory and Methodology*. North-Holland, 1983.
- [127] R. Benayoun, J. De Montgolfier, J. Tergny, and O. Laritchev, “Linear Programming with Multiple Objective Functions: Step Method (STEM),” *Mathematical programming*, vol. 1, no. 1, pp. 366–375, 1971.
- [128] J. T. Buchanan, “A Naive Approach for Solving MCDM Problems: The GUESS Method,” *Journal of the Operational Research Society*, vol. 48, no. 2, pp. 202–206, 1997.
- [129] K. Miettinen and M. M. Mäkelä, “Interactive Bundle-Based Method for Nondifferentiable Multiobjective Optimization: NIMBUS,” *Optimization*, vol. 34, no. 3, pp. 231–246, 1995.
- [130] A. Wierzbicki, “The Use of Reference Objectives in Multiobjective Optimization-Theoretical Implications and Practical Experiences,” *Int. Inst. Applied System Analysis, Laxenburg, Austria, Working Paper WP-79-66*, 1979.
- [131] A. Jaskiewicz and R. Słowiński, “The ‘Light Beam Search’ approach—an overview of methodology applications,” *European Journal of Operational Research*, vol. 113, no. 2, pp. 300–314, 1999.
- [132] L. M. Rios and N. V. Sahinidis, “Derivative-Free Optimization: A Review of Algorithms and Comparison of Software Implementations,” *Advances in Optimization II* (AIChE 2009), 2009.
- [133] S. P. Bradley, A. C. Hax, and T. L. Magnanti, *Applied Mathematical Programming*. Addison-Wesley, 1977.
- [134] T. G. W. Epperly, *Global Optimization of Nonconvex Nonlinear Programs Using Parallel Branch and Bound*. 1995.
- [135] P. N. Koch, J. P. Evans, and D. Powell, “Interdigitation for Effective Design Space Exploration Using iSIGHT,” *Structural and Multidisciplinary Optimization*, vol. 23, no. 2, pp. 111–126, 2002.
- [136] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2009.
- [137] Wikipedia contributors, “Hill Climbing,” *Wikipedia, the free encyclopedia*. Wikimedia Foundation, Inc., 05-Aug-2012.

- [138] “Artificial Intelligence/Search/Iterative Improvement/Hill Climbing - Wikibooks, open books for an open world.” [Online]. Available: [http://en.wikibooks.org/wiki/Artificial\\_Intelligence/Search/Iterative\\_Improvement/Hill\\_Climbing](http://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Iterative_Improvement/Hill_Climbing). [Accessed: 09-Aug-2012].
- [139] M. Harman, S. A. Mansouri, and Y. Zhang, “Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications,” *Department of Computer Science, King’s College London, Tech. Rep. TR-09-03*, 2009.
- [140] B. S. Mitchell and S. Mancoridis, “Using Heuristic Search Techniques To Extract Design Abstractions From Source Code,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, USA, 2002, pp. 1375–1382.
- [141] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by Simulated Annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [142] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, “Equation of State Calculations by Fast Computing Machines,” *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.
- [143] N. Tracey, J. Clark, and K. Mander, “Automated Program Flaw Finding Using Simulated Annealing,” in *Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis*, New York, NY, USA, 1998, pp. 73–81.
- [144] S. Bouktif, H. Sahraoui, and G. Antoniol, “Simulated Annealing for Improving Software Quality Prediction,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, New York, NY, USA, 2006, pp. 1893–1900.
- [145] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis, “Search Based Approaches to Component Selection and Prioritization for the Next Release Problem,” in *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, Washington, DC, USA, 2006, pp. 176–185.
- [146] F. Glover, “Tabu Search—Part I,” *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, Jun. 1989.
- [147] F. Glover, “Tabu Search—Part II,” *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, Dec. 1990.
- [148] F. Glover, “Tabu Search: A Tutorial,” *Interfaces*, vol. 20, no. 4, pp. 74–94, Jul. 1990.

