ABSTRACT

Title of Document: ROVER: ARCHITECTURAL SUPPORT FOR EXPOSING AND USING CONTEXT

Christian Butiu Almazan, Doctor of Philosophy, 2010

Directed By: Professor Ashok K. Agrawala, Department of Computer Science

Technology has advanced to the point where many people feel it has created a world with an insurmountable amount of information. Information includes messages people send to each other, logged data from their activities, and the services available to them. This problem has been exaggerated in modern societies by high availability of Internet connectivity. All types of information contains context, whether they have been stated explicitly or understood implicitly. Understanding, handling, and using context represents one of the most critical steps towards coping with the amount of information available today.

In this dissertation, we examine two topics: context and the design of a context-aware platform. We describe fundamental types of context associated with every piece of information and discuss issues which may occur when implementing a system which utilizes context.

We present a context-aware platform called Rover. The Rover architecture provides a conceptual framework geared towards understanding how application

developers can utilize a variety of aspects of context to assist the development of modern applications. To aid developers in figuring out what context may be useful in their application, we describe the concept of a Rover ecosystem: a logical organization analogous to how similar groups of people interact with each other. We also discuss how information and context can be shared between ecosystems.

To examine the feasibility of the Rover architecture's conceptual framework, we have implemented a reference implementation of the core unit of a Rover ecosystem: the Rover server. We discuss the details of the Rover server and describe the implementation of an emergency response application which demonstrates the utility of the conceptual framework.

ROVER: ARCHITECTURAL SUPPORT FOR EXPOSING AND USING CONTEXT

By

Christian Butiu Almazan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirement for the degree of
Doctor of Philosophy
2010

Dedication

To those people who touched my life and have moved on.

Acknowledgements

During my time in graduate school, I have been extremely fortunate to collaborate and meet people with enormous talents and heart. I will not forget the experiences we have shared throughout the years.

First and foremost, I thank my advisor Dr. Ashok Agrawala, who has been much more than just an advisor to me. He challenges everyone who meets him and makes sure their eyes open up to everything around them. To computer science professors Dr. James Reggia, Dr. A. Udaya Shankar, and Dr. Amitabh Varshney: thank you for serving on my dissertation committee. I truly value your suggestions and thoughts, all of which make my work stronger. Thank you Dr. Saúl Sosnowski for serving as dean's representative for my examination. Your perspectives from outside computer science have allowed me to see my work in a different light.

I cannot possibly thank enough all of the undergraduate and graduate students at the University of Maryland who I have had the privilege to work with me in one way or another. Thank you to Vandita Arora, Arun Balasubramanian, Aniket Dutta, Vinay Gangadhar, Rachana Gandi, Neha Gupta, Gleneesha Johnson, Morgan Kleene, Shivsubramani Krishnamoorthy, Saurabh Kulkarni, Thomas Krug, Dr. Lorin Hochstein, Dr. Cristian Lumezanu, Yi-Jung Lo, Elvita Lobo, Carrie Long, Matthew Mah, Jaswanthi Meganathan, Ankur Oberoi, Mohana Pramod, Nick Rutar, Raquel Sosnowski, Bao Trinh, Dr. Arunchandar Vasan, and Dr. Moustafa Youssef. To all of the students in who have ever taken Dr. Agrawala's class on the Information-Centric Design of Computer Systems: thank you for listening and providing feedback from my presentations. To all the people not mentioned here, know I have not forgotten you.

Special thanks to Dick Baer, Dr. Chuck Rieger, Bill Strum who have provided me invaluable advice throughout my graduate career. Thank you to Brenda Chick and Elizabeth Warner for all of the support they have provided for Maryland Day demonstrations of the work described in this dissertation.

My friends throughout the years, Amanda Blayne, Warren Chin, Amira Dahan, Kate Fuller, Kara Rothmann, Lauren Salvatore, Robert Schatz, and Emily Zima have all provided me with encouragement and I thank them for it.

To my parents, Angel and Nenita, and my sister, Annette: thank you for believing in me and supporting me throughout my entire life. To my in-laws, Paul and Jamie Krug, thank you for believing in me.

I leave my final thanks to my wonderful and extremely patient wife, Amelia. Thank you for your infinite support and your complete faith in me. Your support has made me a stronger person in all aspects of life.

Table of Contents

Dedication ii
Acknowledgements iii
Table of Contents v
List of Tables vii
List of Figures viii
Chapter 1 Introduction 1
1.1 Evolving to the Information Age 1
1.2 Regressing to the Attention Age 3
1.3 Towards a Context-Aware Age 4
1.4 Relevant Issues Outside of the Scope of this Dissertation 6
Chapter 2 Context 9
2.1 An Aside on Information 9
2.2 Defining Context 14
2.2.1 Dictionary-based Definitions 14
2.2.2 Linguistic Domain Influences 15
2.2.3 Common Computer Science Terminology 18
2.2.4 Context-Aware Definitions 19
2.3 Our Definition of Context 22
2.4 Summary 22
Chapter 3 Aspects of Context 23
3.1 Considerations Concerning the Use of Context 23
3.1.1 Interpretation Issues 24
3.1.2 Cultural Issues 27
3.1.3 Abstraction Issues 28
3.1.4 Accuracy, Resolution, and Latency 30
3.2 Critical Aspects of Context 31
3.2.1 Identity 31
3.2.2 Time 32
3.2.3 Location 34
3.3 Categorizing Context 36
3.4 Active Context 37
3.5 Summary 39
Chapter 4 Context-Aware Computing 40
4.1 Goals and Types of Context-Aware Computing 40
4.2 Context-Aware Computing 43
4.3 Context Models 47
4.4 Standards for Representing Context 49
4.5 A Context-Aware System Examination 52
4.6 Summary 54
Chapter 5 The Design of a Context-Aware Architecture 55
5.1 Basic Requirements of a Context-Aware System 55
5.2 Motivating Scenarios 57
5.2.1 Context-Aware Services for Everyday Needs 57
5.2.2 Using Context to Facilitate Emergency Situation Response 59

5.3 Rover Ecosystems ... 61
5.3.1 Server Tier ... 63
5.3.2 Utility Tier ... 64
5.3.3 Assistance Tier ... 66
5.3.4 User Tier ... 67
5.4 Context Representation ... 68
5.4.1 Context Data Structures ... 68
5.4.2 Context Wire Representations ... 70
5.5 Application Programming Interface ... 71
5.5.1 Context API ... 71
5.5.2 Service API ... 74
5.5.3 Message API ... 76
5.5.4 History API ... 76
5.6 Presence and Context Standards in Rover ... 78
5.7 Summary ... 79
Chapter 6 The Implementation of a Context-Aware Ecosystem ... 81
6.1 Desired Attributes of a Rover Server Implementation ... 81
6.2 Prior Rover Server Implementations ... 82
6.3 Rover Server Implementation Details ... 84
6.3.1 Development Environment ... 84
6.3.2 Remote Procedure Call Mechanism ... 85
6.3.3 Mobility Support ... 86
6.3.4 Authentication ... 87
6.3.5 Database Support ... 90
6.3.6 Rover API ... 94
6.4 Example Application Scenario: V911 ... 94
6.4.1 Ecosystems and Entities in an V911 Emergency Response Scenario ... 95
6.4.2 Fire Event Scenario ... 100
6.4.3 Discussion ... 104
6.5 Summary ... 105
Chapter 7 Concluding Remarks ... 106
7.1 Future Directions ... 107
Appendix A Rover Server Reference Implementation Function List ... 109
A.1 Session API ... 111
A.2 Session API ... 116
A.3 Context API ... 117
A.4 Service API ... 121
A.5 Message API ... 123
A.6 History API ... 125
References ... 127

List of Tables

Table 1: Supporting context operations on the Rover server, including supplying, subscribing, and querying. 71

Table 2: Rover function calls allowing discovery and calling of services. 74

Table 3: Functions for communicating to other entities in an ecosystem via the Rover server. 76

Table 4: Querying functions for historical data on the Rover server. 76

Table 5: An evaluation of the Rover architecture with respect to previous context-aware framework discussions. 79

List of Figures

Figure 1: The traditional noisy channel diagram as described in Information Theory..... 10

Figure 2: The noisy channel model augmented with meaning. 12

Figure 3: An abridged version of McComb's model of interpretation. 25

Figure 4: Venn diagram showing two distinct Rover ecosystems with some overlap. 61

Figure 5: The four-tier entity structure in a Rover ecosystem, showing the logical design of how information flows between entities.. 62

Figure 6: The main operations of the Rover server: a message handler determines the destination of all messages sent to the Rover server... 63

Figure 7: Example workflow of expanding latitude and longitude context to a building number. 72

Figure 8: The lifetime of a *CallService* RPC call. 75

Figure 9: Illustration of the components used to support mobility................................. 86

Figure 10: The Rover server authentication mechanism. This shows how the three components interact with each other in two different phases. 88

Figure 11: Database diagram of how the context model has been represented in a relational database... 91

Figure 12: Context provenance database structure. 92

Figure 13: Call history database structure. 93

Figure 14: Message database schema. 93

Figure 15: Example campus services, including dining services, weather information, and real-time transportation updates... 95

Figure 16: An overview of the two Rover ecosystems and the entities which reside within them.. 96

Figure 17: Dispatcher Console screenshot with both Rover ecosystems incorporated into the display. 96

Figure 18: Annotated Dispatch Console screenshot. 98

Figure 19: V911 Sequence Diagram of Events... 101

Chapter 1

Introduction

Historians group various ages of human evolution together. An examination of these ages yield an interesting correlation: as humans go from one age to the next, the length of time it takes to get to the next age decreases substantially as the time it takes for humans to interact with one another decreases [59]. The interaction time decreases from building upon technology, specifically communication and transportation, from previous ages.

1.1 Evolving to the Information Age

Our current age, typically referred to as the Information Age, has evolved primarily due to the development of the Internet [58]. We have evolved from a time where it took days, weeks, or even months to send a message to another person to a world in which a message can be delivered almost instantaneously to any place in the world.

Over the last decade, we have moved past exclusively using desktops and workstations to increased usage and acceptance of mobile devices: cell phones, smart phones, and Internet appliances. One of the reasons we have been able to move from tethered to untethered devices has been advances in mobile connectivity [16]. This includes advances in areas such as wireless cellular networks (from 1G to 4G) [69], wireless local area networks (802.11 [69] and ZigBee [120]), and wireless personal area networks (Bluetooth [14]). The increase of mobile connectivity means messages can be transmitted and viewed quicker than it ever has been before.

In addition to developed countries which have technology readily available to them, philanthropic efforts have begun to connect impoverished countries and areas with low bandwidth. The *One Laptop per Child* project exemplifies one of the highest profile efforts to provide children in developing countries with laptops and connectivity [55].

The Information Age has reached a point in which modern societies surround themselves with technology. The late Mark Weiser envisioned a world in which computers would fade into the background, where we would take advantage of their capabilities seamlessly and without thought [114]. He coined the term Ubiquitous Computing to describe this paradigm, a shift from the computing world consisting primarily of desktop computers [115].

While many people carry laptops and mobile devices around with them and have Internet appliances in their homes and workplaces, computers have not faded completely into the background. After almost two decades, embedded computing devices have not been placed into our clothing and we still build homes traditionally, albeit with extra wiring and considerations for wireless signals. We still carry around separate devices, such as an audio player and a cellular phone, though devices exist which combine multiple features together.

Even though more computing appliances will become available and research continues in the field of Ubiquitous Computing, we need to recognize that the effective use of the information marks one, if not the most, important critical aspect of computers fading into the background. For now, let us assume information includes only the messages that every computing device provides, and simply that.

1.2 Regressing to the Attention Age

A study conducted by researchers at the University of California at Berkeley's School of Information Management and Systems attempted to answer the following question: "How much new information is created each year?" [63]. The researchers observed the following two facts concerning new information:

1. Traditional physical storage mediums, which include print, film, magnetic, and optical, produced five exabytes of new information in 2002. This represented a growth of thirty percent from 1999.
2. Communication mediums, including radio, television, telephone, and the Internet, created 17.3 exabytes of information in 2002, mostly from landline and wireless phones.

This study only considered new information. Therefore, the amount of information available from historical archives, such as traditional libraries and Internet archives[1], has not been accounted for. Sifting through the amount of information available today, both old and new, can be a daunting and difficult task. Furthermore, the growing popularity of social networking applications [2] increases the amount of communication between participating individuals and groups.

We can take solace that one of the researchers involved in the Berkeley study indicated that most of the information available does not interest most people and applications [111]. However, we still need to be able to sift through all of the information given to find information relevant to us, while not knowing if the information available to us has all of the relevant information in the first place. For instance, after a

[1] One popular archival site which caches web sites includes the Internet Archive [52].
[2] Social networking applications include Facebook [30] and Twitter [108].

person comes back from vacation and on the telephone, they commonly say to others, “Let me clean up my e-mail and get back to you.” Reviewing e-mail requires a person to be attentive to figuring out what e-mails to prioritize, read, and mark as relevant, a time consuming process.

Due to the amount of information readily available, people have to deal with information overload [106]. The act of dealing with information overload requires people to be attentive to the information presented to them. The amount of attention spent handling all of the information increases as the information presented to people increases over time. As the Information Age continues, one could argue, on the basis that ages last very short periods of time today, that we have entered another age: the Attention Age [20]. We spend large periods of time attending to sifting through unnecessary information, without knowing if we have the appropriate information in the first place.

1.3 Towards a Context-Aware Age

As researchers and professionals continue working towards the goal of Mark Weiser’s vision of computers disappearing into the background, we have to recognize that we need to step out of the attention age into an age where technology provides a reliable way of providing the appropriate information to users. To do this, we believe that we have to consider *context*. Context plays an essential role in our ability to interpret and use information presented to us. While we as humans consider context all the time in all our activities, computer systems do not usually take that into account. In fact, a computer system only takes the context a designer thought about during the design process into account. As a consequence, a computer system’s ability to provide relevant information to us gets restricted and we end up having to sift through high volumes of information.

We believe that the next age is the Context-Aware Age in which machines will become context-aware. This dissertation is a study of the problem of making machines context-aware.

In this dissertation, we start by defining what *context* consists of, how *context* can be utilized to provide a useful mechanism, and building a computer system which effectively uses *context* to aid in the flow of information. We believe the effective use of *context* will bring us closer to answering the following two questions:

- How can we facilitate the sharing of relevant information between entities?
- What can be done to simplify the attention needed by a user?

To help answer these questions and start bridging the gap towards complete effective use of information, we have designed, developed, and tested an architectural platform we call Rover. This platform enables software architects to *understand* context from the onset of their design, and developers to *utilize* context effectively in their program development, all while giving developers the ability to integrate previously existing services, such as inference mechanisms and transformation mechanisms. The Rover platform has the following goals in mind:

- The ability to capture and store context sensed by the appropriate entities, including users, services, and sensors.
- Manage context and unfold additional context from existing context.
- Create generic primitives and building blocks that utilize context from all levels of interaction, where appropriate.
- Take advantage of existing services by augmenting their usefulness with the appropriate use of context.

- Allow organizations, which may have heterogeneous and homogeneous data, to exchange data and context with each other.

  This dissertation will proceed as follows:

- A discussion of how context has previously been defined and how we define the term (Chapter 2).
- Discussion concerning issues that arise from using context and the most critical parts of context: identity, time, and location (Chapter 3).
- A survey of research on systems, architectures, and applications which utilize context (Chapter 4).
- An overview of the design of the Rover architecture (Chapter 5).
- An evaluation of the implementation of the Rover server (Chapter 6).

1.4 Relevant Issues Outside of the Scope of this Dissertation

As this dissertation primarily focuses on the core building block of a context-driven architecture, we list several relevant issues here that fall outside of the scope of this dissertation. Rover can be used to address several of these issues, but we hope by listing them here, future studies can take place as appropriate. In fact, the issues listed here can be developed independently of the Rover architecture and integrated into a Rover system since it has an extensive set of interfaces. We list the issues here, cite a relevant source, and discuss their relevance to a Context-Driven Age:

- Usability Issues [99]

  Input methods and displays represent the primary way for users to interact with computing systems. By improving the way users interact will decrease the amount of attention required to handle information presented to them. In addition to just reducing the user's attention, the input mechanisms provide additional context regarding the user.

- Security Issues [5]

  Only basic issues of security in the Rover architecture, that of secure communication mechanisms and basic methods of authentication, have been incorporated in this dissertation. We realize that much work needs to be done to improve the overall areas of confidentiality, integrity, and availability.

- Privacy Concerns [80]

  Context consists of highly personal information about individuals and groups. There needs to be mechanisms in place such that only the appropriate context will be conveyed only to those who have the appropriate access.

- Dissonant Information [96] and Providing Misinformation through Deliberate Lying [22]

  Any context generated with malicious intent or context which provides a contradiction to existing context can be harmful to an entity or the overall state of a system. In this dissertation, we assume that any piece of context provided to a computer system does not contradict another piece of context and has no malicious intent.

- Methods of Reasoning [96]

  Pieces of context can typically lead to other context, such as knowing a building number on a campus can yield spatial coordinates. We provide mechanisms to expand context, but do not define how to reason about context. Our reference implementation contains relatively simple methods of expanding context. Also, we do not concern ourselves directly with matching related pieces of context [29].

Chapter 2

Context

Prior to discussing context-aware platforms, we have to first determine the significance of the term *context* with respect to context-aware computing. The term context has been extensively used as a concept for understanding words in linguistics and understanding the circumstances surrounding an event. Computer science has used the term context for specific circumstances, such as context switching in operating systems and context-free grammars in theory of computation, but only in the last two decades has a general definition of context attempted to be elaborated on, specifically for context-aware computing.

When thinking about how to define context for context-aware computing applications, one may begin to think about interrogative words one could ask concerning a piece of information[3], such as who did what and where did it originate. However, this does not give any insight as to how to define context formally. In fact, research related to anything concerning context almost always defines context with regard to the specific problem domain. We explore several different domains of where context has been defined and see how it may apply to our own definition of context.

2.1 An Aside on Information

Prior to discussing context in detail, we first need to pay particular attention to the term *information*, a term we alluded to in chapter 1. As we have used the term *information* in several places already, such as the discussion of the Information Age and then the term

[3] Interrogative words include, but not limited to: who, what, why, where, how, and when.

by itself, we believe it would be useful to distinguish *information* and possible interpretations of the term[4].

Physicists and philosophers, especially those studying the philosophy of sciences and quantum mechanisms, frequently divide *information* into two distinct categories: *everyday information* and *technical information* [113] [105]. We first examine *technical information*: the category which primarily deals with mathematics and engineering, especially in the measurement of the usefulness of bits in communications. This type of information can be traced back to the origins of Information Theory.

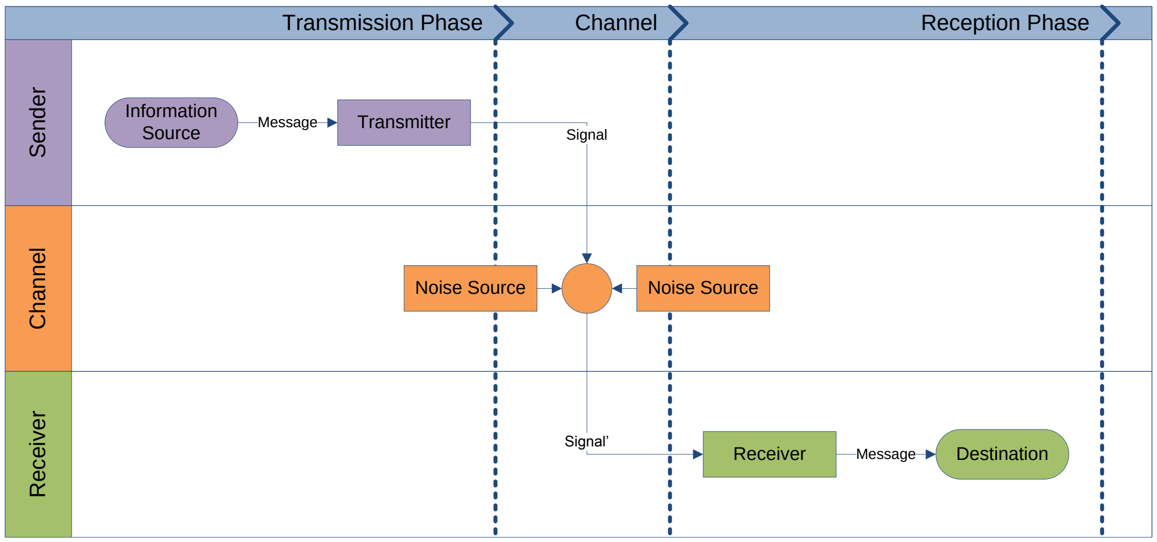


Figure 1: The traditional noisy channel diagram as described in Information Theory.

The father of Information Theory, Claude E. Shannon, considered the transmission of a message from one entity to another entity through a noisy channel [97]. Figure 1 illustrates the general flowchart of a noisy channel environment. In this environment, a *sender* wants to send a *message* to a *receiver*. The *message* has to be

[4] John Bell published a list of *bad words* in relation to physics, formulation, and quantum mechanics [11]. These words typically have different meanings and connotations depending on the situation the word has been used in. *Information* happens to fall on this list, hence our need to define *information* to our situation. We hope that the term *context* would not fall into a so-called list formulated for computer science, if one were to be created, as *context* has been continually redefined.

translated into an appropriate *signal* such that it can be transmitted by a *transmitter* over a particular *channel*. The *signal* represents a set of symbols which can be transmitted over a communication medium, the *channel*. In most scenarios, the *channel* may have *noise* introduced into it by any number of *noise sources*, thus creating *signal'*. The receiver receives s*ignal'* and understands how the *transmitter* transmits and the characteristics of the *channel*. With this understanding, the receiver can reconstruct the original *message*. Consider the following example:

> *Amber alert!! 3 yr old boy taken by man in Rochester MN driving 2006 Mitsubishi Eclipse.. Plate # 98B351.. Repost if you wish.. You know you would keep it going if it was someone you knew "Seriously!!! I expect to see this repeated on this page many times....a child is in danger* – A Facebook Status Message from 01/08/2010.

In this example, an individual wants to convey a *message* to his or her friends via a status message update on the Facebook social networking site[5]. The user types in a *message*, which will be translated into a *signal* to be sent over the Internet to the Facebook server. The Internet *channel* may be noisy. Therefore, the server then reconstructs the *message* from *signal'*. If the *message* sent by the *information source* can be reconstructed bit-by-bit and *received* by the *destination*, the Information Theory problem, at this point, has been addressed.

Unfortunately, the reception of a *message* does not mean the receiver will use the *message* the same way *sender* intended it to be used. In fact, these *messages* have *meaning*, but as Shannon points out, the engineering problem does not concern itself with the contents of the *message* [97]:

> *Frequently the messages have meaning; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem.*

[5] We realize that multiple communication channels will be utilized in a single Internet connection, but we abstract the channels into a single channel.

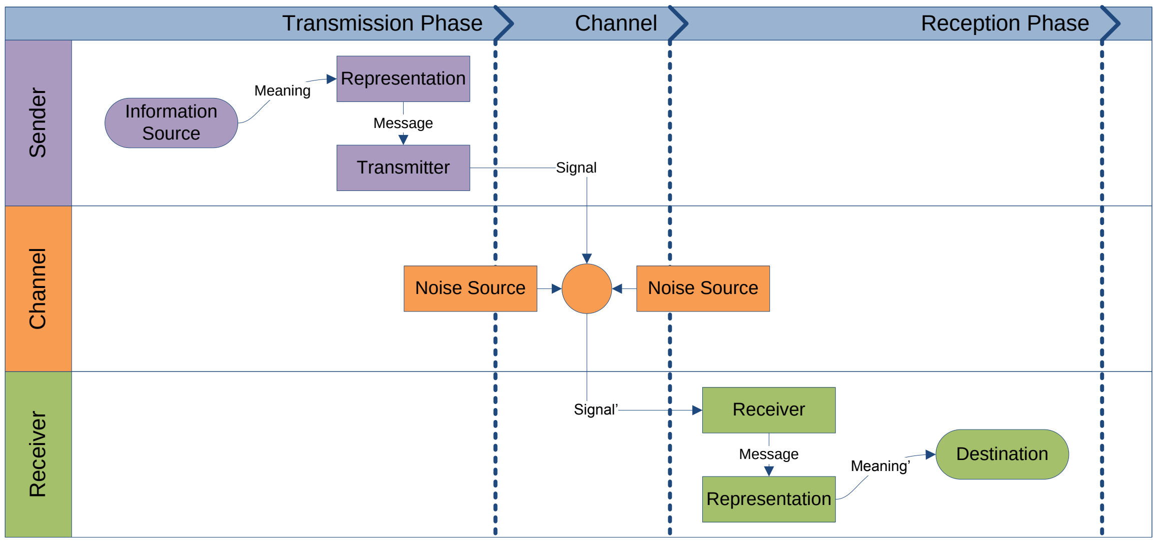


Figure 2: The noisy channel model augmented with meaning.

This brings us to *everyday information*. As von Baeyer [113] and Timpson [105] point out, *everyday information* concerns itself with the natural use of language and communication between people, typically involving understanding and meaning. These aspects have no relevance to Information Theory. Going back to the Facebook example, one can ask: does this *message* convey something current or even accurate? Given the state of how fast messages reach other people, we need to be able to understand the *messages* being passed around. Figure 2 augments the noisy channel model from Figure 1 by adding *meaning* to the diagram.

We should note that processing any information, either technical information or everyday information, has to be acquired, understood, analyzed, and either stored or utilized. These activities require processing power and time. With this in mind, we return to the Facebook example. The Facebook example's *sender* has good intentions: he wants to try to help a child in need. However, given the nature of the Internet, as the message has quickly reached both New York City, the place where *sender* resides, to

Maryland, the place where the *receiver* resides, we frequently want to verify the message contents and its relevance to the situation. To do this, we could look at the active Minnesota Active Alerts and, if we have access to it, Department of Motor Vehicles information regarding the car. At the time of the Facebook status message update, Minnesota had no active Amber Alerts [68] and, in fact, the alert did not come from a legitimate source as it never existed in the first place [72].

The assigning of a meaning to a message has to be carried out in a context and as the context changes, meaning may change also. For example, a person may ask another person, “How is the *rabbit* doing?” Depending on the context of the conversation the two people are having, the message, in particular the term *rabbit*, changes. In one situation, *rabbit* may mean an animal. *Rabbit* could be a microprocessor controller in another situation. One final situation to consider, *Rabbit* can refer to an automobile model made by Volkswagen.

In this dissertation, we primarily consider *everyday information*, but recognize that the problem contained within *technical information* needs to be solved before even considering *everyday information*. If we receive incorrect *messages* to begin with, *meaning* would almost always be incorrect. To this extent, when we say *information,* we mean what we call an *information item*: a *message* that has associated *context* with it. An *information item* should not be limited to a *message*, as occurrences, actions, and activities also have surrounding *context*. We spend the remaining part of this chapter explaining our definition of *context*.

2.2 Defining Context

Bazire and Brézillon state that “Although frequently used in cognitive sciences or other disciplines, context stays a very ill-defined concept” [10]. This poses an issue since it implies that no generic definition for context exists, assuming we discard definitions from their respective language dictionaries.

For our purposes, we begin defining what we mean by *context* with a review of a variety of definitions from a few fields, recognizing that it would be impossible to review all of the connotations of context as defined by others. Therefore, we have selected those that we consider relevant to our discussion.

2.2.1 Dictionary-based Definitions

The word *definition* typically invokes a thought of the word dictionary, as a dictionary contains simply that: definitions. The meaning and etymology of a word, in this case *context*, can give us insight as to what it may mean. The etymology of the word *context* comes from the Latin *contextus*, meaning “to weave together” [39]. This implies *context* embodies information, that is, the support for the information that exists in the first place.

Now, let us consider relevant traditional definitions from a dictionary. We present four definitions, the first two from Merriam-Webster [66] and the latter two from WordNet [33]:

1. “the part of a discourse that surround a word or passage and can throw light on its meaning”
2. “the interrelated conditions in which something exists or occurs : environment, setting”

3. “discourse that surrounds a language unit and helps to determine its interpretation”
4. “the set of facts or circumstances that surround a situation or event”

All of the definitions of context relate directly back to the etymology of the word: something which surrounds information. We recognize that context has traditionally meant the set of surrounding facts that set up a meaning or the reason an occurrence has happened. With that being said, we have to rule out the first and third definitions from the definitions presented when discussing about general *context*, as linguistics only represents one kind of information item. At this point, we define *context* to be everything which surrounds information.

2.2.2 Linguistic Domain Influences

Although we ruled out specific definitions which relate to linguistics, we may be able to use influences found in linguistics literature to discover a more generalized notion of context. From a high level, linguistics involves the act of communication through spoke and written language that generally involves two steps: interpretation and expression [65]. Interpretation means taking words and deriving concepts from them, while an entity expresses itself by taking a concept in its mind and conveys what it has in its mind in a physical form, such as speaking, writing, or gesturing. Although McComb states that “expression is far easier than interpretation,” we believe that it can be just as difficult to express. We frequently find ourselves saying “*how do I say this.*” This could stem from two reasons: trying to ensure that the receiver understands the same ideas we have or trying to find the appropriate words to express ourselves when we do not know the exact words we want to use.

Let us consider the activities we perform when we attempt to interpret new words. Sharing the ideas from the dictionary-based definitions that *surrounding clues* yield of what we call context, we first take a look at work done by Drum and Konopak on learning word meanings from *surrounding contextual clues* [26]. Every word encountered may have multiple definitions. Drum and Konopak assert context can aid in learning new meanings from existing knowledge [6] and finding additional meanings concerning a word. They identify four qualitative measures in order to attempt to identify whether or not an entity can identify a word they may not know. These qualitative measures include:

- Perpetual – similar sounding words.
- Syntactic – internal structure or grammatical function of words.
- Semantic – general meaning dimensions of a word.
- Correct – a specific definition.

These qualitative measures give entities which desire to express something the mechanisms to place words into a conversation that could aid an interpreter. Additionally, Drum and Konopak also state that linguistic hints can help identify words, such as the surrounding text, within the word itself, and the statement the word resides in.

In research performed by Sternberg, he acknowledges that individuals have varying sizes of vocabulary and he considers three *processes of knowledge acquisition* for trying to figure out the definition of a word [101]. These processes include:

- Selective Encoding – use relevant information only.
- Selective Combination – combining relevant cues.

[6] In this case, what an entity has already experienced.

- Selective Comparison – comparing new information with the old information one already knows.

Sternberg continues to describe the possible cues used to uncover the meanings of a word. These cues include:

- Temporal – frequency of a word in text.
- Spatial – the specific location of a word in text.
- Value – the value of a word.
- Static – properties of a word, typically adjectives, such as size, shape, color, odor, and feel, which surround a word.
- Functional Descriptive – the purpose or purposes of a word.
- Causal/Enablement – what causes this word or what conditions enable it?
- Class Membership – what group does the word belong to?
- Equivalence – synonyms and antonyms.

The cues stated by Sternberg have been defined in terms of a linguistic construct; however, we believe that each of these cues has a general context analogue. For example, every time the definition contains the term *word*, we replace it with *information*. In traditional information systems, *class membership*, refers to as a schema or ontology. Ontology systems frequently include inference mechanisms which define *equivalence*. Schemas and ontologies define *properties*, which represent *static* cues in Sternberg's research.

After examining the linguistic definitions, we now define *context* to be all of which surrounds information and provides evidence to support the information. We have also learned a few things about what to consider concerning *context*:

- Drum and Konopak only considered taking a look at new information. However, *context* concerns itself with both old and new information.
- When an entity expresses a thought, they can provide clues to aid the receiving entity to interpret it.
- *Context* may not be useful by itself, but may be combined with other *context*.

2.2.3 Common Computer Science Terminology

This dissertation concerns itself directly with the broad generalization of context as applied to computer systems. Therefore, it would be useful to state how the field of computer science has previously used the term *context*. We begin by examining a computing dictionary's definition of *context* [46]. The standard definition of *context* states "that which surrounds, and gives meaning to, something else," a definition which can be seen as similar to a traditional English language dictionary. On the other hand, *context* has also been defined as follows: "in a grammar it refers to the symbols before and after the symbol under construction. If the syntax of a symbol is independent of its context, the grammar is said to be context-free" [46]. Although this definition stems from the general field of the theory of computation, this still signifies that defining context depends on the surrounding situation.

We have an abundant number of computer science terms which contain the term *context*, including context-sensitive help, contextual search, and multitasking context switch [16]. If we take a look at a context switch in the pure computer science domain, we only consider operating systems and individual processes [47]. Processes have their own environmental context, which includes what step to execute and memory to access. For reasons such as security and memory management, processes have different environments. The environment, in this definition, should be considered a specific context, that which has been defined by the operating system. When the operating system decides to change execution from one process to another process, the operating system saves the state of the currently running process, its environment, its context, and the state of the process needed to be run. This series of actions represents a context switch.

Context-sensitive menus show users only relevant menus [48]. When a user right-clicks on the screen, the menu shown will typically be determined by the active window. The usage of the term *context* here means that we only show a menu relevant to what a user may be doing at the time.

Aside from the general definition of context in computing, the uses of the term *context* for each of the other specific use terms represent a very narrow focus that would not be relevant in a widely general definition of context relevant for this discussion.

2.2.4 Context-Aware Definitions

With the increase of mobile devices and sensors embedded in our everyday lives, context-aware computing has had a wide array of research concerning the development of these computers. However, almost every system and architecture in the literature redefines context to fit their research. Therefore, we explore recent, relevant definitions.

We begin our exploration of context-aware research definitions from what could be considered the first paper that introduced the term *context-aware computing*[7]. As location determination research began to come to the forefront of computer science research, Schilit and Theimer implicitly stated that context represents, location, spatially

[7] Dey [24] states Schilit and Theimer [90] coined the term context-aware computing first.

close people and objects, and the changes to people and objects [91]. According to Schilit and Theimer, context consists only of the location and nearby entities, that is, people and objects.

Schilit, Adams, and Want assert this to be true by saying context consists of three important aspects: where you are, who you are with, and what resources are nearby [90]. However, they go beyond the Schilit and Theimer definition by stating that *things of interest* move and change over time, where the *things of interest* could include "lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situations."

Location should not be the only context considered in a computing environment. Brown suggests that context, when attached to what he calls an electronic note, consists of anything associated with a note's environment [15]. This includes location, proximity of objects, critical states, computer states, imaginary states, and time. A generalization the definition of an environment has been stated by Pascoe. He states the idea of nearby environments to what he calls any physical or conceptual states relevant to an entity [78].

Some of the recent ideas of how context should be defined include the relationships of hypertext documents through linking [28] and constraints of a given task [10]. However, Coutaz, Crowley, Dobson, and Garlan state that we cannot possibly predefine the entire context possible since the environment changes frequently [18].

The most widely used definition of context today in context-aware computing research comes from Dey [24]. He first critiques other definitions of context in the context-aware domain by stating that they fall into one of three categories:

1. Dictionary Definitions – should not be considered sufficient or used directly.

2. Synonyms of Context – difficult to apply in practice.

3. Specific Definitions – limited scope, not widely applicable.

With this in mind, Dey suggests the following definition of context:

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

Although relevant for context-aware computing in general, this definition would not be sufficient for what Zimmermann, Lorenz, and Oppermann call an operational definition of context [121]. The authors state Dey provides only a formal definition, that is, without regard to the practice of using context in, what they call an operational environment. They continue to say that definitions of context should not only include formalisms, but also the operations useful in order to utilize context effectively. Zimmermann, Lorenz, and Oppermann extend Dey's definition by stating five categories of context (individuality, activity, location, time, and relations) and operations on context (methods and reasons of context transitions and methods of sharing context between entities).

Definitions of context-aware computing frequently build upon one another. For example, prior to defining context, Dey mentioned several previous definitions of context [24]. We can take one thing away from all of the definitions from the literature of context-aware computing: context changes frequently. Context needs to be considered a dynamic, not static, concept which can change for any number of reasons. These reasons include time passing, change of location, and so on.

2.3 Our Definition of Context

We have examined four different domains to explore how we should define context. By looking at domains both inside and outside of computing in general, we can keep our definition as generic as possible while incorporating other domain influences. We define our formal definition of context as follows:

> *Context consists of one or more relationships an information item has to another information item. An information item can be any entity, either physical (such as a person, a computer, an object), virtual (such as a computer service, a group of people, a message), or a concept (location, time, and so on). A relationship describes a predicate connecting two or more information items, which may change at any time for any reason.*

This definition stresses the term *relationship,* in particular, the *relationship* between two *information items*. These *relationships* could be practically anything that answers an interrogative question, such as "who is a friend with who" and "where is someone located?" Unlike Dey, we do not consider context only in the domain of context-aware computing, but computing in general.

2.4 Summary

In this chapter, we examined what we mean by *context*. Prior to examining context, we needed to understand the term *information*, a word with different connotations depending on who one talks to. We focus primarily on *everyday information*. As for *context*, we defined the term with respect to *information items* and the *relationships* each information item has with one another.

Chapter 3

Aspects of Context

In chapter 2 we presented the definitions of *context* and created our own definition of context that we use for our discussions herein this dissertation. Prior to discussing the development of our context-aware architecture, we have to understand two aspects of how we utilize context: *considerations concerning the use of context* and *core aspects*. Considerations concerning the use of context include any issues which may arise when utilizing context in an actual architecture. Core aspects represent relationships which context typically has associated with it.

3.1 Considerations Concerning the Use of Context

We need to pay attention to four caveats of using context which stem from the idea of misusing context. By misuse, we do not mean only malicious intent, but also accidental misuse also. Consider linguistic research done by Pressley, Levin, and McDaniel [84] in which they state that context can be misused. Their example involves learning new words or figuring out the exact meaning of words in a sentence or paragraph. If we interpret a word or series of surrounding words incorrectly, then it not only means we have the incorrect meaning of the current word or phrase, but also all future words and phrases will be derived from the incorrect meaning. In this section, we go beyond linguistic context and consider context in general. Clearly, misused context can lead to undesirable problems. Even though we do not discuss how to solve these problems, we list them so that we can pay attention to them.

3.1.1 Interpretation Issues

The lifetime of an entity, whether a computer system or a human, goes through life differently. Hence, each individual entity has unique series of experiences to draw upon. These experiences range from their interactions with other entities and all of the events that can possibly occur. Entities utilize these experiences to incorporate new information they receive, but since each entity has different experiences, their interpretations may be different, including being different from the original intent of the expresser. Augier, Syed, and Thanning indicate that if an entity needs to know something about a particular event, the questions they ask to ascertain additional context will be determined by an individual's prior experiences [4].

To understand how interpretation takes place, let us first reconsider the basic model of communication between two entities in the style of Information Theory [97], ignoring the fact that the engineering problem does not concern itself with meaning or semantics. To recap the traditional model, a sender intends to send a message to a receiver. The sender translates the message into an appropriate signal, a set of bits, in order to be moved over a communication medium, a channel. While the message moves over the channel, a noise source may introduce noise or alter the signal. When the receiver receives the signal, a translation mechanism reconstructs the message by analyzing the potentially altered signal. The receiver then receives the message derived from the received signal by the receiver.

In the traditional model of communication, the transmitter and the receiver have a well-defined model of communication. Both the transmitter and the receiver negotiate an exact alphabet to translate via signals. The transmitter and the receiver know certain characteristics of the noise source; therefore they can both transform the message to a signal, and vice versa, appropriately.

Our work not only depends on the ability to communicate messages correctly, but we have to take the engineering problem a step further. The messages have meaning and semantics. The experiences we utilize contain context and relationships. As we have already alluded to, the way we interpret messages depends on our previous experiences. Take the following quote as an example: *The messages I've tried to send, my information's just not going in*[8]. The statement clearly explains the situation: someone sends a series of messages, but it never has its intended effect. This may or may not reflect the Information Theory problem, as the receiver may not be able to reconstruct the original message, but let us assume the engineering problem has been solved. Therefore, the message does not go through to the other person due to him not having the appropriate model or the sender not sending the appropriate messages across.

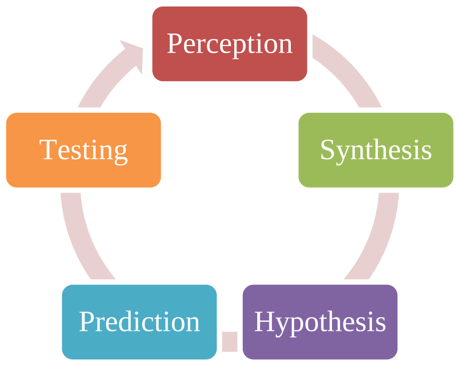


Figure 3: An abridged version of McComb's model of interpretation.

Like context and information, the process of interpretation differs from person to person. For the purposes of our discussion, we present an abridged form of McComb's *Model of* Interpretation as illustrated in Figure 3:

[8] From La Roux's song called Bulletproof. We intentionally leave out the rest of the song and the fact the song concerns personal relationships.

1. Perception – an initial classification of low-level information, such as what our eyes see or things we hear.
2. Synthesis – attempt to match what information we have just received with prior knowledge from previous experiences.
3. Hypothesis – attempt to make sense of what we have just seen.
4. Prediction – try to predict what may happen in future occurrences of perception.
5. Testing – figure out the correctness of what we have just analyzed

Although one may not completely agree with McComb's process, we assert and recognize that previous experiences play a crucial role in how we interpret information. Take, for instance, what Norman calls signifiers [75]. Signifiers represent subtle clues of an entity's surroundings that may aid an entity in finding additional context or information, regardless of whether or not a signifier has been placed intentionally or accidentally. We consider the following example as described by Norman and relate it with McComb's process:

> A train rider arrives at a train station's platform around the time a train has been scheduled to leave for its destination. If the platform still has people on it, then the person probably did not miss his train. Otherwise, the rider may have missed his train.

In this example, the person involved utilizes previous experiences to interpret the situation at hand with the signifiers, or what we call information items[9], presented. The initial *perception* uniquely identifies everything in the person's field of vision: other people on the platform, the presence of a train, and the platform itself. Now, consider the *synthesis* phase. The person attempts to find higher level information items, such as identifying the number of people on the platform. Then, the person forms a *hypothesis*

[9] In this example, information items also include the train rider, the train station, the train station's platform, the train, and the train schedule.

concerning whether or not a train has come yet. Based on the hypothesis, the person tries to *predict* if more people will come or not. Finally, the person attempts to *test* whether or not the train has left, by either asking someone else or just reacting to the presence of new people or detecting the presence of a train.

For this example, we have assumed that the person involved has prior experiences to perform any part of the interpretation. Without any prior experiences relevant to the situation, such as a train station scenario, the person may attempt to assimilate information and fit it with their previous experiences or ask questions relating to the situation to other people to learn and acquire a suitable model for the situation.

3.1.2 Cultural Issues

When dealing with context, cultural considerations needs to be taken into account. A wide variety of cultural influences will affect the way we utilize and specify context. Entities assimilate information from a variety of divergent and heterogeneous data sources that may have relevant data, but may not match directly due to cultural formatting.

A broad spectrum of cultural considerations has been presented by various researchers [98] [73]. Examples of cultural representation issues include time and date (displaying time and time zones), temperature, address formatting, and names. When not taken into account problem, cultural issues could lead to issues of varying degrees of problems. For example, the date *02/03/2009* could be *February 3, 2009* in one country or *March 2, 2009* in another.

While the date example could lead to a missed meeting or a missed flight, cultural misinterpretations could lead to much more catastrophic failures. The Mars Climate

Orbiter (MCO) had been designed to measure climate and weather over the atmosphere on Mars. However, the MCO burned up in the atmosphere due to two companies misunderstanding cultural representations, that of mixing up Imperial and metric units [70]. Since these two units use the same numeric representation, verifying whether a number is an Imperial or a metric unit may be impossible without evidence in addition to the numeric representation. If one component converts a unit from Imperial to metric, the numeric errors add up slowly and may not be caught until it is too late. Appropriate representations and understandings, either specified in software or in a protocol, and appropriate unit tests have to be in place to avoid such failures.

3.1.3 Abstraction Issues

Computer users and programmers use abstractions every single day, whether they explicitly recognize this or not. We do not transmit electric signals into our word processors, as input devices take care of translating keystrokes and mouse clicks into appropriate signals a computer can recognize while output devices translate bits to an appropriate sound or image a human can receive[10].

Computer scientists and engineers create different building components at various levels of the computer building process. These components represent useful building blocks such that experts can build on top of one another at, what we have come to know as, different levels of abstraction [17]. These abstractions, for a computer, allow translation of individual bits into electrical circuits, machine language to individual bits, assembly language to machine language, programming language to machine language, and higher order methods to individual lines of code. Without the notion of abstraction

[10] Petzold has an elaborate example, intended for general readers, on how to build circuits from low-level components [79].

and building blocks, building a computer would be a rather difficult task as it stands today.

Some people assert abstraction represents a fundamental key to computing [56]. For clarity, let us examine the definitions of abstraction [25], which people agree with unlike information and context:

1. *An abstract or general idea or term.*
2. *The act of considering something as a general quality or characteristic, apart from concrete realities, specific objects, or actual instances.*

One further example that illustrates the concept of abstraction comes from creating a transit map. Kramer illustrates this by showing us two transit maps of the London Underground: one which plotted out stations and routes faithfully to their actual geospatial coordinates and another one which has relative positions of stations and routes [56]. People felt the exact representation of all of the information led to a cluttered map, which the general public had a hard time reading. The abstracted version had been found to be much more effective in use by transit riders. Other transit agencies across the world followed suit to abstract their maps.

Travelers looking for stations they need to get to appreciate abstraction implicitly since they do not need every level of detail. When appropriate, a lower level of detail may be obtained when necessary, such as when people need to perform track repairs. The transit mapping example shows how to remove unnecessary details from the set of information items displayed, keeping all of the information items available when necessary, and the computer building example demonstrations how to piece together a complex structure.

Abstraction applies directly to context. We distinguish between two cases of the use of abstraction with respect to context:

1. We do not necessarily need the lowest level of detail of a piece of context. Therefore, we can ignore, but not remove, unnecessary levels of context.
2. It may be necessary to group related pieces of context together. Therefore, we should combine context and create an appropriate representation of related data that could be used to build more elaborate structures.

3.1.4 Accuracy, Resolution, and Latency

In terms of time and location, two of the core aspects of context, we need to distinguish the difference between three terms: accuracy, resolution, and latency. Accuracy means how close the measured value is to the actual value. If we interpret an information item to be accurate enough, then we would not have to perform any interpretations to make the information more accurate. For example, if we have location context accurate to about ten meters, relative to an exact position, this may be sufficient enough for a large variety of applications, while other applications may require sub-meter accuracy [64].