- [149] E. Díaz, J. Tuya, R. Blanco, and J. Javier Dolado, “A Tabu Search Algorithm for Structural Software Testing,” *Computers & Operations Research*, vol. 35, no. 10, pp. 3052–3072, Oct. 2008.
- [150] E. Diaz, J. Tuya, and R. Blanco, “Automated Software Testing Using a Metaheuristic Technique Based on Tabu Search,” in *in: Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE’03)*, 2003, pp. 310–313.
- [151] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [152] H.-P. Schwefel and T. Bäck, *Artificial Evolution: how and why?* 1997.
- [153] H. G. Beyer, *The theory of evolution strategies*. Springer-Verlag New York Inc, 2001.
- [154] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, 1st ed. A Bradford Book, 1992.
- [155] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2010.
- [156] J. J. Dolado, “A Validation of the Component-Based Method for Software Size Estimation,” *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 1006–1021, Oct. 2000.
- [157] J. . Dolado, “On The Problem of the Software Cost Function,” *Information and Software Technology*, vol. 43, no. 1, pp. 61–72, Jan. 2001.
- [158] S. Wappler, “Automatic Generation of Object-Oriented Unit Tests Using Genetic Programming,” *Universitätsbibliothek der Technischen Universität Berlin*, 2007.
- [159] S. Wappler and J. Wegener, “Evolutionary Unit Testing of Object-Oriented Software Using Strongly-Typed Genetic Programming,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1925–1932.
- [160] I. F. Sbalzarini, S. Müller, and P. Koumoutsakos, “Multiobjective Optimization Using Evolutionary Algorithms,” in *Proceedings of the Summer Program*, 2000, vol. 2000.
- [161] H. D. Jørgensen, “Interactive Process Models,” *Department of Information and Computer*, 2004.
- [162] S. Cohen and A. Soffer, “Scrutinizing UML and OPM Modeling Capabilities with Respect to Systems Engineering,” in *International Conference on Systems Engineering and Modeling, 2007. ICSEM '07*, 2007, pp. 93–101.

- [163] “Semantics Of A Foundational Subset For Executable UML Models (FUML), Version 1.0.” OMG, Feb-2011.
- [164] “Concrete Syntax For UML Action Language (Action Language For Foundational UML - ALF),” *Concrete Syntax For UML Action Language*. [Online]. Available: <http://www.omg.org/spec/ALF/>. [Accessed: 24-Aug-2012].
- [165] “Foundational UML Reference Implementation | ModelDriven.org.” [Online]. Available: <http://portal.modeldriven.org/project/foundationalUML>. [Accessed: 23-Jul-2012].
- [166] R. Wang and C. H. Dagli, “Executable system architecting using systems modeling language in conjunction with colored Petri nets in a model-driven systems development process,” *Systems Engineering*, vol. 14, no. 4, pp. 383–409, 2011.
- [167] I. Reinhartz-Berger and D. Dori, “OPM vs. UML—Experimenting with Comprehension and Construction of Web Application Models,” *Empirical Software Engineering*, vol. 10, no. 1, pp. 57–80, 2005.
- [168] M. Peleg and D. Dori, “The Model Multiplicity Problem: Experimenting With Real-Time Specification Methods,” *Software Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 742–759, 2000.
- [169] Y. Grobshtein and D. Dori, “Creating SysML Views from an OPM Model,” in *Model-Based Systems Engineering, 2009. MBSE’09. International Conference on*, 2009, pp. 36–45.
- [170] K. Jensen, “An Introduction to the Practical Use of Coloured Petri Nets,” *Lectures on Petri Nets II: Applications*, pp. 237–292, 1998.
- [171] L. Wells, “Performance Analysis Using CPN Tools,” in *Proceedings of the 1st international conference on Performance evaluation methodologies and tools*, 2006, p. 59.
- [172] L. Wells, “Performance Analysis Using Coloured Petri nets,” in *10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings*, 2002, pp. 217 – 221.
- [173] C. Lakos, “Object Oriented Modelling with Object Petri Nets,” *Concurrent object-oriented programming and petri nets*, pp. 1–37, 2001.
- [174] T. Miyamoto and S. Kumagai, “A Survey of Object-Oriented Petri Nets and Analysis Methods,” *IEICE TRANSACTIONS ON FUNDAMENTALS OF ELECTRONICS COMMUNICATIONS AND COMPUTER SCIENCES E SERIES A*, vol. 88, no. 11, p. 2964, 2005.