We contrast accuracy with resolution. Resolution means how far apart a value can possibly be measured or detected. In terms of location again, if we take a look at city-level resolution, then we do not need to consider geospatial coordinates, except for possibly determining which city the coordinates reside in. Like accuracy, we need to consider whether or not a particular resolution would suit the needs of an application.

When we talk about latency in computer systems, we can discuss latency from two standpoints. The first standpoint stems from computer networking. In computer networking, latency measures the amount of time it takes a message go from a sender to a receiver. This measure may be either one-way (from source to destination) or round trip (from source to destination and back to the source). The second standpoint comes from a sensor standpoint. Latency in this sense means the amount of delay it took to start a measurement of something, such as temperature or location, to the consumer of the sensor.

3.2 Critical Aspects of Context

While context encompasses several different relationships concerning information items, such as temperature and brightness, we believe all pieces of context contain core elements. We generalize two key ontological commitments from a discussion on spatio-temporal databases [35]:

1. *assume space and time* – location and time
2. *names of objects* – identity

These three critical aspects (identity, time, and location) coexist together. Every relationship concerning information items contains some sense of all three aspects. It may not be relevant for all three of these elements to be discussed for a particular scenario, as they can be abstracted, but these three aspects are be represented in one way or another.

3.2.1 Identity

The ability to name information items, or entities[11], has been one of the most crucial aspects of developing computer systems. In programming languages, we name classes, objects, and variables. In data structures, we use identifiers as keys. In networks, we use identifiers to name resources. Outside of the virtual world, we physically give names to

[11] Examples include people, places, objects, activities, and events.

people, things, locations, and so on. Identities represent virtual naming constructs, as identities by themselves do not exist in a physical manifestation.

We can distinguish identities among three different axes: reality, cardinality, and specificity. In reality, we differentiate between physical (representing an actual object in the physical world) and virtual (represent something that does not have a direct analogue to the physical world). By cardinality, we determine whether an identity represents only a single named entity (such as an individual) or a group of related entities (a type of abstraction). Groups can be further broken down into small-scale (entities in a building), medium-scale (entities on a block), and large-scale (citywide entities). Specificity represents whether or not an identity has a name or could be considered anonymous. A named entity, such as a street name, represents a permanent[12] identification. An anonymous entity may be a chair at a particular moment (the chair a person sits on).

3.2.2 Time

Almost everything we do depends on some notion of time, as we live in a causal universe[13]. In general, every event occurs at one particular instance of time and time can be used to measure the speed of an execution or find out when something might occur.

Three permissible timing-based metrics can be used: performance, ordering, and scheduling. Performance metrics typically involve how long it takes for an execution. For example, a search engine displays how long the internal computations took to complete, but we should note it may take longer for it to be received by a user's computer, and how long it takes for a user to interpret the information displayed on the

[12] Permanent here really means semi-permanent., as described by Frank [35]. For our discussions, we do not get ito the philosophical debate concerning information items, or entities, lasting forever.

[13] We do not argue for or against the existence of a non-causal universe., in which an effect precedes a cause.

screen. Ordering metrics typically ask if we can guarantee a specific order of events[14], such as guaranteeing the correct order of reads and writes onto a hard disk. Scheduling metrics usually involve the time when to execute a particular event or when a particular event occurs. For example, we want to execute a particular program at a specific time or we want to know when a train will arrive.

When talking about time, we can elaborate on it through two different types of timing relationships: absolute and relative. Absolute time represents a date and a time with respect to a global clock, such as using Coordinated Universal Time (UTC). Relative time typically starts from a specific starting point, possibly with respect to an absolute time and may not have a mention of a specific date or time. Unlike absolute time, relative time can be represented by a sequential number in terms of a series of events.

In computer systems, time can be represented in a number of ways, depending on what needs to be conveyed. It could be represented as a numerical digit (the order of execution), a specific date and time, or an execution graph.

When looking at the considerations concerning context, we consider abstractions. For performance metrics, we may want to look at the performance of the system as a whole, series of components, or an individual component. In ordering metrics, a manager may not necessarily concern himself with when individual events finish, but when a series of events have completed. When considering culture, we need to take a look at what issues we may have with time. Although standards exist to account for date and time [51], we still need to be careful how people perceive them.

[14] For an elaborate example on order of distributed events, see Lamport [57].

An advantage of time comes from the fact that since everything happens over a period of time, we can record context over a particular period of time to help determine application behavior [90]. Additionally, we can record context over time to determine how certain events occurred over a particular situation.

3.2.3 Location

Regardless of whether or not an event occurs in a physical location or a virtual space, it happens at a location. This location could be either the exact place of where something had occurred or a location of interest. Like time, location can be expressed in two different ways: physical and logical. Physical location represents a place in the physical world. We can break down physical locations into two additional categories: absolute and relative [42]. Absolute location means a *shared reference grid* for all information items. In terms of geospatial coordinates, this includes latitude, longitude, and altitude. Relative coordinates means with respect to a local frame of reference. Logical location means something virtual in a computer system, such as a computer's Internet address. Both physical and logical locations can potentially have an analogue to one another.

Let us consider the history of physical location representation. Early work only utilized a very basic coordinate system in three dimensions: x, y, and z [71]. However, application-level reasoning cannot always use a coordinate-based system. Therefore, we need to distinguish between numerical coordinates and semantic representations [41]. Semantic representations typically use a hierarchy of representations, such as $\langle building \rightarrow room \rightarrow floor \rangle$ or $\langle country \rightarrow city \rightarrow state \rangle$, abstracting coordinates away from the user.

We can also differentiate physical locations by differentiating physical positions and symbolic locations [42], also called symbolic and geometric models [16]. The Global Positioning System (GPS) [45] is the classical way of obtaining outdoor location. Physical positions typically display latitude, longitude, and, where available, altitude. Symbolic locations typically come in the form of a location relative to a semantic representation, which has a physical coordinate. Examples include *on a train approaching Denver*, *next to a mailbox*, or *in the kitchen*.

At the present time, standardization bodies have begun efforts to standardize differences between geodetic (geometric, physical locations) and civic location types (semantic, symbolic locations) [95]. Additionally, all notions of location obtained by computing devices usually include uncertainty due to the fact these devices[15] may not be able to obtain the exact location [71] [42] [41]. Also, geometric and symbolic representations will usually be organized hierarchically for scalability and abstraction purposes [16].

With respect to location, we can use abstraction to our advantage in which both physical and logical representations can benefit. In physical categories, we can group entities that have similar proximity, such as a street address, a place, a city, or a country, as well as the entities that reside in them. With virtual locations, we can group entities based on virtual categories, such as subnets, networks, and objects in a data structure. While we can use abstraction to our advantage, we need to be careful with cultural considerations. Semantic differences need to be accounted for when describing civic locations. For example, the term *dance club* may mean something completely different depending on the location the term has been used in.

[15] We mean the mechanisms to obtain the location or measurements used in inferring location.

3.3 Categorizing Context

Aside from the three core aspects of context, many different types of context co-exist together. While we may never decide on a global schema or ontology for all possible contexts, when we design a system we have to choose how to classify different types of contexts. From the literature, we explore a small sampling of categorizations, as every classification differs from one another.

In a general categorization of context, a three-dimensional context model has been proposed [92]. This work has broken down context into three separate axes for general level context: self (device state, physiological, cognitive), environment (physical, social), and activity (behavior, task). An analogous model for a general computing system distinguishes models between user, task, and system [60].

In mobile computing, one group of researchers has distinguished between infrastructure, application, system, location, and physical context [89]. In early research in context-aware computing, context has been distinguished between computing, user, and physical context [90], which has been further elaborated by adding time to the context [16]. Some sensor networking researchers distinguish context between user level (human and human-computer communication), context information level (context relevant for the user), and sensor information level (derived from a sensor) [8]. These types of classifications represent coarse-level groupings.

Several context-aware system classifications have distinguished context into much finer grain categories. Rodden, Chervest, Davis, and Dix distinguished the following context categories: space, time, real versus virtual, mobility by movement (fixed, mobile, autonomous), mobility by device type (free, embedded, pervasive), cardinality (personal, group, public), entity type (people, devices, objects), presence, identity, and attributes [89]. Chen and Kotz identified low-level contexts, including location, time, nearby objects, network bandwidth, orientation, and other context (such as light, tilt, vibration, proximity, sound, temperature, pressure, and carbon dioxide) [16].

While there may be several different ways to categorize context, there seems to be no consensus as to how to classify it. Each individual architect of a system determines what would be the most appropriate context to be placed into a system. Architects also decide how their decisions will affect future integration with other computer systems. Although this dissertation does not focus on a particular schema or classification scheme, the architecture we aim to develop has the ability to map context from one model to another mode, hence allowing integration to take place.

3.4 Active Context

One vital consideration concerning context deals with looking at past and present context. Most context-aware systems, which we will discuss in chapter 4, typically only present context is stored and analyzed in the active system. We call this *active context*. For the purposes of this dissertation, we consider active context to be all the context an entity has that holds true at the present time. This does not mean that context from the past may not matter, as some context will stay persistent in active context throughout an entity's lifetime, including a person's gender and birth date or an automated program's start date. Active context differs from past context as past context may not have any direct relevance to the present scenario, except possibly when taking a look at how a piece of active context may have been derived.

We recognize that keeping track of active context and making past context available to the system can be beneficial. However, we need to understand the advantages and disadvantages of utilizing both active context and past context. Rover allows active context to be stored both in-memory and in a database. If stored in-memory, the active context state will be lost if the Rover server shuts down unexpectedly. With past context, the state of the active context up until the point of shut down can be reconstructed. Additionally, with past context, which includes all active context, developers have the opportunity to utilize additional information items to provide additional capabilities or services to people utilizing their development efforts.

Distinguishing between active context and past context can be difficult. Certain context, such as time and location, changes frequently. Therefore, active context can be updated as soon as time and location change. However, other context requires additional processing or even input from the user. Possible context this applies to includes being hungry or thirsty. The hunger or thirst of a person may change on the system depending on if the user has told the system of their desire to eat or drink, or possibly when a person has completed a meal. Depending on the type of context provided to the system, an application designer needs to account for possible situations where active context may be changed.

Active context has a distinct performance and storage advantage over past context. Since there will be less active context available than past context, less operations for comparing context need to be performed and less storage space is required to keep track of the active context.

3.5 Summary

Context has a seminal role in the way entities actuate and interpret information items. In this chapter, we explored four separate themes. The first theme involved issues involved with using context in our everyday lives. Next, we examined the critical aspects of context, which every relationship has to have: identity, time, and location. Then, we took a look at how researchers and corporations have classified context. Finally, we discussed what we mean by active context. Context plays a critical role in the development of system architectures and this chapter outlines the important aspects of context we have to consider prior to using it.

Chapter 4

Context-Aware Computing

The earliest known usage of the term *context-aware computing*[16] came from Schilit and Theimer in 1994 [91] and has been an active field of research ever since. In this chapter, we overview the goals and types of context-aware computing, survey previous work on context-aware systems and taxonomies, examine standards related to context-aware messaging, and context models. Where applicable, we indicate how work in this dissertation compares to related work.

4.1 Goals and Types of Context-Aware Computing

From the early years of context-aware computing, researchers recognized the benefits of utilizing context to aid users and systems. In his doctoral dissertation, Nelson outlined three major strengths of context-awareness [71]. He states *context-awareness leads to*:

- *Automation* – automatically perform repetitive tasks, such as logging and redirection.
- *Adaption* – applications can change its functionality depending on context.
- *Personalization* – users and systems can configure application behavior based on context they specify directly.

Our work directly provides provisions for automation and provides support for adaption and personalization, depending on an administrator's and a developer's customization. In terms of personalization and adaptation, the advantages they provide to users may be seen as a disadvantage to some as context-aware computing appears, *prima*

[16] As stated by Dey [24].

facie, to take control away from the user [9]. With the added context available to users and systems today, if a lot of activities will be automated, users may not want these features, for example, shutting off your phone automatically in a movie theater. In this dissertation, we do not concern ourselves of this conjecture[17], but we acknowledge the levels of interactivity that may be supported by context-aware computing as stated by Barkuus and Dey [9]. These levels of interactivity for users include:

- *Personalization* – users customize aspects of the application to tailor their needs.
- *Active Context-Awareness* – given the context available, an application will automatically change their behavior depending on the situation, such as ignoring text messages while in class or in a meeting.
- *Passive Context-Awareness* – applications make suggestions to users based on surrounding context gathered by the system.

Although these represent the major goals and levels of interactivity for context-aware computing, we recognize that to actually perform any of these tasks requires the appropriate underlying system which supports these methodologies.

During the same time context-aware computing began as an active research field, location-aware computing also came about. With the advent of commonly available location determination mechanisms, such as the Global Positioning System (GPS) for outdoor location [45], the ability to take just one single piece of context, the location, became popular and active research continues to this day. We explore work in location-aware computing before moving onto the more general field of context-aware computing as they share several of the same facets.

[17] See Barkuus and Dey for an empirical evaluation of how different levels of context-awareness affects the way users perceive context-aware computing systems [9].

In some early work, Cyberguide aimed to create components of a location-aware system by personifying them [1]. The components include:

- *Navigator* – positioning component.
- *Cartographer* – mapping component.
- *Librarian* – information component.
- *Messenger* – communication component.

By personifying these components, the authors created a real-world analogue to understand what these individual components actually perform. In another research project, comMotion aims to enhance everyday life of the user with location context [64], using similar components to Cyberguide, but adding functionality to enhance daily life. New functionality includes a query module (find the location of someone, such as a family member), to-do lists, and reminder notifications at particular locations.

In addition to enhancing users' experiences, location context can be used to identify areas of interest [54]. In traditional mapping components, arbitrary boundaries, what the authors call physical zones, have been predefined by a cartographer. These predefined boundaries ignore a *person's use of a space*. For instance, how can one identify where someone typically sits throughout a day. By observing what they call activity zones, one can map out, dynamically, areas of interest. Although the work only discusses a small number of individuals, we believe this could be used in places which have an arbitrary number of entities, in a place such as a museum. This would aid in finding popular areas, and potentially items, of interest.

Location context can be used to provide a seamless location-aware experience over computer networks [27]. This means being able to seamlessly move across heterogeneous wireless networks (both physically and organizationally separated), utilizing multiple location determination technologies, and discovering services based on a user’s location.

Recently, numerous real-time location systems (RTLS) have been developed for use by the general public. The most popular outdoor location determination technology, GPS [45], can be readily accessed by developers for use in mobile devices, as their interfaces have been exposed on platforms such as the Apple iPhone [3], the Google Android [36], and Palm’s WebOS [77]. Applications built on these platforms take advantage of location context, but limited to outdoor location resolution. As indoor location determination systems become readily available, more precise applications which use location context can be developed.

Location-aware computing only provides services with a single core type of context in mind, that is, location. In fact, location-aware systems require additional context to make better use of the components in each system, even though they may or may not explicitly state the context they use in the design of their system. For instance, figuring out what information to give to someone, who to send messages to, and figuring out the appropriate services to use, would require more general context than location.

4.2 Context-Aware Computing

The research area of context-aware computing typically deals with one of two areas: systems and applications. Systems represent the building blocks to create context-aware applications. These building blocks typically deal with being able to handle context as well as provide primitives for a variety of features. Applications use the systems to build software that an end-user would use. These applications would take advantage of the context available to the system as well as the primitives available to it. In this section, we examine the components of a context-aware system as identified by two surveys of such systems.

Baldauf, Dustdar, and Rosenberg surveyed context-aware computing in general [6]. For the purposes of this dissertation, we examine their groupings of what a context-aware framework should include in terms of *design criteria*. They classify the design criteria as follows:

- *Architecture* – What kind of traditional computer system does the framework utilize? Examples include centralized middleware, agent-based, blackboard-based, widget-based, object-based, an extension of the model-view-controller pattern, and distributed systems with a centralized server.
- *Resource Discovery* – What mechanisms does a particular framework have to discover context sensors? This also includes how sensors notify the system when they come online, stop itself, and how the system infers that a sensor failed unexpected.
- *Sensing* – How will context be supplied to the framework? Methods include querying sensor nodes directly, using context acquisition components. The authors mention three types of sensors as defined by Indulska and Sutton [50]:
    - *Physical Sensors* – derives context from physical hardware, including light, visual context, audio, motion, acceleration, location, touch, temperature, and physical attributes.
    - *Virtual Sensors* – context derived from software, such as activities and Internet addresses.

    - *Logical Sensors* – a combination of physical and virtual sensors.

- *Context Model* – How will context be handled, shared, and stored[18]?
- *Context Processing* – How does the framework take *raw, fine-grained data* and turn it into higher-level constructs[19]? This includes using an inference engine, knowledge bases, and reasoning engines.
- *Historical Context Data* – Does the framework keep track of the entire context that passes through the underlying architecture?
- *Security and Privacy* – How, if applicable, does the framework implement security and privacy measures?

Work on context-aware computing, specifically those that employ frameworks, can be classified into the design criteria as described above. This represents only one type of classification. Kjær *categorizes* context-aware systems differently [53]. Although these categorizations have only been applied to middleware-based architecture, this classification, in principle, could be applied to all context-aware systems. The categories described include:

- *Environment* – either infrastructure (entities depend on services of a system) or self-contained (the devices communicate amongst themselves).
- *Storage* – either context will be stored as-is or data the system uses will be ordered according to context.
- *Reflection* – the ability to query metadata either in the application, the middleware system, or the context itself.

[18] We explore context models later in this chapter.
[19] Several possible levels of abstraction can be considered here. Consider the location stack [43]. The lowest level represents direct sensor data, such as radio signal strength indicator values. The next level would be location coordinates. The next level could be room information, then floor information, then building information. As the information moves up the location stack, it creates higher-level constructs.

- *Quality* – measurement of the accuracy and error of the context as well as the resources.
- *Composition* – Can components be activated and/or chained together based on context?
- *Migration* – If available, how can entities migrate from one device to another?
- *Adaptation* – Applies directly to the levels of interactivities. Kjær describes three adaptation methods:
    - *Transparent* – changes happen in the background without intervention from the user or the application itself.
    - *Profile* – applications send their interests to the system.
    - *Rules* – provided either by applications or users, rules typically perform an action if a particular rule has been satisfied.

Both the *design criteria* and *categorizations* have to be taken into account when describing and evaluating a context-aware system or architecture. The reason the authors have been able to distinguish between different categories since each the context-aware architectures they categorize into their work primarily has a specific mechanism for providing each item. For instance, an architecture which supports inferring context from other context only has inference mechanism. Additionally, architectures supporting service composition has a very specific way to perform the composition. In order to not limit developers, our Rover architecture supports capabilities to utilize different mechanisms for each criteria and category mentioned.

We will revisit both when we evaluate our Rover architecture as we believe that they will be able to shed light on the strengths and weaknesses of the Rover system.

Addressing strengths and weaknesses also aid in potentially providing connectivity and bridging the gap between different context-aware systems, work not actively pursued by the research community, except when bridging the gap between context models. Recently, one group of researchers reiterated context-aware research typically only considers single physical locations and multiple physical locations in a single application domain [85].

4.3 Context Models

In order to provide a mechanism to express, store, and possibly evaluate context, designers and developers of systems create models which represent the context itself. Several different types of models have been specified to encapsulate context and we outline the major categories of models as defined by Chen and Kotz [16] and Strang and Linnhoff-Popien [102]:

- *Key-Value Models* – dictionary-based modeling.
- *Hierarchical Models* [20] – a hierarchical data structure consisting of tags and attributes. Hierarchical models can be represented in formats such as the Extensible Markup Language (XML) [116].
- *Graph Models* – use a graph-based data structure to represent context, such as the Unified Modeling Language [76].
- *Logic-Based Models* – utilize formalisms in mathematical logic, giving this type of structure the advantage of having inference engines implied by designers and developers of the models.

[20] Also known as markup scheme models [101] and tagged encoding [16].

- *Ontology Models* – a representation of concepts in a domain and their relationships. Represented in formats such as the Resource Description Framework (RDF) [117] and the Web Ontology Language (OWL) [44]. Like logic-based models, ontology models coincide with the development of reasoning techniques.

Although researchers such as Strang and Linnhoff-Popien [102] and Baldauf, Dustdar, and Rosenberg [6] recommend ontology-based models, we believe that it may not be the best model for every application. We believe that each application has to evaluate each model in terms of their strengths and weaknesses and then select the one most appropriate for it. This includes evaluating each model's expressiveness, reasoning capabilities, storage space, speed of execution, and wire formats.

Baldauf, Dustdar, and Rosenberg also mention "using non-ontology based models requires a lot of programming effort and tightly couples the context model to the rest of the system" [6]. However, we believe that a fine balance has to be found in determining what context model would be appropriate depending on a system's requirements for time critical applications and specific needs. For instance, it may be advisable to utilize a geospatial database over an ontology model for modeling location context, especially when geospatial operations are performed frequently. It would be possible to keep both a geospatial representation with an ontology representation in tandem, but the developer would need to recognize that would require additional storage space, execution time, and the need for synchronization between the two models.

While exploring context models, it became apparent that people only discussed representation formats and inference capabilities for context *for now*. In work done for a context cube representation [40], the researchers recognize that context-aware systems, along with their respective context models, only focus on what they call the *intermediate state*, or the immediate needs of a system. The context cube system, modeled directly after a traditional data warehouse and data cubes, aims to analyze past history. The system can group context together in the three core aspects of context: location, time, and identity.

4.4 Standards for Representing Context

The development of standards allows context, and related information, to be passed between two or more computing devices in a uniform and consistent manner. In general, two types of standards exist: those that give an abstract model of a framework (such as the Open System Interconnection Reference Model [104]) or a concrete standard which specifies the exact way messages should be passed back and forth between entities (either on the same machine or remotely). In this section, we primarily focus on concrete standards. Several different standards today enable context to be specified in a standardized format. We briefly review two standards, one built on another one, here, with regard to communication. All of these standards utilize XML [116] as a wire format, but we imagine they could be represented with different wire formats, such as RDF [117].

The Presence Information Data Format (PIDF) [103] specifies the core context for online presence, such as an instant messaging user. Online presence explicitly identifies the user, their communication status (open or closed), how to communicate with them, preferred communication method, indications of when a particular presence context has changed, and other freeform text. PIDF provides limited context concerning a user, that is, how to contact them and their availability.

The Rich Presence Extensions to the Presence Information Data Format (RPID) provides additional fields to specify context beyond what PIDF provided originally [93]. By building on top of PDIF, through XML Schema inheritance [118], RPID provides the following extended elements:

- *Activities* – the current action, or state, of the user, such as eating or sleeping.
- *Class* – groups together related people, devices, and services.
- *DeviceID* – a unique identifier for each device.
- *Mood* – the state of being of a user.
- *Place-Is* – the physical properties of a place, such as light and noise.
- *Place-Type* –type of place, such as a library or a car.
- *Privacy* – provides what information may be viewable by others.
- *Relationship* – description of how one person relates to another person.
- *Service-Class* – indicates if a service will be delivered electronically, postal delivery, or in-person.
- *Sphere* – a group of related facts which indicates the person's state and role.
- *Status-Icon* – an icon which represents the user graphically.
- *Time Offset* – offset from the Coordinated Universal Time (UTC) of the user's location.
- *User-Input* – indicates how a person communications (such as a keyboard or voice input) and how long ago they sent their last communication.

Several of these elements, such as the activity, mood, place-type, and relationship, define an enumeration of supported elements. In particular, the *place-type* element has been extended by the Location Types Registry [94]. The problem with defining a specific set of possible elements comes from the fact that it may be impossible to enumerate all of the possible elements that could occur. To alleviate this problem, standards typically have a miscellaneous field which allows a user or a developer to allow an arbitrary number of values to be played. However, this complicates a different issue: interpretation.

Suppose two different standards proposing two different schemas, for instance regarding location, the Location Types Registry [94] and the National Information Exchange Model [110]. Developers integrating these two standards have to match each individual element of the two schemas together. This is made more difficult if a miscellaneous field allows an end user or another developer the ability to add their own value. By allowing additional values, natural language processing may be required to attempt to match the values unknown at design time.

Additionally, consider the *mood* element from the RPID standard. If a person fills in the value *pain*, does this mean *physical* pain or *mental* pain? It may be possible that to distinguish between the two types of *pain* with additional context concerning the user in question.

Even though interpretation problems persist, standards provide a way to exchange information uniformly and could be used in useful ways. For instance, if we used RPID's *place-is* element and discover a value of *noisy*, then we know that we should not use *voice* as a communication mechanism. However, we believe that a context-aware framework should not limit context exchange to one particular standard, such as definition types of locations and activities, as this should be evaluated by application designers and developers. Therefore, we believe that any context-aware framework developed have to be able to transform messages from one standard to another standard where applicable.

4.5 A Context-Aware System Examination

In this section, we examine a context-aware system. Dey describes the Context Toolkit, [23]. Dey's dissertation follows a similar format to our discussion here, but our conceptual frameworks differ significantly. Here, we take a look at a few key issues we have identified in the Context Toolkit.

The Context Toolkit consists of *widgets*. These widgets provide context capabilities, including supplying context, aggregating context, interpreting context, and providing context services. The first key difference comes from the underlying infrastructure. All widgets in the Context Toolkit communicate with each other directly, using peer-to-peer communication. Widgets can be discovered by using the Discoverer component. Today, this would not be permissible on most networks. Cellular providers and enterprise networks typically do not allow consumer and employee devices to allow incoming connections.

Application developers must also take care when constructing their applications. Since widgets are decentralized by default, at least a Discoverer must be well-known. Although widgets may know exactly what other widgets they need to connect to, bypassing a Discoverer may not be ideal as widget Internet locations may change, as dynamic Internet Protocol addresses may be assigned to them upon connecting to a network.

Due to its decentralized nature, reconstructing a series of events requires all of the participating widgets to store context as it arrives at the widget, for context aggregators, or when the context is generated. Even though each widget synchronizes its time regularly with a Simple Network Time Protocol (SNTP) [67] server, individual widgets store its context on in-memory local cache. Therefore, should a widget terminate before aggregating the data for logging purposes, a reconstruction of events would not be possible. The Context Toolkit does allow widgets to connect directly to a server-based database, which alleviates the reconstruction problem. However, developers using the Context Toolkit will have to manage their own databases and access privileges to those databases. Context distributed within the Context Toolkit framework are specified with a key-value pair. To discover widgets which supply context or context-enhanced services, consumers pass the exact key-value pairs to Discoverers to find widgets which provide the exact pairs.

Although we disagree with Strang and Linnhoff-Popien [102] and Baldauf, Dustdar, and Rosenberg [6] who recommend ontology-based models for all context-aware systems, we believe that key-value pairs are suitable only after rigid and fixed boundaries have been specified regarding specific applications, with the designers of these applications understanding their context may be limited. Additionally, the strict restriction that all key-value pairs must be specified means that, especially for services, all of the context must be provided. In certain situations, some context may be optional. A context-aware system must allow for both required and optional context.

4.6 Summary

In this chapter, we presented research in the field of context-aware computing. We reviewed the goals and classifications context-aware applications, early work of location-aware systems, generalized context-aware framework characteristics, context models, and a brief look at two related standards which reflect how to represent context in over-the-wire messages.

In our discussion of our context-aware framework, Rover, we will take a look at how Rover can be classified using the same classification elements provided by the work we surveyed here and how we model context.

Chapter 5

The Design of a Context-Aware Architecture

Designing an effective context-aware architecture requires all the lessons learned from understanding the meaning of context, what context should consist of, and learning from designs of existing context-aware architectures. We have designed a context-aware architecture, Rover, which encompasses the core attributes of a context-aware architecture.

This chapter describes in detail the architecture we have designed, including the boundaries between different entities in the architecture, the representation of context, the interfaces to communicate between entities, and critical logging capabilities. We will conclude this chapter by addressing the design criteria of context-aware systems set forth by Baldauf, Dustdar, and Rosenberg [6] and the categories of classification from Kjær [53] by outlining how the design of Rover fits into each criteria or classification.

5.1 Basic Requirements of a Context-Aware System

In this section, we identify the key requirements that have to be included in a context-aware system in order to support as many context-aware paradigms as possible. These requirements have been identified through experience developing Rover as well as the motivating scenarios described in the next section. The requirements we have identified include:

- Organize Similar Entities Together

  Entities in the same organization share similar context. This includes the same location maps[21], profiles, and other context. Therefore, these entities connect to a common server, or logical set of servers, in order to facilitate sharing context between entities. This also separates unnecessary information items from different servers.

- Share Information Items and Context Between Organizations When Necessary

  Even though organizations keep context concerning their own entities on their own servers, it may be necessary to share some context between two or more organizations. When necessary and allowed, context may be shared between different organizations which may span different domains.

- Common Context Data Structure

  Context is frequently passed around entities in a context-aware system. A common data structure which encapsulates context will facilitate context being sent between entities. The common data structure for an individual piece of context should incorporate two of the three core aspects of context: identity (who the context describes) and time (when the context has been generated). As location can encompass a user's entire active context, location can be specified separately from the common data structure.

- Uniform Access to Context, Services, and Messaging

  Entities frequently need access to context storage, call services, and send messages to other entities. Context-aware architectures need to support uniform

[21] We assume organizational entities are located in the same physical location.

access to these mechanisms. Without a uniform access mechanism, entities which need to migrate from one organizational server to another one may not be feasible.

5.2 Motivating Scenarios

Prior to describing our design of Rover in detail, we describe a two part example scenario which has influenced our thought processes throughout the entire design phase. We have been actively working on a quality of life improvement system called MyeVyu. This system takes advantage of the capabilities of Rover and context to enhance the way people use computing devices in their daily lives.

MyeVyu has been developed with a campus community environment in mind and our examples reflect this. However, developers can create services which reflect a wider variety of applications. We examine the following two related scenarios we have developed for MyeVyu to illustrate how context, in general, aids in the flow of information: providing users with context-aware services for everyday needs and facilitating emergency situation response by using context.

5.2.1 Context-Aware Services for Everyday Needs

Individuals in a campus environment traditionally find information about people, events, and places in a static directory. For example, if one wanted to know about the daily lunch and dinner specials at a campus eatery, they would look at the specials posted at the campus eatery, go to the website indicating daily specials, or subscribe to a syndicated news feed to receive updates via his computing devices. These three methods of service delivery do not take advantage of any context to derive custom tailored services. By taking context into account, such as their dietary preferences, a daily special can be custom tailored for each individual on campus.

The ability to take advantage of context for services which people need everyday has several positive implications. Services can now be custom tailored for an individual’s preferences and requirements, as the daily specials example illustrates. Active context, the context which an individual has at the current time, such as location, can further customize a service. When traditionally looking up the local weather, an individual would turn on a television or radio and wait for the local weather to be given or they would go on the Internet and manually type in the local city or zip code. By using location as context, current weather conditions would be displayed without prompting for any input.

Services developed for everyday needs have typically been created in isolation. To create a composition of services, developers take two or more services and create a composite service of interest, such as the popular service composition example of making reservations for multiple components of a trip [100]. The act of service composition only has implied context, meaning a person typically wants to go from one location to another location on a particular date. Having additional context, through calling additional services, would be beneficial.

To demonstrate potential benefits, let us return to the campus environment. Large campuses typically have bus transportation services and real-time arrivals of buses available to the public at large through a service. Suppose a person wants to go from one place to another and has not eaten for six hours. The person finds out when the next bus will be at their nearby stop and how long it will take. Developers can now compose interesting services.

Knowing the trip endpoints, a developer can query a weather service, inform the person of any weather advisories, suggest whether or not to wear, or even just bring, a jacket, and warn the person if they should not travel at all based on the weather. Furthermore, since the trip will take an hour, a service can tell the person what dinner services will be available based on their preferences. Although the person will be leaving at two o'clock in the afternoon, by the time they arrive at the campus eatery, lunch will no longer be served. Many similar examples can be considered once the context information is available to the developers of the applications.

5.2.2 Using Context to Facilitate Emergency Situation Response

Emergency situations can arise at any moment without notice. At one moment, individuals could be using context-aware services, but in an instant a situation requiring emergency personnel could arise. People in a campus environment currently have two options: call 9-1-1 or press a button on an emergency call box. Unfortunately, these mechanisms offer limited context: 9-1-1 [31] has Enhanced 9-1-1 capabilities to provide location [32] and each emergency call box can be correlated with a location. Additional context needs to be derived through a conversation with the operator who answers the alert with the individual sending the alert.

Individuals and operators using a context-aware architecture can take advantage of readily available active context to aid in an emergency response. Individuals communicate indirectly to an operator through an intermediary, such as a context server, which forwards all relevant context to the operator, which includes their identity, location, and medical records, if necessary. The operator also has context concerning first responders, which includes police officers, firefighters, and emergency medical technicians (EMTs), in the area.

When deciding which first responders to send, the operator can utilize context-aware services to aid the decision process. Suppose we know the individual has a medical emergency not caused by any malicious means. The operator knows EMTs would be the appropriate personnel to send to the location. Knowing the location of the individual and the EMTs themselves may not be sufficient, as the actual distance[22] between the entities may be missing other important factors, such as the driving distance, taking traffic into account, and whether or not the first responder and the person have a wall physically separating them. Therefore, a call to the appropriate mapping service which provides driving directions and a separate call for a service which provides traffic notifications would aid the operator in making the most appropriate decision. In fact, it may be possible to notify nearby medical professionals of an emergency scenario in the event that EMTs may not reach the individual in time.

Context represents one of the key aspects of using a context-aware architecture when dealing with an emergency scenario. Being able to playback previous events represents another important aspect when designing and using a context-aware architecture. In the event of an emergency situation, playing back the course of events helps aid investigations, court cases, and training.

[22] The actual distance between two points on Earth can be calculated with the Great Circle distance formula. Assuming latitude and longitude have been represented in radians and not degrees and obtaining the distance in miles, the equation would be $3963.0 \times \arccos(\sin(latitude_1) \times \sin(latitude_2) + \cos latitude1 \times \cos latitude2 \times \cos(longitude2 - longitude1))$.

5.3 Rover Ecosystems

To support the scenarios described, we have designed an architecture called Rover, a support platform for handling context and facilitating the flow of information between all participating entities. Rover has been designed with to support multiple scenarios. Therefore, we logically separate different supporting scenarios into what we call Rover ecosystems (commonly referred to as an ecosystem). An individual Rover ecosystem consists of related entities working together that have related interests in each other. Rover ecosystem administrators (referred to as an administrator hereafter) control various aspects of the ecosystem, including communication mechanisms and authentication.

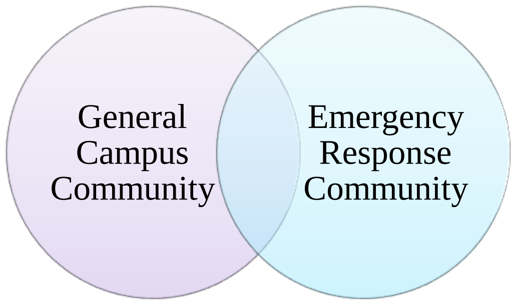


Figure 4: Venn diagram showing two distinct Rover ecosystems with some overlap.

For example, support for everyday needs of a campus community and the emergency response scenarios, described in Section 5.2, contain two distinct Rover ecosystems, as shown in Figure 4: the general campus community ecosystem and the emergency response ecosystem. The individual ecosystems share, amongst themselves, common context and authentication mechanisms. The general campus community contains common services, such as weather information and campus eatery information. The emergency response community does not generally need this information, but shares context amongst themselves, including first responder locations. This location should generally not be shared with the general campus community ecosystem.

When a user on the general campus community ecosystems needs assistance from a first responder, such as when the user has a medical emergency, the user communicates with the appropriate entity on the emergency response community ecosystem. The user may not know the exact location of the entity, as this will probably be considered a security risk. Therefore, we propose to use man-in-the-loop approach for ensuring only the appropriate entities have location information only when they require it.

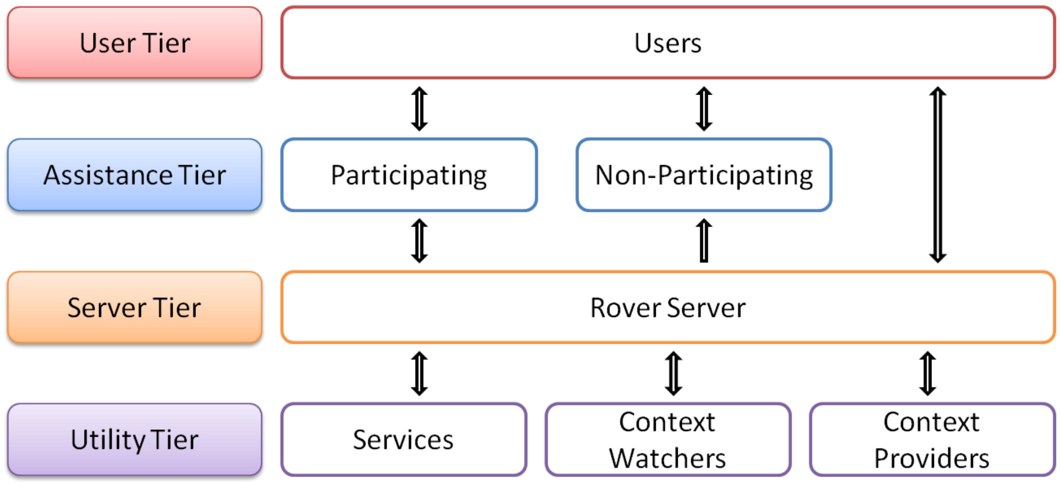


Figure 5: The four-tier entity structure in a Rover ecosystem, showing the logical design of how information flows between entities.

The remainder of this section discusses the different entities which coexist on a Rover ecosystem. We have logically broken down the entities into four different tiers: the user tier, the assistance tier, the server tier, and the utility tier. The way these tiers communicate with each other has been shown in Figure 5. Each Rover ecosystem has one or more Rover servers associated with it. With few exceptions to be explained later, all messages need to be sent through a Rover server. We now explain each of these tiers in detail.

5.3.1 Server Tier

The server tier consists of one type of entity: the Rover server. As the core of an ecosystem, making this an infrastructure-based system, one or more Rover servers have the responsibility for managing all of the entities in the ecosystem, the communication between all of the entities, authorization, secure communication, facilitating the transfer of messages between entities, keeping track of active context, and storing the complete communication history between entities and the Rover server.

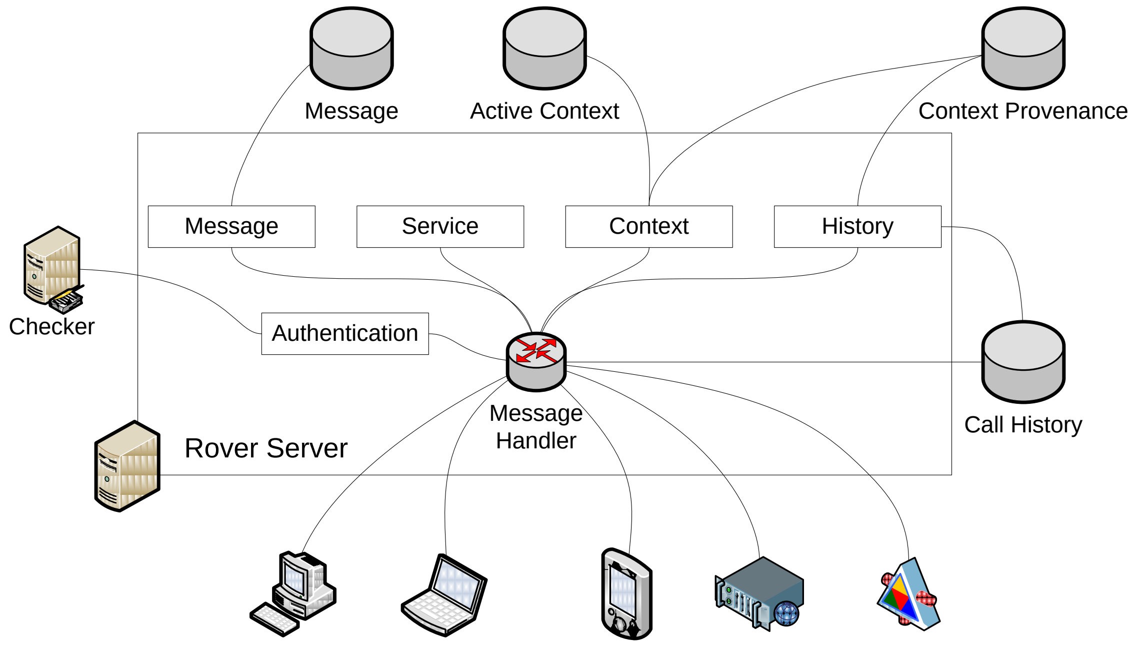


Figure 6: The main operations of the Rover server: a message handler determines the destination of all messages sent to the Rover server.

To be able to support the most number of clients, the Rover server has been designed to allow administrators to choose the communication transport layer protocol entities use to communicate with the Rover server. This includes low-level transport layer protocols, such as the Transmission Control Protocol (TCP) [82] and the User Datagram Protocol (UDP) [83], or high-level protocols such as the Hypertext Transfer

Protocol (HTTP) [34]. Our reference implementation supports TCP and HTTP using custom remote procedure calls (RPCs) for the over-the-wire message format.

Connections to the Rover server should be authenticated. Anonymous connections can be supported, but not recommended. As Rover keeps track of entities, authenticating entities allows Rover to associate all context and activity to a particular entity, while an anonymous connection offers limited historical correlations in the future. One additional advantage comes from the ability of utility tier and assistance tier entities to take advantage of specific, long term context. However, it should be up to the administrator to choose the authentication policy of the ecosystem.

The authentication method has been left open in the current design of the Rover server, as long as the method supports a username and password credentialing method. An administrator may allow entities to be checked against one or more authority domains. Therefore, all entities that want to log into a Rover server supplies a three-tuple data structure representing a credential for logging in:

$$credential \leftarrow \langle domain, username, password \rangle$$

Each individual *credential* has a unique *domain*, *username*, and *password*. An *entity* may log into a Rover server more than once, if the administrator allows this. An administrator sets up credential checkers for each *domain*. The Rover server checks the *username* and *password* against the *domain*'s credential checker.

5.3.2 Utility Tier

Entities in the utility tier represent all backend components. These components can be connected to the Rover server through a network connection or even embedded within a Rover server. Utility tier entities can act as one or more of the following types of entities:

services, context watchers, or context providers. We describe the differences between the three types here:

1. *Services* – Entities which provide a particular service to other entities, with the exception of high bandwidth applications. Entities do not connect directly with services, but indirectly through the Rover server. Services publish their ability to perform a particular task and, if necessary, what context needs to be available for the service to execute properly.
2. *Context Watchers* – Entities which can passively receive context updates via a push mechanism concerning any context provided to the Rover server. Context watchers decide what they will receive.
3. *Context Providers* – Entities which supply context updates to the Rover server. This can be done in any number of ways, similar to the three types of sensors described by Indulska and Sutton [50]. The possible methods include:
    - Communicating context provided by embedded sensors and supplying the context provided by the sensors to the Rover server. Embedded sensors may not be able to connect to the Rover server directly, so context providers act as a bridge to the Rover server.
    - Act as both a context watcher and a context provider such that context provided by the Rover server can be expanded upon. For instance, converting context provided in Imperial units to metric units.
    - Supplying context generated from software related artifacts, such as network characteristics and software availability.

Each of these utility tier entities typically publishes its role to the Rover server so that it may be discovered by other entities in the same tier or other tiers when needed. If necessary, these entities can be chained together to provide related compound services, either by the Rover server or applications which utilize these components. Additionally, utility tier entities can act as a bridge to communicate from one ecosystem to another.

We need to pay special attention to the service utility entities. In many cases, these entities have not been created by the data source or service that actually provides the service. These special entities can act as a proxy to the actual source and augment the call to that source with context provided by the Rover server.

5.3.3 Assistance Tier

A Rover server has the responsibility of transferring information between entities and keep track of context. It has not been designed to handle large bandwidth applications. Entities in the assistance tier, on a Rover server's behalf, handle these types of applications, including large file transfers and video conferencing. Communication between the user tier and the assistance tier depends on the protocols used for communication by the assistance tier entity. Assistance tier entities fall into one of two categories: participating and non-participating.

- *Participating* – Assistance tier entities which actively participate in an ecosystem through communicating with the Rover server. These entities can take advantage of context and services the Rover server knows about. It also enables the Rover server to influence the decisions made by these entities, such as informing a participating entity a particular user has access to a video stream or not.

- *Non-Participating* – These entities participate indirectly in an ecosystem, as they only communicate with user tier entities. They provide useful capabilities, but do not communicate with, or even know about, any Rover server is available. Administrators typically do not have any control over these entities, including video conferencing software solutions which may use proprietary source code administrators do not have access to.

5.3.4 User Tier

Users, and the applications which access the Rover server, operate in the user tier. In an ecosystem, they always communicate with the Rover server directly and have the ability to communicate with assistance tier components to provide high bandwidth applications, such as video streaming. Entities in the user tier do not have direct access to any entities in the utility tier. However, applications can decide how best to use the facilities the Rover server provides to enhance their experience, as they have access to all functionality the Rover server provides.

The Rover server keeps track of user entities logged into the system. If a user moves from one device to another device, all necessary information items, from context to data, can be stored temporarily on the Rover server until picked up from another device. This migration-enabled mechanism needs to be supported by the applications developed, as it represents a separate feature than the Rover server's ability to keep track of every entity's context.

5.4 Context Representation

Except for non-participating assistance entities, every other entity participating in an ecosystem typically supplies context to the Rover server. We have defined a similar triple structure as the Resource Description Framework (RDF) defines [117] and extended it the core capabilities of what RDF provides, (but do not extend RDF directly, as this would represent work beyond the scope of this dissertation). We recap what RDF defines as a triple:

$$triple \leftarrow \langle subject, predicate, object \rangle$$

An RDF *triple* has three parts: a *subject*, a *predicate*, and an *object*. The *subject* traditionally represents a resource in the RDF specification. The *predicate* indicates a relationship the *subject* has with an *object*. The *object* can be a reference to a resource or a literal. For example, $\langle Alice, is\ married,\ true \rangle$ represents a triple with a *subject* Alice, a *predicate* is married, and a literal *object* true, in contrast to $\langle Alice, is\ married, Bob \rangle$, which represents the same *subject* and *predicate*, but defines a resource *object*, Bob. A resource can be traversed, as it can also be used as a *subject*, thereby allowing RDF triples to represents graphs.