- [175] R. Kocef and V. Janousek, "System Design with Object Oriented Petri Nets Formalism," in *Software Engineering Advances, 2008. ICSEA'08. The Third International Conference on*, pp. 421–426.
- [176] J. Saldhana and S. M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," in *International Conference on Software Engineering and Knowledge Engineering*, 2000.
- [177] B. Farwer and I. Lomazova, "A Systematic Approach towards Object-Based Petri Net Formalisms," in *Perspectives of System Informatics*, 2001, pp. 255–267.
- [178] C. Lakos, "From coloured Petri nets to object Petri nets," *Application and Theory of Petri Nets 1995*, pp. 278–297, 1995.
- [179] C. Maier and D. Moldt, "Object Coloured Petri Nets-A Formal Technique for Object Oriented Modelling," *Concurrent object-oriented programming and petri nets*, pp. 406–427, 2001.
- [180] M. Peleg and D. Dori, "Extending the Object-Process Methodology to Handle Real-Time Systems," *ournal of Object-Oriented Programming (JOOP)*, vol. 11, no. 8, pp. 53–58, 1999.
- [181] L. Zhou, E. A. Rundensteiner, and K. G. Shin, "Schema Evolution of an Object-Oriented Real-Time Database System for Manufacturing Automation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 6, pp. 956–977, Dec. 1997.
- [182] DStudio, "Opcat Systems." [Online]. Available: <http://www.opcat.com/>. [Accessed: 25-Dec-2011].
- [183] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Nov. 1990.
- [184] P. Trinidad, "Abductive Reasoning and Automated Analysis of Feature Models: How are they connected?" 2009.
- [185] "Arc inscriptions << CPN Tools Homepage." [Online]. Available: [http://cpntools.org/documentation/concepts/colors/inscriptions/arc\\_inscriptions](http://cpntools.org/documentation/concepts/colors/inscriptions/arc_inscriptions). [Accessed: 27-Aug-2012].
- [186] K. Jensen, "An Introduction to the Theoretical Aspects of Coloured Petri Nets," *A decade of Concurrency Reflections and Perspectives*, pp. 230–272, 1994.
- [187] "python-snakes - SNAKES is the net algebra kit for editors and simulators," *python-snakes*. [Online]. Available: <http://code.google.com/p/python-snakes/>. [Accessed: 27-Aug-2012].

- [188] “Guards << CPN Tools Homepage.” [Online]. Available: <http://cpntools.org/documentation/concepts/colors/inscriptions/guards>. [Accessed: 27-Aug-2012].
- [189] “Code segments << CPN Tools Homepage.” [Online]. Available: [http://cpntools.org/documentation/concepts/colors/inscriptions/code\\_segments](http://cpntools.org/documentation/concepts/colors/inscriptions/code_segments). [Accessed: 27-Aug-2012].
- [190] D. Dori, R. Feldman, and A. Sturm, “From conceptual models to schemata: An object-process-based data warehouse construction method,” *Information Systems*, vol. 33, pp. 567–593, Sep. 2008.
- [191] “Petri Nets Tool Database.” [Online]. Available: <http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html>. [Accessed: 18-Aug-2012].
- [192] E. Brinksma, H. Hermanns, and J.-P. Katoen, *Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science Berg en Dal, The Netherlands, July 3-7, 2000. Revised Lectures*. Springer, 2001.
- [193] A. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed. McGraw-Hill Science/Engineering/Math, 1999.
- [194] K. Jensen, L. M. Kristensen, and L. Wells, “Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems,” *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3–4, pp. 213–254, Mar. 2007.
- [195] A. Iglesias, “Pure Petri Nets for Software Verification and Validation of Semantic Web Services in Graphical Worlds,” *International Journal of Future Generation Communication and Networking*, vol. 3, no. 1, pp. 33–46, 2010.
- [196] L. W. Wagenhals, S. Haider, and A. H. Levis, “Synthesizing Executable Models of Object Oriented Architectures,” in *Proceedings of the conference on Application and theory of petri nets: formal methods in software engineering and defence systems-Volume 12*, 2002, pp. 85–93.
- [197] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1*, 2nd ed. Springer, 2003.
- [198] P. Buchholz, J. P. Katoen, P. Kemper, and C. Tepper, “Model-Checking Large Structured Markov Chains,” *Journal of Logic and Algebraic Programming*, vol. 56, no. 1, pp. 69–97, 2003.
- [199] S. Naguleswaran and L. B. White, “Planning Without State Space Explosion: Petri Net to Markov Decision Process,” *International Transactions in Operational Research*, vol. 16, no. 2, pp. 243–255, 2009.