5.4.1 Context Data Structures

We relate our context data structure, a context *entry*, to an RDF triple here. All context entries requires a *subject*. This subject includes two fields, a *domain* and a *username*:

$$subject \leftarrow \langle domain, username \rangle$$

A *subject* has a direct correlation to the *credential* data structure, described in the server tier authentication discussion. This allows the Rover server to correlated logged in entities to context entries, as the Rover server automatically sets their logged in context as logged_in. However, *subjects* need not be limited to entities which log into the Rover server. For instance, a context entry could be about the environment of a room, but the room itself cannot log into a Rover server. In this case, the *domain* has to be left blank and the *username* represents the actual *subject* in question. It has been left up to the administrator of an ecosystem to determine how to define these subjects.

Each context entry has two levels of relationship information associated with it, as opposed to the single level *predicate* an RDF triple has, which could be related to other information, but not embedded in the RDF triple itself:

$$root \leftarrow \langle name_{root} \rangle$$

$$leaf \leftarrow \langle root, name_{leaf} \rangle$$

The *root* represents a high level concept, such as location or temperature. A *leaf* has a direct correlation to a *root* and represents a lower level concept. For a *root* containing location, the *leaves* may be latitude and longitude. A temperature *root* could have *leaves* such as ambient *room temperature* and *units of measurement*.

To be useful in deployment situations, context entries need to have values assigned to them. For instance, the *root*/*leaf* value of location/latitude could have 38.992571 as a value and location/longitude would be -76.936312. A context watcher and provider could listen for the entire context with the *root* location and provide a new *root*/*leaf* value for location/building as “A.V. Williams Building.”

If we simply used traditional RDF triples for all of these values as they have been specified. We believe that all related metadata associated with a context entry needs to be stored. We encapsulate the following data structures into a context entry:

$$creator \leftarrow \langle subject \rangle$$

$$entry \leftarrow \langle identifier, subject, root, leaf, value, creator, evidence, timestamp \rangle$$

An *entry* consists of an eight-tuple data structure. We have already discussed the *subject*, *root*, and *leaf* portion of the *entry*. The *identifier* represents a Rover server specific unique identification which allows entities to find a specific *entry* by reference. The *value* has a one-to-one correlation to an RDF triple's *object*. This *value* could be another *entity* or a string literal. The *creator* represents the *entity* which informed a Rover server of this particular *entry*. Unlike the *subject*, the *creator* has to be an entity that authenticates and participates directly with the Rover server.

Evidence consists of either another *entity* or a string literal which supports how supporting data for generating the *entry*. The *timestamp* represents when the *entry* arrived at the Rover server. A *timestamp* can be represented by either a datetime object or a scalar long representing the number of seconds from the UNIX epoch on the local machine running the Rover server.

5.4.2 Context Wire Representations

Each of the data structures defined can be serialized into any number of different over-the-wire formats. Examples include a custom line-based format for line-based communication, the Extensible Markup Language (XML) [116], the JavaScript Object Notation Language (JSON) [19], RDF [117], or any other representation format. Context data structures defined here can be stored on any number of storage mechanisms, including in a relational database, a triplestore, or in a format serialized directly from the programming object with support from the programming language.

Administrators of an ecosystem have to realize that the decisions they make in choosing which wire format to use and what storage mechanism will affect the computing environments they can choose. Context typically has models, or schemas, associated with them. Again, it will be up to the administrators to decide how to define schemas of related context.

5.5 Application Programming Interface

Rover servers support calls that utilize the available context and services in its ecosystem. Therefore, we specify the application programming interface (API) that needs to be supported by all Rover servers.

5.5.1 Context API

Function	Description
InformContext	Send the Rover server context concerning yourself.
SupplyContext	Supply the Rover server context concerning another entity.
ObtainContext	Queries the active context entries.
ObtainContextByReference	Obtains an active context entry by reference.
WatchContext	Subscribes to specific context updates.
UnwatchContext	Unsubscribes from context update notifications.

Table 1: Supporting context operations on the Rover server, including supplying, subscribing, and querying.

Every entity which participates in an ecosystem has access to operations involving context on the Rover server, including supplying context, subscribing to context, and unsubscribing to context. Table 1 lists the API available to all connected entities.

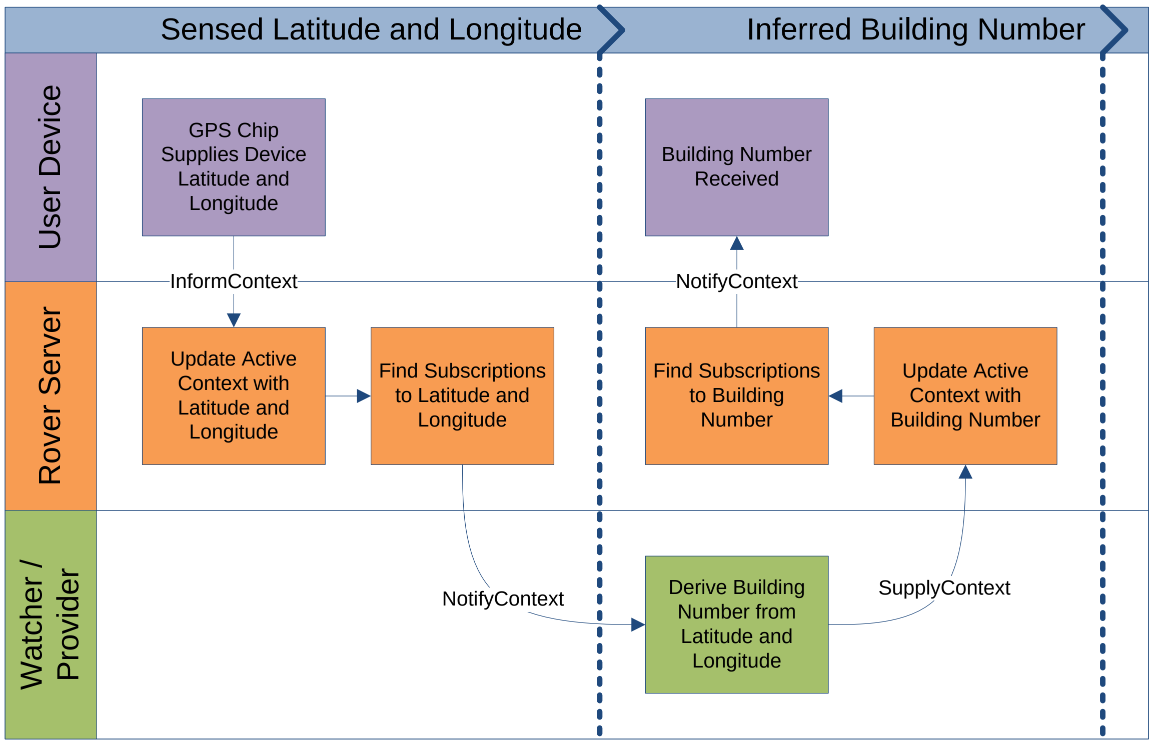

Sensed Latitude and Longitude
Inferred Building Number
User Device
Rover Server
Watcher / Provider
GPS Chip Supplies Device Latitude and Longitude
InformContext
Update Active Context with Latitude and Longitude
Find Subscriptions to Latitude and Longitude
NotifyContext
Derive Building Number from Latitude and Longitude
SupplyContext
Update Active Context with Building Number
Find Subscriptions to Building Number
NotifyContext
Building Number Received

Figure 7: Example workflow of expanding latitude and longitude context to a building number.

Supplying context to the Rover server can be executed by two functions: *InformContext* and *SupplyContext*. *InformContext* updates an active context entry for the calling entity, whereas an entity calls *SupplyContext* to perform an update to the active context on behalf of another entity. For instance, if entity A calls *InformContext* and updates its location/latitude and location/longitude, entity B could be a context watcher and provider which calls *SupplyContext* to update entity A's active context on its behalf. An example workflow of how this works has been shown in Figure 7.

Both *InformContext* and *SupplyContext* automatically overwrite the root/leaf's value. Going back to the context entry example, the root/leaf value for location/building could be "A.V. Williams Building" with a location/latitude value of 38.992571 and a location/longitude value of -76.936312. In certain scenarios, supplying a particular context entry would nullify other leaves that contain the same root. For instance, if the location/building changes to “Computer Science Instructional Center,” the location/latitude and location/longitude values would not be the same as the A.V. Williams Building example, but in fact would have a location/latitude value of 38.98998 and a location/longitude value of -76.9362. A developer needs to decide if updating only a root/leaf pair is appropriate or if not removing other leaves with the same root would make entries in the active context contradict themselves.

We support querying context from the Rover server through *ObtainContext* and *ObtainContextByReference*. *ObtainContext* allows entities to query active context by specifying the exact context they want or an expression. Entities can query the active context by any combination of root, root/leaf, root/leaf/value, and subjects. *ObtainContextByReference* allows entities to obtain a specific context entry by a reference identifier. This may be desirable in situations in which a context entry may be referenced by service composition, but not every individual, entity may not need the values of a context entry.

Interested entities, such as context watchers, subscribe to context with the *WatchContext* function. This operates the same way as *ObtainContext*, but instead of querying the active context, entities will be notified when another entity calls *InformContext* or *SupplyContext*. When an entity no longer has an interest in a subscription to a particular context, the entity calls *UnwatchContext*.

5.5.2 Service API

Function	Description
PublishService	A component informs the Rover server of a service it can provide.
UnpublishService	Indicates that a service will no longer be available.
ListAllServices	Lists all of the services available on the Rover server.
ListRelevantServices	List of services which can be accessed, dependent on context.
CallService	Calls a particular service.

Table 2: Rover function calls allowing discovery and calling of services.

Services provide the ability to bridge data services outside of an ecosystem as well as internal mechanisms as well. Table 2 lists all of the service functions available on a Rover server. To allow services to be discoverable, utility tier service entities call *PublishService*. The function call includes the name of the service, a description of the service, which could contain required data another entity have to provide when calling the service, the required active context of the calling entity, and any optional context it may be able to use. To remove the service from being discoverable, service entities call *UnpublishService*.

Discoverable services can be found by other entities connected to the Rover server by calling *ListAllServices* and *ListRelevantServices*. *ListAllServices* returns a list of every single available service without regard to whether or not the entity calling *ListAllServices* can even call each service based on their active context, while *ListRelevantServices* filters the service list by whether or not the calling entity can actually call the service at the moment *ListRelevantServices* had been called.

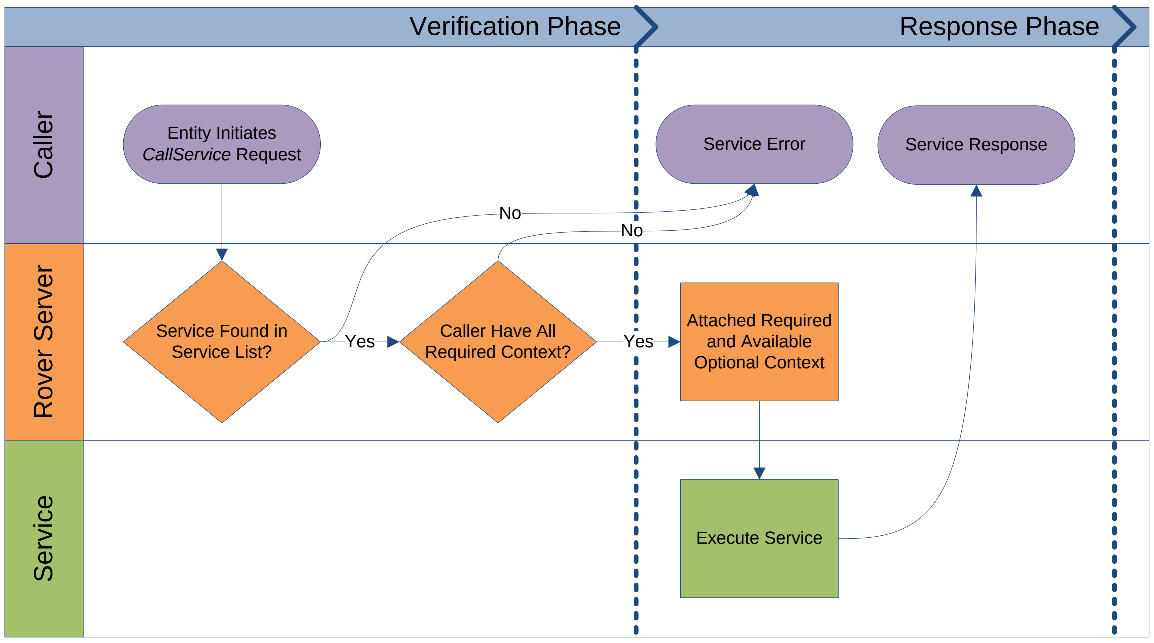

Verification Phase
Response Phase
Caller
Rover Server
Service
Entity Initiates *CallService* Request
Service Error
Service Response
No
No
Service Found in Service List?
Yes
Caller Have All Required Context?
Yes
Attached Required and Available Optional Context
Execute Service

Figure 8: The lifetime of a *CallService* RPC call.

To execute a service, an entity calls *CallService*. The entity has to specify the *subject*, which includes the *domain* and *username*, of the entity which will handle the execution, the service name, and any request data that the service may require. Figure 8 illustrates the lifetime of the call. The Rover server determines whether or not the service can be called based on the active context. If it can be called, it forwards the entire function call to the *subject*, which executes the desired service and returns a response. The response will be sent through the Rover server and forwarded to the calling entity. If the service cannot be called due to a lack of appropriate active context, a network error when calling the remote service, or a service error, the calling entity will receive an error message.

5.5.3 Message API

Function	Description
SendMessage	Sends a message to another entity.
BroadcastMessage	Sends a message to entities fulfilling certain contextual properties.
GetMessages	Obtains stored messages from the Rover server.

Table 3: Functions for communicating to other entities in an ecosystem via the Rover server.

The Rover service provides a messaging API to allow entities to communicate directly with one another. Connected entities receive messages as soon as the message has been sent to them via the Rover server. However, if an entity has been sent a message and it does not have an active connection to the Rover server the message has been sent through, the Rover server needs to store the message in a mailbox so that it may be retrieved at a later time. The message functions available on a Rover server have been summarized in Table 3.

Entities can send messages by either calling *SendMessage* or *BroadcastMessage*. *SendMessage* sends a scalar string over to a specific entity. *BroadcastMessage* also transmits a scalar string, but instead of specifying a specific entity, the message transmitter specifies active context, similar to the *ObtainContext* function. *BroadcastMessage* sends a message to all entities with active context entries, including those that may not be online, that satisfy the constraints given. Should an entity not be connected to the Rover server at the time another entity sends a recipient entity a message, the recipient can call *GetMessages* to retrieve stored messages.

5.5.4 History API

Function	Description
ObtainContextProvenance	Retrieves context provenance, filtered by parameters.
ObtainCallHistory	Sends back the call history, filtered by parameters.

Table 4: Querying functions for historical data on the Rover server.

One of the key advantages of providing an infrastructure-based ecosystem comes from handling all of the function calls, context, messages, and services, through server tier: a central server or set of servers. This allows us to store everything which passes through the server tier. All of the data stored can be used by entities during the execution of a call or situation, a playback of a sequence of events, or performing offline calculations at a later point in time. The server tier supports two types of historical logging: context provenance (all updates to context) and call history (all communication to and from the Rover server). A Rover server supports two read-only calls, summarized in Table 4: *ObtainContextProvenance* and *ObtainCallHistory*.

Context provenance represents all of the context entries that have been recorded in a Rover server. Every single entry will be stored in a database and has the exact fields as a context *entry* with the addition of when context has been deleted: the *removal* field. Provenance will be stored as follows in an eight-tuple:

$$provenance \leftarrow \langle subject, root, leaf, value, creator, evidence, timestamp, removal \rangle$$

All updates to the active context will be recorded as a *provenance* record with the *removal* flag set to false. The recording mechanism for *provenance* does not allow records to be deleted. If a context *entry* has been removed from the database, a new *provenance* record will be created with the *removal* flag set to true. Querying *provenance* records can be done through the *ObtainContextProvenance*, which allows the records to be filtered by any of the record elements.

All function calls on a Rover server are captured by the call history recording mechanism. We support retrieving any of the records in the call history by calling the

ObtainCallHistory function. The record consists of a nine-tuple as follows, followed by a description of all of the elements:

$$history \leftarrow \langle source, target, meta, function, info,$$
$$request, response, timestamp, duration\rangle$$

- *source* – the calling *entity*
- *target* – the intended *entity* of the function call, such as the *entity* which executes a server or the *entity* which a context update concerns
- *meta* – the type of call, which could be *context*, *service*, *message*, or *history*
- *function* – the name of the function
- *info* – any additional data which the Rover server decides to store about the call
- *request* – all of the parameters of the function call, stored as a dictionary
- *response* – the entire response given back to the calling *entity*
- *timestamp* – the initial reception of the call by the Rover server
- *duration* – the amount of time, in milliseconds, the call took to complete

5.6 Presence and Context Standards in Rover

Rover does not explicitly utilize any standards for either presence or context. Since messages require our own custom format, using a standard for communication to and from the server tier, more specifically the Rover server, does not seem reasonable. However, all of the other tiers (user tier, assistance tier, and utility tier) can utilize any standard they see fit to communicate with entities outside of a Rover ecosystem. The Rover server allows multiple standards representing the same context and information items to be matched together, as long as an architect matches these items correctly.

5.7 Summary

In this chapter, we presented the architecture of a Rover ecosystem, the specification of a Rover server, and the data structures surrounding how to represent context. We summarize the design of a Rover ecosystem and a Rover server by taking a look at how it fits into key aspects as discussed as design criteria by Baldauf, Dustdar, and Rosenberg [6] and the categories of classification from Kjær [53]. We outline how Rover fits into their descriptions in Table 5.

	Criteria/Category	Evaluation
Baldauf, Dustdar, Rosenberg [6]	Architecture	Four-tier server-based ecosystem. These ecosystems separate logical functionality and can communicate with each other.
	Resource Discovery	Services and context may be discovered.
	Sensing	Context providers and users
	Context Model	Custom RDF-like model. Schemas can be specified, but are not explicitly placed into a Rover ecosystem.
	Context Processing	Context stored on server, processed by external components.
	Historical Context Data	Context and call logs kept on server.
	Security and Privacy	All entities and components directly connected with the Rover server have to be authenticated. Communication may be done over a secure channel. Privacy has not been addressed directly in the architecture, but may be implemented by an administrator as part of the ecosystem.
Kjær [53]	Environment	Infrastructure-based, as individual nodes do not communicate directly with each other.
	Storage	Depends on the Rover server implementation.
	Reflection	Context can be queried and received by all participants.
	Quality	Can be supplied in a context entry's evidence parameter.
	Composition	Services, context watchers, and context providers may be combined by chaining input and output from each service.
	Migration	Can be supported by specific services, if desired.
	Adaption	Custom designed by a Rover system ecosystem administrator.

Table 5: An evaluation of the Rover architecture with respect to previous context-aware framework discussions.

The criteria and categories not address directly by our work in this dissertation include *security and privacy*, *composition*, and *adaption*, as they fall outside of the scope of this work and will usually be left up to the a Rover ecosystem administrator to decide how to implement these key ideas. However, we believe that the Rover ecosystem, based on the evaluation, provides all of the key benefits needed to provide appropriate context mechanisms to support the scenarios we outlined in this chapter.

Chapter 6

The Implementation of a Context-Aware Ecosystem

In Chapter 5, we examined the design of a context-aware ecosystem. When implementing the features outlined from the design, we have to recognize the limitations of the realization of the logical design to a physical one, as well as the additional features that may be required in addition to the design. This chapter presents the implementation of a Rover server in detail.

6.1 Desired Attributes of a Rover Server Implementation

Although we have implemented only a reference implementation for a Rover server, for the purposes of this dissertation, we still need to develop the reference implementation with attributes that would make deployment on a larger scale feasible. We have identified the following attributes to be critical for a reference implementation:

- *Independent Over-the-Wire Communication Medium*

  The Rover server should be designed in such a way that its features can be utilized from any network. Since a Rover server for any given ecosystem will always be connected to the Internet or a specific intranet, as long as each entity which connects to the Rover server has a path to it, this would be suitable. Whether an entity connects to a Rover server via a cellular network or over traditional Internet, the Rover server should not be affected in any noticeable way which affects its operations or functionality.

- *Cross-Platform Connectivity*

  We cannot assume we can force entities to develop programs in one particular way, as the world consists of numerous heterogeneous devices and programming languages running on a variety of platforms and development environments. A Rover server should support as many devices as possible. Although HTTP can be used as a transport protocol, the fact that HTTP push from the server does not have widespread acceptance, other mechanisms should be supported in addition to HTTP to support push capabilities directly.

- *Independent Wire Formats*

  A Rover ecosystem developer chooses a wire format suitable for the most number of clients that will be used in the ecosystem. Simple text-based solutions, XML, and JSON offer suitable formats that could be used for delivering messages from the Rover server to connected entities.

- *Cross-Platform Server Deployment*

  A Rover server should be designed in such a way that the core components for the Rover server, including backend databases, should be able to be run in any number of operating systems and platforms.

6.2 Prior Rover Server Implementations

The first Rover implementation from 2002 focused primarily on two goals: building a mechanism for enabling location-aware computing and providing a scalable server for location-aware clients connecting to it [7]. As described in this dissertation, the current design of Rover goes beyond using location as the primary means of context. Focusing primarily on location limits the extensibility of a Rover server.

The key feature of the first Rover server implementation had been its contributions to providing a processing scheme for interleaving computation code with disk and network operations effectively by using what the authors called *actions*. Actions provide an efficient mechanism as opposed to using traditional multithreading. The current version of the Rover server does not consider high efficiency to be its main goal, but uses similar ideas in its implementations through the use of asynchronous programming support from the runtime environment the Rover server executes on.

After the first Rover implementation, the focus shifted from location-awareness and efficient computations to supporting a system which handles general context. In 2007, we described a version of Rover in the same spirit as this dissertation's description of how to design a context-aware architecture [2]. The previous version discussed the overall system centralized around the Rover server itself. We now focus on what we call a Rover ecosystem. By understanding the interaction of how different entities interact with each other as a whole, we can better conceptualize how to represent and utilize context in general.

This dissertation describes a more extensive API than we previously described. Although our current work utilizes the same underlying programming language and framework for development, as described below, we support a remote procedure call (RPC), an HTTP interface for better support for a wide variety of end-user devices and service endpoints, as well as a line-based TCP interface to support clients that cannot directly utilize our RPC implementation and require push notifications.

6.3 Rover Server Implementation Details

This section describes the server tier in detail. Specifically, how we have constructed a Rover server. We developed the Rover server using a Linux-based distribution, Fedora Core 7 [88], running Linux kernel version 2.6.23.17-88.fc7 [61] using a single core 1.5 GHz Intel processor with 512 MB of RAM. We chose non-state-of-the-art equipment to demonstrate that it would be feasible to run a Rover server on any type of machine. However, we recognize using a lower end machine means that it will not be able to handle as many concurrent connections and have lower throughput than a state-of-the-art server.

6.3.1 Development Environment

Even though we utilized a Linux-based operating system, we chose to use Python, version 2.5, a cross-platform scripting language [86]. By using a cross-platform scripting language, we can run a Rover server on practically any platform which Python has been ported to, including Windows-based machines.

Although Python provides networking capability in its standard library [87], we chose to utilize the Twisted Networking Library [108]. Twisted provides true asynchronous capabilities so that networking and disk operations can be done independently of computations, similar to the first Rover server implementation [7]. Like Python, Twisted has been ported to several platforms, including Windows.

We seriously considered developing a Rover server using the Microsoft .NET Framework [81], version 3.0, and utilizing its built-in Windows Communication Foundation (WCF) for communication and networking, as it has support for a wide variety of service-oriented protocols. By using service-oriented protocols [12], we can reach a wide variety of devices. However, we opted not to use the Microsoft .NET Framework. WCF would have provided us industry standard service-oriented protocols, but in order to support push messages, in cases such as the *SendMessage* and *Broadcast* function calls, devices have to utilize a Windows-based platform which supports the Microsoft .NET Framework[23]. Additionally, all of these messages must be delivered over HTTP. Therefore, devices have to be able to start a web server to support pushing messages. Even if all devices supported this, it would not be feasible to do this as network policies and device constraints would not allow this in all situations. Therefore, we chose Python and the Twisted Networking Library instead.

6.3.2 Remote Procedure Call Mechanism

For prototyping purposes, we chose to use Twisted's Perspective Broker. The Perspective Broker package provides an RPC-style mechanism for communication. We decided to utilize this instead of a customized line-based protocol[24] as it makes it easier to pass scalar values and complex values, such as lists and dictionaries. Although made specifically for Python and the Twisted Networking framework, the over-the-wire format for communication with the Perspective Broker package has been fully specified. This allows interoperability with other programming environments. A Java programming language [37] implementation exists, called TwistedJava [107]. We have written example client code using both Python and Java over TCP and transport layer security (TLS).

[23] Like Twisted's Perspective Broker, it may be possible to support other platforms.

[24] We wrap the Perspective Broker mechanism to support a line-based protocol to support clients that cannot utilize the Perspective Broker mechanism.

6.3.3 Mobility Support

The Perspective Broker implementation requires a long-lived connection to the Rover server from the connecting entities. However, it may not be entirely possible to keep connections actively open all of the time. For instance, when a mobile device roams from one network to another network, it will reconnect and drop a Perspective Broker connection to the Rover server each time, depending on how networking has been implemented on the device. This would require a device to keep track of, for instance, the entire context it has already received, request context it has not received, and keep informing the Rover server it needs to watch context each time it reconnects. Therefore, in addition to a straight implementation of using a direct connection to the Rover server, we have implemented an intermediary Rover proxy which impersonates clients when connecting with the Rover server. Figure 9 shows the components which support mobility, where PB stands for Perspective Broker and entity can be any user, assistance, or utility tier entity.

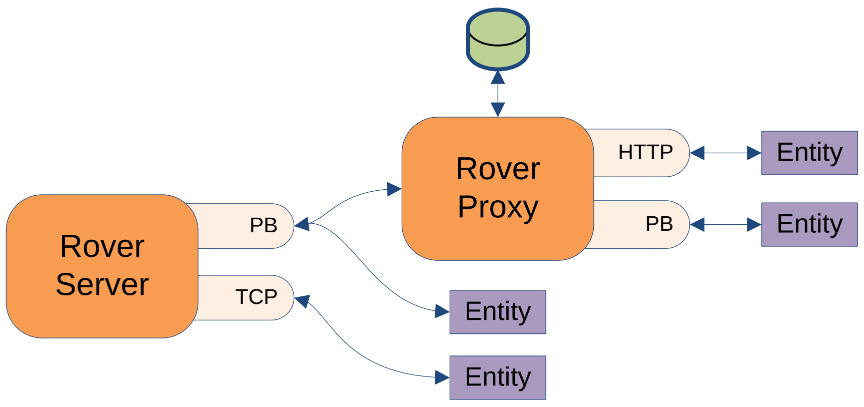


Figure 9: Illustration of the components used to support mobility.

The intermediary typically runs within the same runtime environment as the server, on a different runtime on the same physical machine as the server, or another machine altogether. In our reference implementation, the intermediary runs on the same physical machine, but not in the same runtime. A Rover ecosystem administrator can decide the actual implementation details, but typically, if the Rover proxy does not reside on the same machine as the Rover server runtime, then it should use a secure connection protocol. A Rover proxy supports connected Rover entities which drop their connections from the Rover proxy and reconnect it at a later time. The Rover proxy keeps an active connection with a Rover server, on behalf of the client, at all times. This allows context subscriptions and messages to keep flowing to the proxy. The proxy stores all context and message notifications in-memory. When disconnected entity reconnects with the proxy, all backlogged notifications are sent immediately to the entity without the entity having to query the Rover proxy or Rover server.

The intermediary Rover proxy makes it possible to create an HTTP-based querying mechanism. To do this, we developed a custom web server, also written in Python using the Twisted Networking Framework. This server utilizes the intermediary Rover proxy class directly. We support HTTP and HTTPS connections. The parameters for the RPC calls have been mapped directly into HTTP POST data as a dictionary data structure with the parameter name as the key and the contents as the value. In this implementation, serialized data to be placed in the HTTP POST data has been represented over-the-wire with the JavaScript Object Notation (JSON) protocol [19]. Other protocols can be built for use with the Rover server and intermediary Rover proxy, including line-based protocols or service-oriented protocols.

6.3.4 Authentication

Authentication for the Rover server has been designed with multiple domains and organizations in mind. We designed the Rover server with domain-based credential verification architecture for authentication. Every entity that makes a connection to a Rover server or an intermediary Rover proxy must be authenticated in the reference implementation. In most cases, a secure communication channel with TLS would be desirable, though a Rover ecosystem administrator may feel that an unsecure TCP connection may be suitable for certain scenarios.

Perspective Broker supports supplying a username and a password upon login. Since this does not have support for adding an additional parameter, we require the domain of an entity to be embedded in the username field. To send credentials over HTTP or HTTPS, entities utilize HTTP Basic Authentication. We recap the credential data structure, as specified in Chapter 5.3.1, here:

$$credential \leftarrow \langle domain, username, password \rangle$$

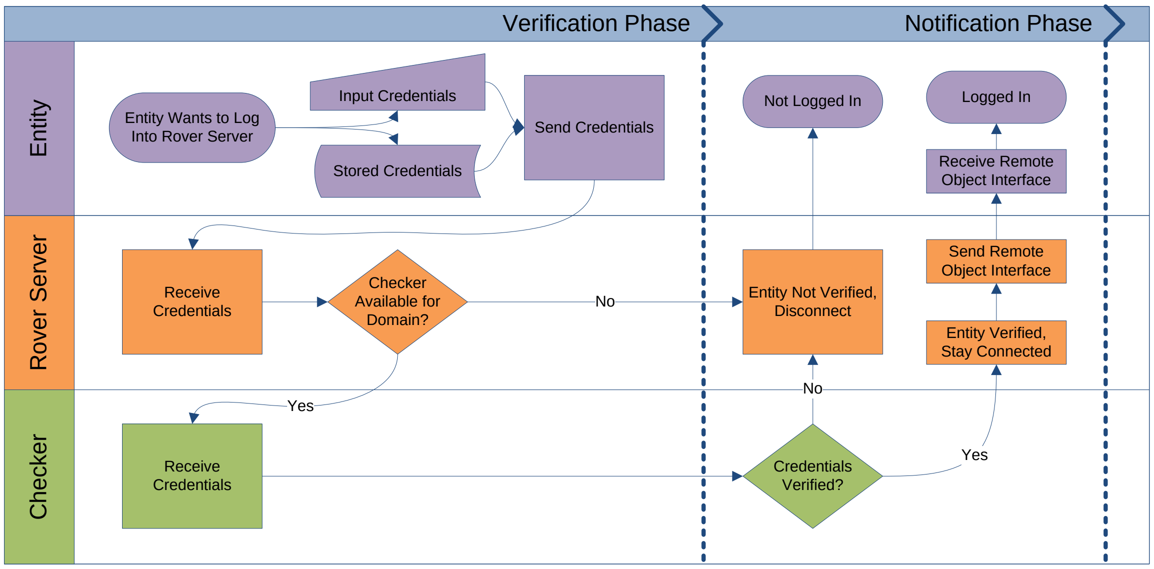


Figure 10: The Rover server authentication mechanism. This shows how the three components interact with each other in two different phases.

Each individual Rover server sets up one or more *verifiers*. Individual *verifiers* relates directly back to an individual domain. If a Rover server receives credentials in which a *verifier* has not been setup for a particular domain, the entity will immediately be rejected; otherwise, the remaining credentials will be checked against the assigned *verifier*. The flow of the authentication process can be seen in Figure 10.

We have created two reference verifiers: a local database credential mechanism and remote verification via the lightweight directory access protocol (LDAP) [119]. For the local database verifier, we implemented a simple database to verify whether or not a username matches a particular password. For database connectivity, we utilized SQLObject [13], an object-relational mapping package, which supports a large variety of databases. We implemented the database in SQLite [49], an embedded database which runs directly from the application itself, though we could have chosen a more traditional client-server database architecture instead.

Verifying credentials need not be limited to the local machine the Rover server resides on. A Rover server can connect to a remote server if credentials will be passed to a remote *verifier*. A relational database management system does not need to be used for verifying credentials. In this reference implementation, we implemented a query to a remote LDAP server to verify the credentials. Other *verifiers* can be implemented as necessary, depending on the needs of a Rover ecosystem and existing credentialing systems available.

Upon a successful login to the system, specialized context for the entity is automatically supplied by the Rover server to the entity's active context. The user will have a "rover/online" value of "true" and the way they log into the Rover server as "rover/connection", consisting of a value of either "direct," "proxy", or "HTTP." When an entity disconnects from the Rover server, the "rover/online" active context will change to "false."

6.3.5 Database Support

In addition to the database support for our local database credentialing verifier, we extensively use databases for active context storage and for logging each and every command. This reference implementation uses SQLite [49] and SQLObject [13] for all database operations. We choose SQLite over other traditional client-server database implementations as this provides us with the most portability in our implementation.

Although we have created our own context model and storage, as described in Chapter 5.4.1, we do not preclude an implementation using other representation or storage mechanisms. For instance, we believe it would be possible to map all of the data structures for context modeling in the Resource Description Framework (RDF) [117], as there have been efforts to extend the RDF representation structure with important contextual attributes such as time [38]. However, we do not explore this particular implementation as extending work done in the Semantic Web goes beyond the scope of this dissertation.

We recap our basic context model from Chapter 5.4.1 below:

$$subject \leftarrow \langle domain, username \rangle$$

$$root \leftarrow \langle name_{root} \rangle$$

$$leaf \leftarrow \langle root, name_{leaf} \rangle$$

$$creator \leftarrow \langle subject \rangle$$

$$entry \leftarrow \langle identifier, subject, root, leaf, value, creator, evidence, timestamp \rangle$$

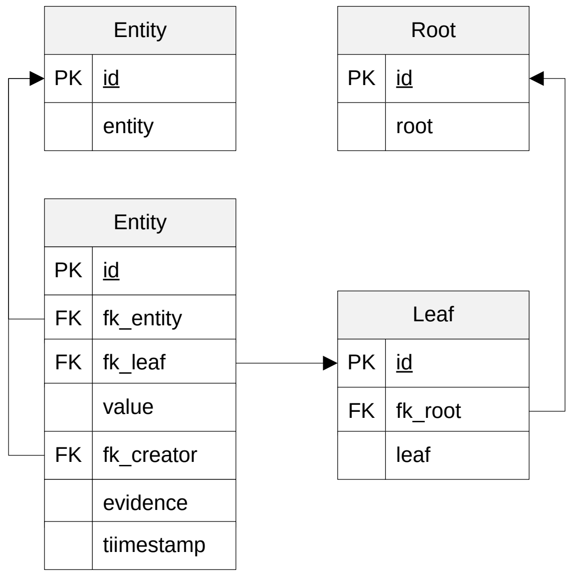


Figure 11: Database diagram of how the context model has been represented in a relational database.

The *root* and *leaf* object pair must be unique per Rover server. Figure 11 shows the relational database representation of the context data. Since we use this storage mechanism for fast access of active context and not a previous state, we must support this context storage mechanism for in-memory databases. SQLite has support for both disk- and memory-based storage. Should a Rover server terminate unexpectedly and the context database for active context uses memory-based storage, we can recover the last available active state using the long-term storage of context history and, possibly, the call history.

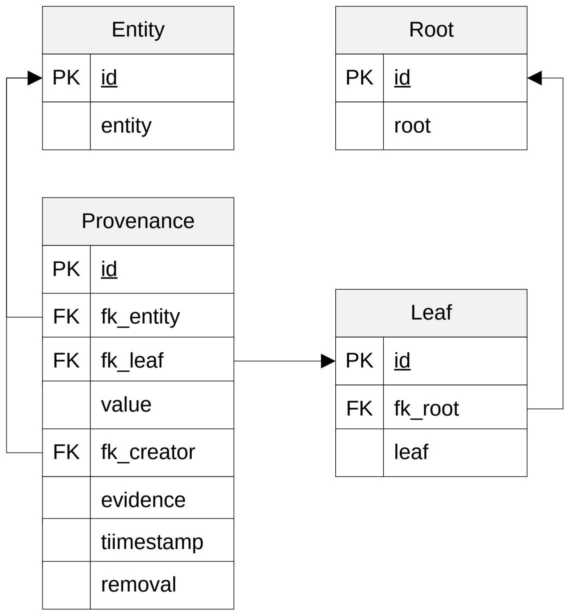


Figure 12: Context provenance database structure.

Context history and call history must be stored by the Rover server on a long-term storage device, such as a hard drive or a network-attached storage device. We cannot use in-memory storage for this as provenance and history will be used for analysis at a later time and for recovery of necessary active context. We recap the model for context provenance and call history from Chapter 5.4.1 below:

$$provenance \leftarrow \langle subject, root, leaf, value, creator, evidence, timestamp, removal \rangle$$

$$history \leftarrow \langle source, target, meta, function, info, request, response, timestamp, duration \rangle$$

The context provenance database structure, as seen in Figure 12, looks very similar to the *Entry* table in Figure 11 with two minor differences. First, we added a *removal* field. The *removal* field makes it easier to reconstruct the active context from the context provenance, when necessary. When removing a context entry, we do not remove the row from the context provenance database; we set the *removal* field to true. Additionally, we also place a *root* column as well. Although this nullifies the effects of database normalization, this allows an easier way to perform offline processing at the expense of an extra 32-bit column.

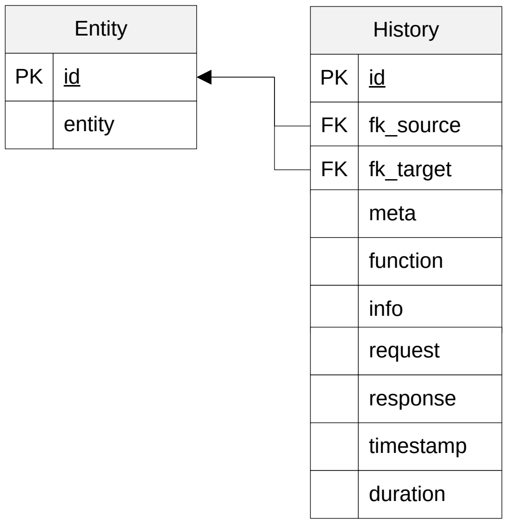


Figure 13: Call history database structure.

The call history is stored in a database with the structure illustrated in Figure 13. All calls to the Rover server will be recorded, including every single part of the request and response. Since the request and the response values may be complex objects, we store them as serialized JSON objects to ensure portability between implementations. Storing the objects using the Python pickle serialization method severely limits portability.

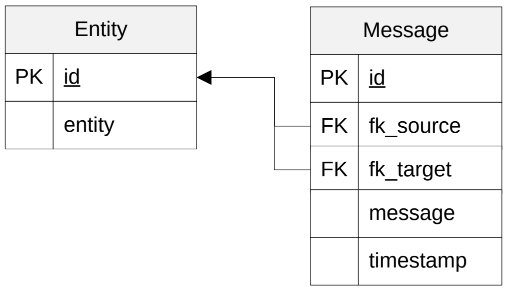


Figure 14: Message database schema.

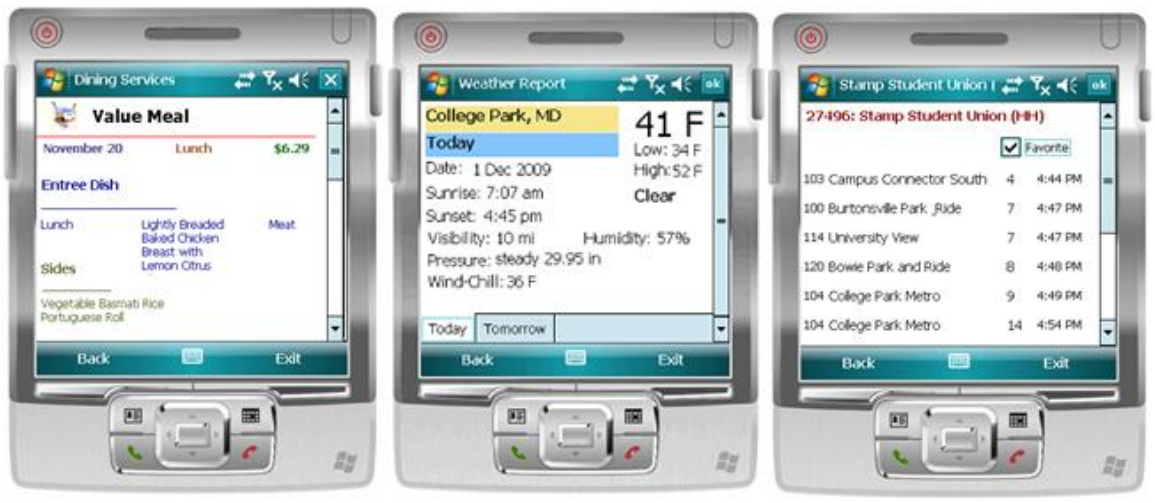
The *SendMessage* and *BroadcastMessage* API calls send messages to a particular endpoint through a Rover server. If the message's recipient does not have an active connection to a Rover server, an intermediary Rover proxy, or has been using HTTP as its communication message, it must be stored on a disk as Rover's messaging system must be implemented with a mailbox. Therefore, we have implemented a simple mailbox system with a database schema as shown in Figure 14.

6.3.6 Rover API

The RPC interface to the Rover server follows the specification in Section 5.5 faithfully. All of the functionality has been described in detail in Appendix A, from A.3 to A.6. However, the Rover server specification does not include considerations for real world extensions such as implementing callback methods for subscriptions and messages as well as creating an intermediary Rover proxy and HTTP interface. These additions have been explained in detail in the same appendices, while functions which aid in connection management have been described in Appendix A.2.

6.4 Example Application Scenario: V911

We present an example application scenario which demonstrates the major capabilities of the Rover architecture and a Rover server reference implementation. We have been working on an application scenario and its implementation to provide next generation 9-1-1 [31] capabilities which we call V911. We originally presented the ideas behind V911 as enhancing emergency response in Section 5.2.2 and we iterate through an example based on a fire situation on a university campus.



Value Meal

November 20 Lunch $6.29

Entree Dish

Lunch	Lightly Breaded Baked Chicken Breast with Lemon Citrus	Meat

Sides

Vegetable Basmati Rice
Portuguese Roll

College Park, MD
Today
41 F
Low: 34 F
High: 52 F
Clear
Date: 1 Dec 2009
Sunrise: 7:07 am
Sunset: 4:45 pm
Visibility: 10 mi Humidity: 57%
Pressure: steady 29.95 in
Wind-Chill: 36 F
Today Tomorrow

27496: Stamp Student Union (HH)

[✓ Favorite]

103 Campus Connector South	4	4:44 PM
100 Burtonsville Park _Ride	7	4:47 PM
114 University View	7	4:47 PM
120 Bowie Park and Ride	8	4:48 PM
104 College Park Metro	9	4:49 PM
104 College Park Metro	14	4:54 PM

Figure 15: Example campus services, including dining services, weather information, and real-time transportation updates.

6.4.1 Ecosystems and Entities in an V911 Emergency Response Scenario

In the scenario we present, two distinct Rover ecosystems which co-exist with each other. We outline the two ecosystems here, illustrated in Figure 16, with the entities within the individual ecosystems. Some entities exist in both ecosystems and we explain why they exist in both.

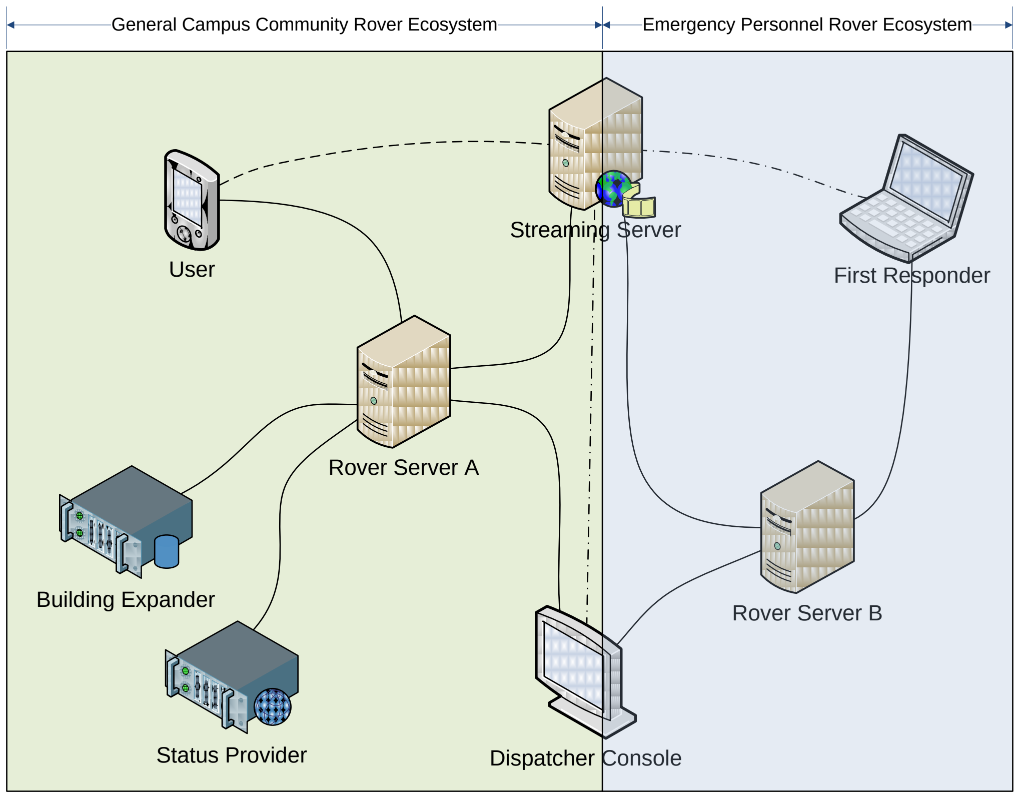
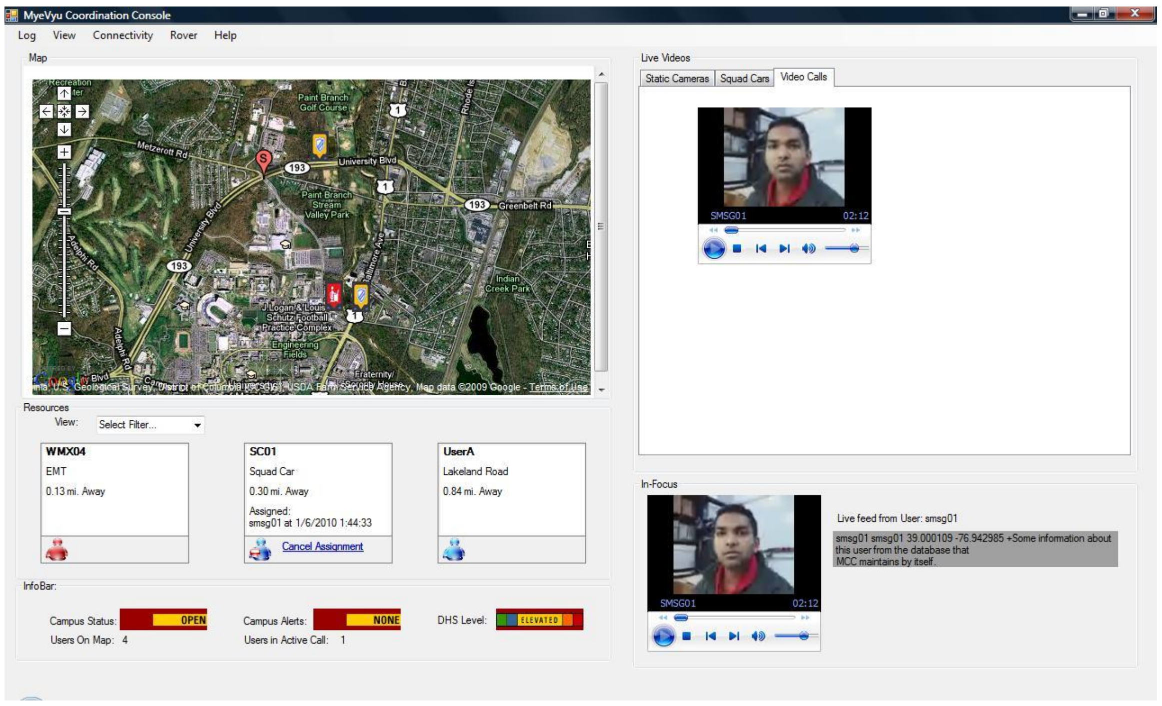


Figure 16: An overview of the two Rover ecosystems and the entities which reside within them.