- [200] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, “FAMA: Tooling a Framework for the Automated Analysis of Feature Models,” in *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, 2007, pp. 129–134.
- [201] “inspyred: Bio-inspired Algorithms in Python — inspyred 1.0 documentation.” [Online]. Available: <http://inspyred.github.com/>. [Accessed: 27-Aug-2012].
- [202] “Timed Nets << CPN Tools Homepage.” [Online]. Available: <http://cpntools.org/documentation/concepts/time/start>. [Accessed: 31-Jul-2012].
- [203] “Franck Pommereau: Nets in nets with SNAKES.” [Online]. Available: <http://pommereau.blogspot.com/2010/01/nets-in-nets-with-snakes.html>. [Accessed: 03-Sep-2012].
- [204] “Resources | Graphviz - Graph Visualization Software.” [Online]. Available: <http://www.graphviz.org/Resources.php>. [Accessed: 03-Sep-2012].
- [205] S.-Y. Son, “Design Principles and Methodologies for Reconfigurable Machining Systems,” Ph.D. Dissertation, 2000.
- [206] L. Tang, “Design and Reconfiguration of RMS for Part Family,” University of Michigan, 2006.
- [207] V. Maier-Sperdelozzi, S. J. Hu, and others, “Selecting Manufacturing System Configurations Based on Performance Using AHP,” *Technical Paper Society of Manufacturing Engineers MS*, no. MS02–179, pp. 1–8, 2002.
- [208] A. Youssef and H. ElMaraghy, “A New Approach for RMS Configuration Selection,” in *Proceedings of the CIRP 3rd International Conference on Reconfigurable Manufacturing*, 2005, pp. 10–12.
- [209] D. Batory, D. Benavides, and A. Ruiz-Cortés, “Automated Analysis of Feature Models: Challenges Ahead,” *Communications of the ACM*, vol. 49, no. 12, pp. 45–47, 2006.
- [210] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, “Automated Reasoning on Feature Models,” in *Advanced Information Systems Engineering*, 2005, pp. 381–390.
- [211] D. Batory, “Feature models, grammars, and propositional formulas,” *Software Product Lines*, pp. 7–20, 2005.
- [212] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, “Using Constraint Programming to Reason on Feature Models,” in *IN THE SEVENTEENTH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING*, 2005.

- [213] D. Streitferdt, M. Riebisch, and K. Philippow, “Details of Formalized Relations in Feature Models Using OCL,” in *Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the, 2003*, pp. 297–304.
- [214] A. Kimms, “Minimal Investment Budgets for Flow Line Configuration,” *Iie Transactions*, vol. 32, no. 4, pp. 287–298, 2000.
- [215] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan, “A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II,” in *Parallel Problem Solving from Nature PPSN VI, 2000*, pp. 849–858.
- [216] “Manned Lunar Landing Program Mode Comparison,” NASA, NASA-TM-X-66764, Jul. 1962.
- [217] “Manned Lunar Landing Mode Comparison,” NASA, NASA-TM-X-66763.
- [218] “Apollo 11 Mission Reports,” NASA, Houston, Texas, MSC-00171, Nov. 1969.
- [219] E. Frazzoli, M. A. Dahleh, and E. Feron, “Robust Hybrid Control for Autonomous Vehicle Motion Planning,” in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on, 2000*, vol. 1, pp. 821–826.
- [220] D. M. Reeves, M. D. Scher, A. W. Wilhite, and D. O. Stanley, “The Apollo Lunar Orbit Rendezvous Architecture Decision Revisited,” 2005.
- [221] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation, Second Edition*, 2nd ed. Academic Press, 2000.
- [222] N. A. Gershenfeld, *The Nature of Mathematical Modeling*. Cambridge University Press, 1999.
- [223] E. F. Crawley, O. L. de Weck, S. D. Eppinger, C. L. Magee, J. Moses, W. Seering, J. Schindall, D. Wallace, and D. Whitney, “The Influence of Architecture in Engineering Systems,” 2004.

## VITA

Renzhong Wang received his B.E. with Honors in Automobile and Tractor Engineering from the Jilin University of Technology, China in July 1998 and the M.S. degree in Systems Engineering from the University of Missouri-Rolla (now Missouri University of Science and Technology) in 2007. Between earning these two degrees, he had over six years work experience in the automotive industry, including working in the field of automobile testing and technical standards at the China Automotive Technology & Research Center (CATARC) and assisting in the administration of the domestic automotive industry at the State Administration of Machinery Industry, the State Economic & Trade Commission, and the National Development and Reform Commission, consecutively. He received his Doctor of Philosophy Degree in Systems Engineering from the Missouri University of Science and Technology, Rolla, Missouri, USA in 2012.

Renzhong Wang has been a member of the International Council on Systems Engineering (INCOSE), the Institute of Electrical and Electronics Engineers (IEEE), and the Society of Automobile Engineers (SAE International), as well as the Tau Beta Pi Honor Societies. His research on the optimum system architecture development using computational intelligence won the INCOSE Foundation/Stevens Institute Doctoral Award for Promising Research in Systems Engineering and Integration for 2008. His research interest includes system modeling and analysis, executable system architecting, data mining and knowledge discovery, and computational intelligence. He is currently employed as a research engineer and is working on developing cyber security solutions.