View: Select Filter...

WMX04
EMT
0.13 mi. Away

SC01
Squad Car
0.30 mi. Away
Assigned:
smsg01 at 1/6/2010 1:44:33
Cancel Assignment

UserA
Lakeland Road
0.84 mi. Away

Campus Status: OPEN
Users On Map: 4
Campus Alerts: NONE
Users in Active Call: 1
DHS Level: ELEVATED

Live feed from User: smsg01
smsg01 smsg01 39.000109 -76.942985 +Some information about this user from the database that MCC maintains by itself.

Figure 17: Dispatcher Console screenshot with both Rover ecosystems incorporated into the display.

- *General Campus Community Rover Ecosystem*

  Entities in this ecosystem utilize the Rover server primarily to obtain campus related services, as originally described in Section 5.2.1. This includes retrieving dining menus, weather information, and real-time transportation updates, illustrated by screenshots in Figure 15. The entities in this system include, with the tiers they reside in:

  - *Rover Server A (Server Tier)*

    Every single entity connects to Rover Server A. This server provides all of the campus related services as well as the first point of contact when a member of the campus community needs an emergency service.

  - *User (User Tier, Utility Tier/Context Provider)*

    A User entity calls services from Rover Server A, can provide audio and video to a streaming server, and initiate emergency requests. User entities can be implemented on any number of platforms. To date, we have written User entity clients on the Nokia N810[25], Apple iPhone and iPod Touch (without streaming capabilities), and the Windows Mobile-enabled Samsung SGH-i907.

  - *Dispatch Console (User Tier, Utility Tier/Context Watcher)*

    The Dispatch Console acts as the Public Safety Answering Point (PSAP) by watching for context events to occur, obtaining status messages concerning campus, and acquire the appropriate audio and video streams from the User. Our implementation of the Dispatch Console can be seen in Figure 17, written using the Microsoft .NET platform.

[25] The Nokia N810 runs a Debian variant called maemo [74].

- *Streaming Server (Assistance Tier/Participating, Utility Tier/Service)*

  The Streaming Server receives and forwards streams, which may consist of audio and/or video. It also provides a service to Rover Server A to obtain locations to stream to for individual entities. We have utilized both VideoLAN [112] and LiveCast [62] to provide streaming capabilities.

- *Status Provider (Utility Tier/Context Provider)*

  The Status Service entity provides context to Rover Server A concerning the operating status of the campus.

- *Building Expander (Utility Tier/Context Watcher/Context Provider)*

  This context expander evaluates a building number and provides the latitude and longitude coordinates of the building.

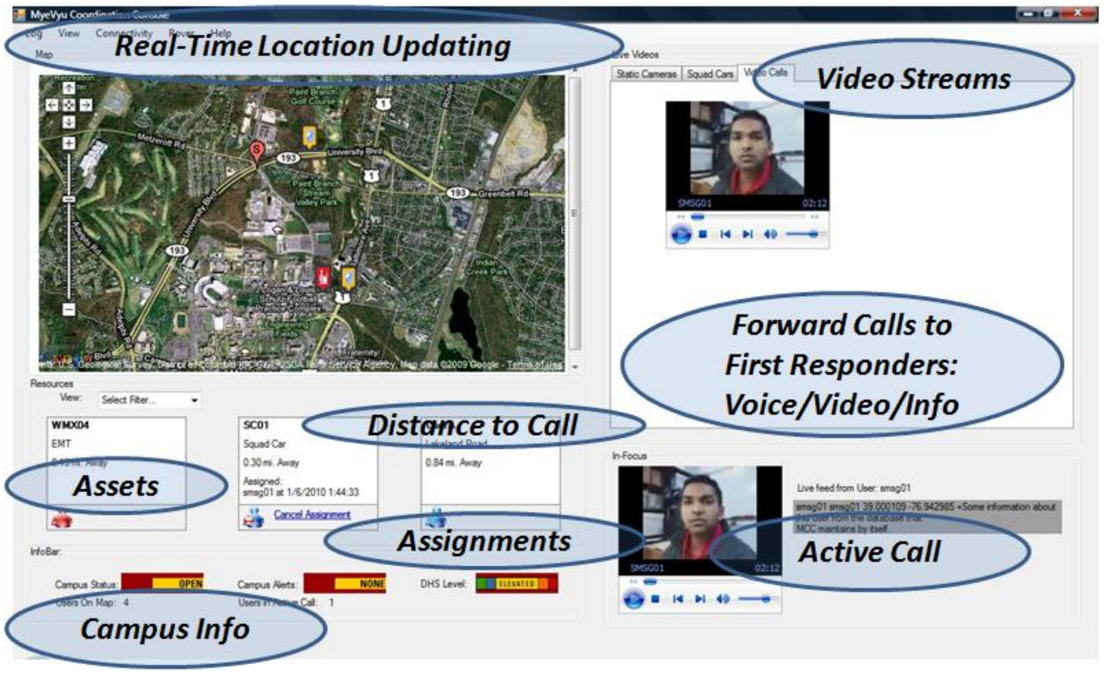


Figure 18: Annotated Dispatch Console screenshot.

- *Emergency Personnel Rover Ecosystem*

  Entities in this ecosystem assist User entities from the campus ecosystem in the event of an emergency. We list the entities involved below:

  - *Rover Server B (Server Tier)*

    Rover Server B communicates only with emergency personnel. This allows information items to be separated such that only the appropriate entities coexist on the same ecosystem.

  - *User (User Tier)*

    The User does not communicate directly with Rover Server B, but sends audio and video streams to the streaming server.

  - *Dispatch Console (User Tier, Utility Tier/Context Watcher)*

    In this ecosystem, the Dispatch Console examines the situation a User presents and decides the actions to take, including looking at the streams from the Streaming Server and the First Responder entities in the area. Figure 18 shows an annotated Dispatch Console.

  - *First Responder (User Tier)*

    First Responder entities indicate their current location and status. The Dispatch Console decides what First Responder entities to send to a scene and what information items to forward to them. First Responder entities can also obtain streams from the Streaming Server as necessary.

  - *Streaming Server (Assistance Tier/Participating, Utility Tier/Service)*

    The Streaming Server forwards audio and video streams from a User and static camera feeds to the Dispatch Console and the First Responder.

Three entities exist in all three ecosystems: the Dispatch Console, the Streaming Server, and the User. However, the Dispatch Console will be responsible for managing key context between the two ecosystems and coordinating key entities. Although this can be automated, the Dispatch Console represents man-in-the-loop operations. This shows that moving information items and context between two ecosystems can either be automated or driven by a user.

6.4.2 Fire Event Scenario

To illustrate how to use the Rover API, we present a detailed fire event situation with the two ecosystems outlined in the previous section. The entire flow of the scenario has been illustrated in a sequence diagram in Figure 19. We have shortened the API calls in this section and in the sequence diagram for readability reasons. We also assume that all entities connected to either Rover server have been connected using a direct TLS connection.

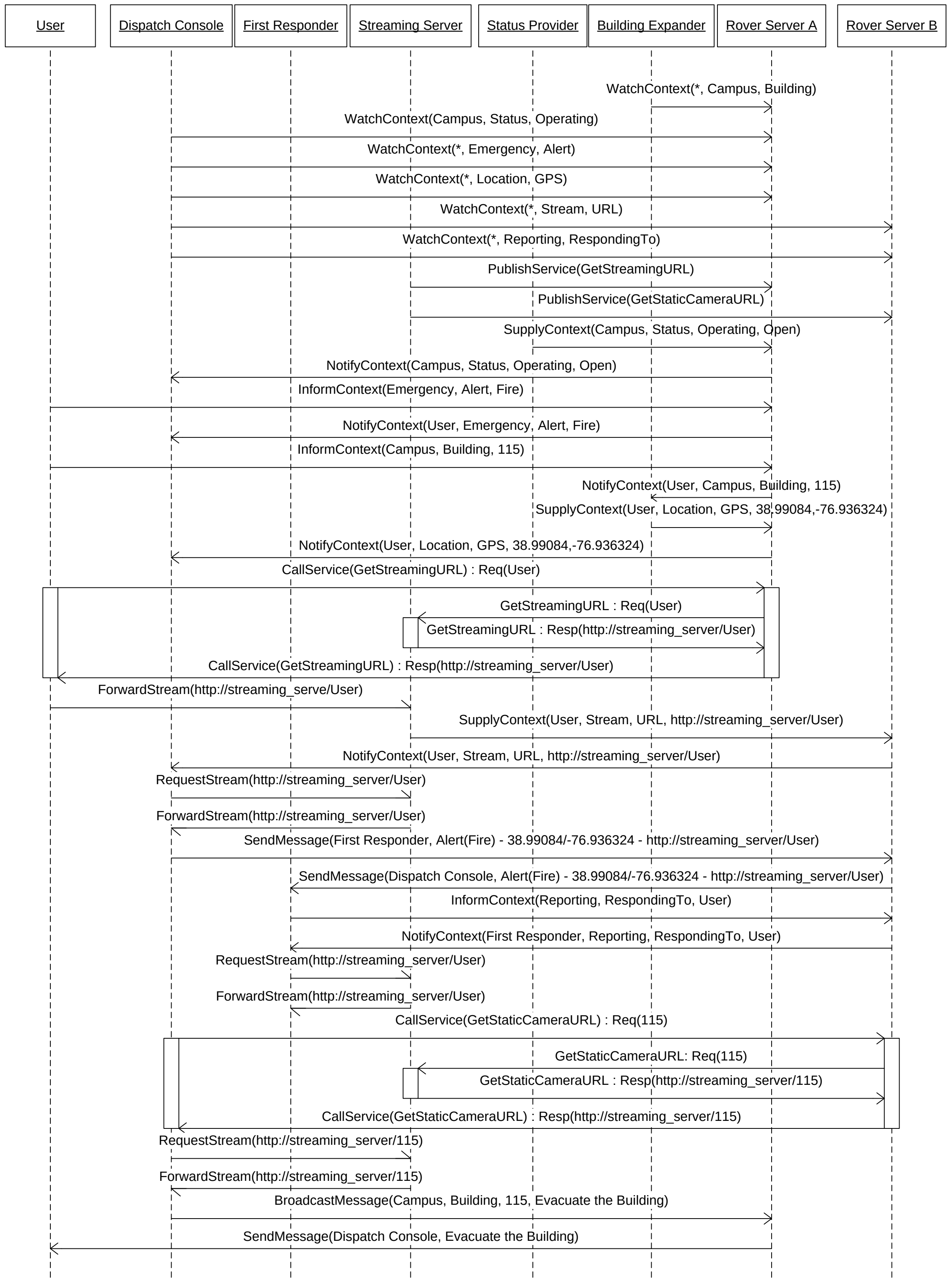

User
Dispatch Console
First Responder
Streaming Server
Status Provider
Building Expander
Rover Server A
Rover Server B
WatchContext(*, Campus, Building)
WatchContext(Campus, Status, Operating)
WatchContext(*, Emergency, Alert)
WatchContext(*, Location, GPS)
WatchContext(*, Stream, URL)
WatchContext(*, Reporting, RespondingTo)
PublishService(GetStreamingURL)
PublishService(GetStaticCameraURL)
SupplyContext(Campus, Status, Operating, Open)
NotifyContext(Campus, Status, Operating, Open)
InformContext(Emergency, Alert, Fire)
NotifyContext(User, Emergency, Alert, Fire)
InformContext(Campus, Building, 115)
NotifyContext(User, Campus, Building, 115)
SupplyContext(User, Location, GPS, 38.99084,-76.936324)
NotifyContext(User, Location, GPS, 38.99084,-76.936324)
CallService(GetStreamingURL) : Req(User)
GetStreamingURL : Req(User)
GetStreamingURL : Resp(http://streaming_server/User)
CallService(GetStreamingURL) : Resp(http://streaming_server/User)
ForwardStream(http://streaming_serve/User)
SupplyContext(User, Stream, URL, http://streaming_server/User)
NotifyContext(User, Stream, URL, http://streaming_server/User)
RequestStream(http://streaming_server/User)
ForwardStream(http://streaming_server/User)
SendMessage(First Responder, Alert(Fire) - 38.99084/-76.936324 - http://streaming_server/User)
SendMessage(Dispatch Console, Alert(Fire) - 38.99084/-76.936324 - http://streaming_server/User)
InformContext(Reporting, RespondingTo, User)
NotifyContext(First Responder, Reporting, RespondingTo, User)
RequestStream(http://streaming_server/User)
ForwardStream(http://streaming_server/User)
CallService(GetStaticCameraURL) : Req(115)
GetStaticCameraURL: Req(115)
GetStaticCameraURL : Resp(http://streaming_server/115)
CallService(GetStaticCameraURL) : Resp(http://streaming_server/115)
RequestStream(http://streaming_server/115)
ForwardStream(http://streaming_server/115)
BroadcastMessage(Campus, Building, 115, Evacuate the Building)
SendMessage(Dispatch Console, Evacuate the Building)

Figure 19: V911 Sequence Diagram of Events

Upon starting up both ecosystems, both Rover servers, and the entities involved in an emergency response begin watching context. The context watcher and provider Building Expander issues *WatchContext(*, Campus, Building)* to Rover Server A with the intent of expanding context giving building numbers to their latitude and longitude coordinates. The Dispatch Console, to Rover Server A, issues *WatchContext(Campus, Status, Operating)* to watch for changes in operating status of the campus, *WatchContext(*, Emergency, Alert)* to listen to User entities which issue an alert, and *WatchContext(*, Location, GPS)* to know the exact location of all entities on Rover Server A. The Dispatch Console also issues commands to Rover Server B: *WatchContext(*, Stream, URL)* to know when any audio and video streams start as well as *WatchContext(*, RespondingTo, Entity)* to receive notifications when First Responder entities indicate they will be responding to a scene.

Although now specified here, any campus services will issue a *PublishService* call to indicate the services they provide. The Streaming Server calls Rover Server A with *PublishService(GetStreamingURL)* indicates where a streaming client should send their streams to, and calls Rover Server B with *PublishService(GetStaticCameraURL)* which indicates the URL to obtain a stream from a static camera, for instance, at a building. This enables an ecosystem administrator the ability to ensure that streams can only be accessed by the appropriate entities.

The Status Service sends a *SupplyContext(Campus, Status, Operating, Open)* call to Rover Server A. This context will be forwarded to the Dispatch Console because of its subscription to *Campus/Status/Operating*.

The User primarily uses Rover Server A for the campus services, not shown in Figure 19. However, the User notices a small fire in the building they occupy. Instead of calling 9-1-1 or messaging a PSAP directly, the User issues an *InformContext(Campus, Emergency, Alert, Fire)* to the server, which immediately gets forwarded to the Dispatch Console as it has a subscription to all Alert messages. The User also transmits its building number (115), which will be received by the Building Expander. The Building Expander performs a lookup and sends the GPS coordinates to Rover Server A, which gets forwarded to the Dispatch Console so that the location of the possible fire will be displayed on a map.

The User communicates with Rover Server A to call the GetStreamingURL service offered by the Streaming Server. Rover Server A forwards the call to the Streaming Server and the Streaming Server sends the result back to the User via Rover Server A. When the User obtains the URL to stream to, the User begins streaming audio and video from their device. Upon receiving the stream, the Streaming Server issues a *SupplyContext(User, Stream, URL, http://streaming_server/User),* which gets forwarded to the Dispatch Console. The Dispatch Console then retrieves the stream for viewing.

Upon reviewing the situation, the person at the Dispatch Console decides to send a First Responder to the scene. After evaluating all of the First Responder entities (their roles and location, not shown in Figure 19), the Dispatch Console issues a *SendMessage* command to Rover Server B with all of the relevant data the First Responder selected should receive: the type of alert, the location to respond to, and a stream to look at. The First Responder sends a *InformContext(Reporting, RespondingTo, User)* to Rover Server

B, which forwards it to the Dispatch Console. The First Responder also retrieves the stream that it should look at.

Knowing the situation could be in an area wider than the User's specific location, the Dispatch Console retrieves the URL of the static camera for the building. The Dispatch Console calls the *GetStaticCameraURL* service served by the Streaming Server via Rover Server B. Upon getting the URL, the Dispatch Console views the stream and decides that the building should be evacuated. To do this, the Dispatch Console sends Rover Server A a *BroadcastMessage(Campus, Building, 115, Evacuate the Building)* message. All entities in this building receive this message. In Figure 19, we illustrate one such entity, the User who started this chain of events, receiving this message.

6.4.3 Discussion

The V911 scenario illustrates the utility of designing a first responder system using the Rover framework. Every single API call and context update is recorded by both Rover ecosystems in Rover Server A and Rover Server B. These calls are placed directly into the call history and the context provenance databases. By logging everything that goes through the Rover servers, the situation can be played back like a video recording by an auditor. The auditor can take a look at the sequence of events as they occurred when reviewing the situation for a report or to use as a future case study on this situation.

The presence of two Rover ecosystems separates unnecessary services and context from being exposed to parties that may not need it or have access to them. The dispatcher acts as a man-in-the-loop which allows information items and context to be abstracted away as necessary such that first responders only receive relevant context.

First responders need not sift through unnecessary information items and context as the dispatcher does this for them.

6.5 Summary

Chapter 5 addressed the design of a Rover ecosystem and corresponding Rover servers while this chapter addressed implementation issues that arise and how we implemented a Rover server. In particular, we described the goals of the implementation, specifics concerning the development environment of the Rover server, mobility support, authentication mechanisms, context handling, and database development. Additionally, we provided a full example on how multiple entities, which have different operating systems and form factors, interact with one another within different ecosystems.

Chapter 7

Concluding Remarks

Understanding how to effectively make use of information requires a thorough examination of context, specifically what context is and how to use it in our everyday lives. From a technology perspective, we, as a society, have are settled into the Information Age, but have found ourselves inundated with copious amounts of information. This requires so much of our attention, that we regularly spend time filtering out irrelevant information and hopefully not disregarding information that may be useful. Thus, we are well into the Attention Age.

Information includes not only messages and text, but also services and interaction between entities, including people. Messages, text, services, and interactions all have associated context with them. By exposing context in the design of a system allows users and developers to understand their information better. Thus, we are attempting to move out of the Attention Age into the Context-Aware Age, where context can help alleviate the amount of attention required to process information.

In this dissertation, we elaborated on definitions of context, the key aspects of context, and things we need to consider when designing a context-aware system. We the discussed our core requirements of a context-aware system, while taking into account the way people organizations operate within their own boundaries, when they need to communicate with organizations outside their scope, and when entities coexist on multiple organizations. To achieve this, we discussed the logical design of a Rover ecosystem and the entities it contains. As these organizations share data, messages, and services, we described the central communication point of a Rover ecosystem: the Rover server. The Rover server manages context, messages, and services in a uniform manner.

We continued our discussion by mapping the logical design of a Rover ecosystem and its central communication point, a Rover server, to an implementation which takes practical issues into consideration. We also took a look at a case study regarding an emergency response scenario in which using the Rover architecture would be beneficial.

7.1 Future Directions

We initially described research areas that we have not addressed in this dissertation in Section 1.4. These areas included usability issues, security issues, privacy concerns, issues dealing with dissonant information, providing misinformation through deliberate lying, and methods of reasoning. One of these should be addressed in the near term: authorization. The current Rover architecture does not have any built in authorization other than authenticating an entity to be connected to a Rover server and nothing else. The following list, albeit incomplete, of authorization mechanisms should be provided in future implementations: certain commands must be protected, specific context should only be read or updated only by privileged entities, restrictions on service calls[26], and restrictions on messages must be in place.

Another important aspect to look at would be how to handle context updates. In the current architecture, updates happen without regard to analyzing active context. Currently, only the function caller specifies whether or not to delete specific context entries. Like authorization, there must be mechanisms in place to determine whether or not context entries should be deleted based on their context. This also alleviates the

[26] Currently, services themselves can determine whether or not to allow a service call to go through or not.

function caller from having to specify the actions to take for each individual context entry, as they may have to evaluate the active context proactively.

We have designed the Rover architecture to be the core building to enable extensions, such as the ones described above, to be developed. Several components of Rover server can be interchanged with specialized components, such as a context mechanism, context inference engine, authorization schemes, communication protocols, and wire formats. This allows the flexibility of the Rover architecture to be utilized with ease for designers and developers.

Appendix A

Rover Server Reference Implementation Function List

The specification for the Rover Server application programming interface (API), as described in Section 5.5, has been completely implemented in our Rover server reference implementation. To account for any differences between the specification and using Python [86] with the Twisted programming framework for networking [108], we have added necessary functionality to bridge the gap between them. Additional functions have been created to support and facilitate the main Rover functionality. This appendix provides the complete function list documentation of the reference implementation.

Functions share objects between each other. These objects have been described in general in Sections 5.4.1 and 5.5.4. Described in Appendix A.1, common objects have the following documentation format:

Object Name:
    example object name (section, if part of the specification, with a description)
Description:
    Example description.
Contents:
    $variable_0$ – type
        Description.
    $variable_1$ – type
        Description.
    …
    $variable_n$ – type
        Description.

A variable can be a scalar value, a list, or a dictionary. A list can consist of either scalar values or common objects, but not both. If a list, the variable's description will begin with "a list of," followed by the type of scalar value or common object's name.

The API functions, described in Appendix A.2 through A.6, follow the documentation format below. If the function's implementation fulfills a specification function, we indicate the section where to find the definition. Availability shows if the function can be used with a particular Rover connection type. Should a function not be usable, the connection type will have a strikeout. We indicate the use of common objects in the description for parameters. The parameter can be a scalar value or a list.

Function Name:
example function name (section, if part of the specification)
Description:
Example description.
Availability:
Direct, Indirect-TCP, ~~Indirect-HTTP~~
Input Parameters:
$parameter_0$ – type
Description.
$parameter_1$ – type
Description.
…
$parameter_n$ – type
Description.
Output Parameters:
$parameter_0$ – type
Description.
$parameter_1$ – type
Description.
…
$parameter_n$ – type
Description.

Similar to the Rover server API functions, entities create callback functions to receive messages directly from the Rover server. Entities pass references to these callback functions to the Rover server. We specify these callback functions in a similar way as the API functions above, with the following modifications:

Callback Function Name:

example function name

Description:

Example description, including related functions which can be passed a reference to the callback object.

Input Parameters:

parameter$_0$ – type

Description.

parameter$_1$ – type

Description.

…

parameter$_n$ – type

Description.

Output Parameter:

parameter – type

Description.

A.1 Session API

Object Name:

a*vatar* (5.4.1 as *subject*)

Description:

Login information item concerning the entity connected to the Rover server.

Contents:

domain – a scalar string

The administrative domain of the user.

username – a scalar string

The identity of the user.

Object Name:

contextEntry (5.4.1 as *entry*)

Description:

An object which represents a context entry, typically stored on a Rover server.

Contents:

identifier – a scalar string

An identifier which uniquely identifies each *contextEntry* object on a specific Rover server.

entity – a scalar string or an *avatar* object

The subject of the context entry.

root – a scalar string

The root portion of the context predicate.

leaf – a scalar string
The leaf part of the context predicate.
value – a scalar string or a reference to something
The object of the context entry.
evidence – a scalar string or a reference to another *contextEntry*
Supports how the context entry has been derived.
creator – a scalar string or an *avatar* object
Who created the context entry.
timestamp – a scalar long or a datetime value
The relative or absolute time of when the Rover server created the context entry.

Object Name:
contextLeafDetail
Description:
An object which represents a portion of a context entry to be updated.
Contents:
leaf – a scalar string
The leaf part of the context predicate.
value – a scalar string or a reference to something
The object of the context entry.
evidence – a scalar string or a reference to another *contextEntry*
Supports how the context entry has been derived.

Object Name:
contextPredicate (5.4.1 as *leaf*)
Description:
An object containing only the context predicate portion of a context triple.
Contents:
root – a scalar string
The root portion of the context predicate.
leaf – a scalar string
The leaf part of the context predicate.

Object Name:
contextPredicateWithEntity
Description:
An object containing a context triple without the object element.
Contents:
entity – a scalar string or an *avatar* object
The subject of the context triple.
root – a scalar string
The root portion of the context predicate.

leaf – a scalar string

The leaf part of the context predicate.

Object Name:

contextPredicateWithValue

Description:

An object representing a context triple without the subject element.

Contents:

root – a scalar string

The root portion of the context predicate.

leaf – a scalar string

The leaf part of the context predicate.

value – a scalar string or reference to something

The object of the context entry.

Object Name:

contextTriple (5.4.1 as an analogy to a *triple*, but *leaf* instead of a *predicate*)

Description:

An object representing an entire context triple.

Contents:

entity – a scalar string or an *avatar* object

The subject of the context entry.

root – a scalar string

The root portion of the context predicate.

leaf – a scalar string

The leaf part of the context predicate.

value – a scalar string or reference to something

The object of the context entry.

Object Name:

serviceDescription

Description:

A five-tuple object describing a specific service, including the entity which runs the service. Any service can be run by multiple entities at different locations.

Contents:

name – a scalar string, identifies a particular service.

description – a scalar string

Describes a particular service and can also specify the input and output parameters for the service described.

requiredContext – a list of *contextPredicate* objects

Context concerning the entity calling the service that must be on the Rover server.

optionalContext – a list of *contextPredicate* objects

Context concerning the entity calling the service that will be forwarded from the Rover server if available, as the service will use the context to aid the service execution.

avatar – an *avatar* object

The service entity which will execute the service.

Object Name:

message

Description:

An encapsulated message that will be sent from one entity to another.

Contents:

sender – a scalar string or an *avatar* object

Identifies the sender of the *message*.

body – a scalar string

The actual message itself.

t*imestamp* – a scalar long or a datetime value

The moment the Rover server receives a message to forward or broadcast.

Object Name:

serviceIdentifier

Description:

A concatenation of a service name with the service entity which executes it.

Contents:

avatar – an *avatar* object

The service entity which will execute the service.

name – a scalar string

Identifies a particular service.

Object Name:

provenanceHistory (5.5.4 as *provenance*)

Description:

An information item containing the record of a context provenance entry.

Contents:

entity – a scalar string or an *avatar* object

The subject of the context entry.

root – a scalar string

The root portion of the context predicate.

leaf – a scalar string

The leaf part of the context predicate.

value – a scalar string or a reference to something

The object of the context entry.

evidence – a scalar string or a reference to another *contextEntry*

Supports how the context entry has been derived.

creator – a scalar string or an *avatar* object
Who created the context entry.
timestamp – a scalar long or a datetime value
Relative or absolute time when the Rover server created the context entry.
removed – a scalar Boolean
Whether or not the context entry has been removed from the active context when this history has been recorded.

Object Name:
callHistory (5.5.4 as *history*)
Description:
An information item containing the record of a call history entry.
Contents:
source – a scalar string or an *avatar* object
The entity which calls a function on the Rover server.
target – a scalar string or an *avatar* object
The entity which the function call concerns, potentially the caller itself.
meta – a scalar string
The type of function called, typically the section titles from Appendices A.2 to 0.
function – a scalar string
The name of the function called.
info – a scalar string
Can contain anything which concerns the function call.
request – a dictionary
Contains all of the information items in the parameters:
keys – the formal name of the parameter
values – the contents of the parameter
response – a scalar string
The response the Rover server sends back to the entity.
timestamp – a scalar long or a datetime value
The relative or absolute time of when the Rover server received the function call.
duration – a scalar long
The amount of time, in milliseconds, it took to execute the function. If you add the *timestamp* and the *duration* together, you get the finishing timestamp of the function call.

Object Name:
remotePerspective
Description:
A reference to code specific to Twisted which can be used to call remote functions on another entity's runtime environment. Used primarily for callbacks.
Contents:

remotePerspective – a scalar reference

Helper object which the Rover server can use to execute callbacks to a connected entity.

A.2 Session API

Function Name:

Avatar

Description:

Returns the domain and username of the connected entity.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

avatar – an *avatar* object

Contains the domain and username of the caller.

Function Name:

LocalTime

Description:

Returns the current date and time of the Rover server.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

datetime – a scalar long

The date and time of the Rover server, specified in milliseconds from the UNIX epoch.

Function Name:

Status

Description:

Returns an information item containing stored session data of the connected component on a Rover server.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

a*vatar* – an *avatar* object

Contains the domain and username of the caller.

messageCount – a scalar integer

Returns the number of messages held on the Rover server intended for delivery to the caller.

currentContext – a list of *contextEntry* objects

Contains the entire active context entries in the Rover server concerning the caller.

contextWatchList – a list of *contextTriple* objects

Contains the subscriptions the connected entity has for context it has an interest in.

A.3 Context API

Function Name:

SetupContextPerspective

Description:

Sets up or removes a *remotePerspective* to be used when a *contextEntry* arrives at a Rover server and a remote entity, connected via the Indirect-TCP interface, has a subscription to it.

Availability:

~~Direct~~, Indirect-TCP, ~~Indirect-HTTP~~

Input Parameters:

remotePerspective – a *remotePerspective* object

The reference to the callback object the Rover server should use when context an entity wants to know about arrives. If the reference has been set to null, then remove the existing *remotePerspective*, otherwise create or replace the current one.

Output Parameters:

isSetup – a scalar Boolean

Whether or not the *remotePerspective* has been added or removed successfully.

Function Name:

WatchContext (5.5.1)

Description:

Indicates to the Rover server that the connected entity wishes to watch a particular context predicate. If connected using the Direct interface, all *contextEntry* notifications will arrive via the *remotePerspective* reference provided. An entity may subscribe to the same *contextEntry* multiple times if a different *remotePerspective* has been setup for each subscription. If connected using the Indirect-TCP interface, notifications will first attempt to be sent via the *remotePerspective* provided by *SetupContextPerspective*. Should this attempt failed, the context will be stored until the entity calls *ObtainStoredContext*.

Connections via the Indirect-HTTP interface must always call *ObtainStoredContext* to retrieve any subscribed context notifications.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

contextPredicate – a *contextPredicate* object

Specifies the predicate to watch. If a particular content of the *contextPredicate* object has a wildcard character, then it will return all values for that particular part.

remotePerspective – a *remotePerspective* object

Only used by connections via the Direct interface, this indicates the reference of where to send all context updates to. If provided in an Indirect-TCP or Indirect-HTTP connection, this field will be ignored.

Output Parameters:

isSetup – a scalar Boolean

Whether or not the Rover server has been setup to actively watch for *contextPredicate* values on behalf of the function caller.

Function Name:

UnwatchContext (5.5.1)

Description:

Removes a particular context predicate from an entity's watch list. For connections via the Direct interface, the *remotePerspective* must be provided, otherwise, it must not be provided.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

contextPredicate – a *contextPredicate* object

Specifies the predicate to not watch.

remotePerspective – a *remotePerspective* object

Only used by connections via the Direct interface, this indicates the reference of where all context updates had been sent to. If provided in an Indirect-TCP or Indirect-HTTP connection, this field will be ignored.

Output Parameters:

isSetup – a scalar Boolean

Whether or not the Rover server no longer actively watches for *contextPredicate* values on behalf of the function caller.

Callback Function Name:

NotifyContext

Description:

A callback function passed to *SetupContextPerspective*, *WatchContext*, and *UnwatchContext*. Entities received pushed context entries through this function.

Input Parameters:

contextEntry – a *contextEntry* object

The entry to send to the entity.

Output Parameter:

None.

Function Name:

ObtainStoredContext (5.5.1)

Description:

Retrieves any context notifications stored on the Rover server that an entity has subscribed to and the Rover server has received.

Availability:

~~Direct~~, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

contextEntries – a list of *contextEntry* objects

The stored context entries received due to a context subscription.

Function Name:

InformContext (5.5.1)

Description:

Updates the calling entity's active context.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

root – a scalar string

The root portion of the context predicate.

contextLeafDetail – a list of *contextLeafDetail* objects

The specific leaves in the active context to update, related to the *root*.

pruneRoot – a scalar Boolean

Indicates whether or not to remove all active context with the same root prior to adding the context.

replaceLeaf – a scalar Boolean

Used to determine whether or not to replace a particular *contextEntry* if a similar one with the same *root* and *leaf* exist.

Output Parameters:

contextEntries – a list of *contextEntry* objects

The context which has successfully been updated on the Rover server.

Function Name:

SupplyContext (5.5.1)

Description:

Updates a specified entity's active context.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

entity – a scalar string or an *avatar* object

The subject of the context entry.

root – a scalar string

The root portion of the context predicate

contextLeafDetail – a list of *contextLeafDetail* objects

The specific leaves in the active context to update, related to the *root*.

pruneRoot – a scalar Boolean

Indicates whether or not to remove all active context with the same root prior to adding the context

replaceLeaf – a scalar Boolean

Used to determine whether or not to replace a particular *contextEntry* if a similar one with the same *root* and *leaf* exist.

Output Parameters:

contextEntries – a list of *contextEntry* objects

The context which has successfully been updated on the Rover server.

Function Name:

ObtainContext (5.5.1)

Description:

Queries the active context and filters result by specified entities, roots, and complete context predicates.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

contextPredicateWithEntities – a list of *contextPredicateWithEntity* object

The filtering constraints. To ignore a particular constraint of the context triple, use a wildcard.

Output Parameters:

contextEntries – a list of *contextEntry* objects

The results of the query. Returns an empty list if no *contextEntry* objects have been found.

Function Name:

ObtainContextByReference (5.5.1)

Description:

Queries the active context by a reference identifier.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

identifier – a scalar string

An identifier which uniquely identifies each *contextEntry* object on a specific Rover server.

Output Parameters:

contextEntry – a *contextEntry* object

The results of the query. Returns *null* if an object does not have the specified *identifier*.

A.4 Service API

Function Name:

PublishService (5.5.2)

Description:

Informs a Rover server that the caller can execute a particular service. Each individual service name pairs with the avatar of the service. The service entity must be a Direct connection to the Rover server. When the connection breaks, the service will be automatically removed from being callable.

Availability:

Direct, ~~Indirect-TCP~~, ~~Indirect-HTTP~~

Input Parameters:

serviceDescriptions – a list of *serviceDescription* objects

Contains all of the services to be published.

remotePerspective – a *remotePerspective* object

The interface the Rover server calls when an entity wishes to call a particular service.

Output Parameters:

serviceDescriptions – a list of *serviceDescription* objects

Contains all of the services published successfully.

Function Name:

UnpublishService (5.5.2)

Description:

Removes a service from being published from a Rover server. The original connection must call this for the function to complete properly.

Availability:

Direct, ~~Indirect-TCP~~, ~~Indirect-HTTP~~

Input Parameters:

serviceDescriptions – a list of *serviceDescription* objects

Contains all of the services to be removed from publication.

Output Parameters:

serviceDescriptions – a list of *serviceDescription* objects

Contains all of the services removed from publication.

Callback Function Name:

Service

Description:

A callback function passed to *PublishService and UnpublishService*, calls to a particular service will be forwarded to this function. The callback function name should only be considered a placeholder, as the actual implementation must have the service name in it.

Input Parameters:

entity – a scalar string or an *avatar* object

Indicates who called the service.

data – a scalar string

The request information item to the service. The *data* will not be used directly by the Rover server except in storage to the call history database.

contextEntries – a list of *contextEntry* objects

The complete list of context entries concerning the calling *entity*.

Output Parameter:

data – a scalar value, the response information item of the service. If an error occurs during execution, the callback function returns an *errorMessage* instead of *data*.

errorMessage – a scalar string, an information item that indicates the reason the call to the service did not succeed.

Function Name:

ListAllServices (5.5.2)

Description:

Retrieves a list of all services published and available on the Rover server.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

serviceDescriptions – a list of *serviceDescription* objects

A list of all published services on the Rover server *ListAllServices* executed on.

Function Name:

ListRelevantServices (5.5.2)

Description:

Retrieves a list of services that can be called by an entity based on the entity's active context on the Rover server.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

serviceDescriptions – a list of *serviceDescription* objects

A list of published services that can be called based on the calling entity's active context.

Function Name:

CallService (5.5.2)

Description:

Calls a particular entity's service. If the service cannot be called for any reason, such as the calling entity not having the appropriate active context, the caller will be notified that the service cannot be called. Otherwise, it will return the results of the call.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

serviceIdentifier – a *serviceIdentifier* object

Indicates whom to direct the service call to and the service to call.

data – a scalar string

An information item that acts as the request data to the service that will be called. The data specification can be written explicitly in a *serviceDescription* object.

Output Parameters:

data – a scalar string

An information item that acts as the response data to the service that has been called. The data specification can be written explicitly in a *serviceDescription* object. The *data* scalar will be provided if the service has been called successfully, otherwise an *errorMessage* scalar will be given instead.

errorMessage – a scalar string

Information item indicating the reason the service did not run successfully.

A.5 Message API

Function Name:

SetupMessagePerspective

Description:

Sets the callback where messages should be forwarded to. This represents an entity that will be directed messages. If an entity provides a valid callback, the Rover server will automatically forwarded all stored messages to the entity such that the entity does not need to call *GetMessages*.

Availability:

Direct, Indirect-TCP, ~~Indirect-HTTP~~

Input Parameters:

remotePerspective – a *remotePerspective* object

The interface that will be used when forwarding messages. If it already exists, the specified *remotePerspective* will replace the previous one.

Output Parameters:

isSetup – a scalar Boolean value

Indicates whether or not the *remotePerspective* has been setup successfully or not.

Function Name:

RemoveMessagePerspective

Description:

Removes the callback for messages of the entity which called the function.

Availability:

Direct, Indirect-TCP, ~~Indirect-HTTP~~

Input Parameters:

None.

Output Parameters:

isSetup – a scalar Boolean value

Indicates whether or not the *remotePerspective* specified in *SetupMessagePerspective* has been removed successfully.

Callback Function Name:

InformMessage (5.5.3)

Description:

Callback function to receive all pushed messages from the Rover server, set by calling the *SetupMessagePerspective* function.

Input Parameters:

m*essage* – a *message* object

The message that has been pushed from the Rover server. Contains all necessary information items to discern who sent the message, what the message contains, and the reception time.

Output Parameter:

None.

Function Name:

GetMessages

Description:

Retrieves all stored messages on the Rover server intended for the calling entity.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

None.

Output Parameters:

messages – a list of *message* objects

The stored *message* objects on the Rover server for a particular entity.

Function Name:
SendMessage (5.5.3)
Description:
Sends a directed message to a specific entity.
Availability:
Direct, Indirect-TCP, Indirect-HTTP
Input Parameters:
target – a scalar string or an *avatar* object
The intended recipient of the message.
body – a scalar string
The contents of the message.
Output Parameters:
isAccepted – a scalar Boolean
Whether or not the message has been accepted for delivery.

Function Name:
BroadcastMessage (5.5.3)
Description:
Broadcasts a message based on the active context.
Availability:
Direct, Indirect-TCP, Indirect-HTTP
Input Parameters:
contextPredicateWithValues – a list of *contextPredicateWithValue* objects
If a particular entity satisfies the given constraints, the entity will receive the message sent.
body – a scalar string
The contents of the message.
Output Parameters:
isAccepted – a scalar Boolean
Whether or not the message has been accepted for delivery.

A.6 History API

Function Name:
ObtainContextProvenance (5.5.4)
Description:
Returns a list of context provenance entries from the database that matches the given parameters.
Availability:
Direct, Indirect-TCP, Indirect-HTTP
Input Parameters:

entity – a scalar string or an *avatar* object
Filter by subject.
root – a scalar string
Filter by the root portion of the context predicate.
start – a scalar long
The earliest entry to obtain, in milliseconds, from the Unix epoch.
finish – a scalar long
Represents the latest entry to obtain, in milliseconds, from the Unix epoch.

Output Parameters:

provenanceHistories – a list of *provenanceHistory* objects
The list of entries which match the parameters given.

Function Name:

ObtainCallHistory (5.5.4)

Description:

Returns a list of call history entries from the database matching the parameters.

Availability:

Direct, Indirect-TCP, Indirect-HTTP

Input Parameters:

source – a scalar string or an *avatar* object
Filter by the original callers.
target – a scalar string or an *avatar* object
Filter by the intended destination of the function calls.
meta – a scalar string
The name of the group of the functions to filter by.
function – a scalar string
The name of the function to filter by.
start – a scalar long
The earliest entry to obtain, in milliseconds, from the Unix epoch.
finish – a scalar long
Represents the latest entry to obtain, in milliseconds, from the Unix epoch.

Output Parameters:

callHistories – a list of *callHistory* objects
The list of entries which match the parameters given.

References

[1] Gregory D. Abowd et al., "Cyberguide: A Mobile Context-Aware Tour Guide," *Wireless Networks*, vol. 3, pp. 421-433, 1997.

[2] Christian B. Almazan, Moustafa Youssef, and Ashok K. Agrawala, "Rover: An Integration and Fusion Platform to Enhance Situational Awareness," *First International Workshop on Research Challenges in Next Generation Networks for First Responders and Critical Infrastructures (NetCri'07)*, pp. 582-587, April 2007.

[3] Apple. (2010, March) iPhone Dev Center. [Online]. http://developer.apple.com/iphone/

[4] Mie Augier, Syed Z. Shariq, and Morten Thanning Vendelø, "Understanding Context," *Journal of Knowledge Management*, vol. 5, no. 2, pp. 125-136, 2001.

[5] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11-33, January-March 2004.

[6] Matthias Baldauf, Schahrm Dustdar, and Florian Rosenberg, "A Survey on Context-Aware Systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263-277, 2007.

[7] Suman Banerjee et al., "Rover: Scalable Location-Aware Computing," vol. 35, no. 10, pp. 46-53, October 2002.

[8] Louise Barkhuus, "Context Information vs. Sensor Information: A Model for

Categorizing Context in Context-Aware Mobile Computing," *Symposium on Collaborative Technologies and Systems*, pp. 127-133, 2003.

[9] Louise Barkhuus and Anind Dey, "Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Explained," *UbiComp 2003*, vol. LNCS 2864, pp. 149-156, 2003.

[10] Mary Bazire and Patrick Brézillon, "Understanding Context Before Using It," *CONTEXT 2005, LNAI 3354*, pp. 29-40, 2005.

[11] John Bell, "Against 'measurement'," *Physics World*, pp. 33-40, August 1990.

[12] Martin Bichler and Kwei-Jay Lin, "Service-Oriented Computing," *Computer*, vol. 39, no. 3, March 2006.

[13] Ian Bicking. (2009) SQLObject. [Online]. http://sqlobject.org/SQLObject.html

[14] Bluetooth Special Interest Group, "Specification Volume 0: Specification of the Bluetooth System," Core Package 3.0 + HS, 2009.

[15] P.J. Brown, "The Stick-E Document: A Framework for Creating Context-Aware Applications," *Electronic Publishing*, vol. 8, no. 2-3, pp. 259-272, June/September 1995.

[16] Guanling Chen and David Kotz, "A Survey of Context-Aware Mobile Computing Research," Computer Science Department, Dartmouth Computer Science Technical Report TR2000-381 2000.

[17] Timothy Colburn, "Methodology of Computer Science," in *The Blackwell Guide to the Philosophy of Computing and Information*, Luciano Floridi, Ed. Malden: Blackwell Publishing, 2004, ch. 24, pp. 318-326.

[18] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan, "Context is Key," *Communications of the ACM*, vol. 48, no. 3, pp. 49-53, March 2005.

[19] Douglas Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," The Internet Society, RFC 4627, 2006.

[20] Thomas H. Davenport and John C. Beck, *The Attention Economy: Understanding the New Currency of Business*.: Harvard Business Press, 2002.

[21] Debian. (2010, February) Debian -- The Universal Operating System. [Online]. http://www.debian.org/

[22] Bella M. DePaulo, Deborah A. Kashy, Susan E. Kirkendol, Melissa M. Wyer, and Jennifer A. Epstein, "Lying in Everyday Life," *Journal of Personality and Social Psychology*, vol. 70, no. 5, pp. 979-995, 1996.

[23] Anind K. Dey, "Providing Architectural Support for Building Context-Aware Applications," Georgia Institute of Technology, Thesis for the Doctor of Philosophy 2000.

[24] Anind K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, pp. 4-7, 2001.

[25] Dictionary.com. (2009) Dictionary.com Unabridged (v 1.1). [Online]. http://dictionary.reference.com/browse/abstraction

[26] Priscilla A. Drum and Bonnie C. Konopak, "Learning Word Meanings from Context," in *The Nature of Vocabulary Acquisition*, Margaret G. McKeown and Mary E. Curtis, Eds.: Lawrence Erlbaum Associates, 1987, ch. 5, pp. 73-87.

[27] Sastry Duri, Alan Cole, Jonathan Munson, and Jim Christensen, "An Approach to

Providing a Seamless End-User Experience for Location-Aware Applications," *WMC 2001*, pp. 20-25, 2001.

[28] Michael Engelhardt and Thomas C. Schmidt, "Semantic Linking - a Context-Based Approach to Interactivity in Hypermedia," *CoRR*, 2004.

[29] Jérôme Euzenat and Pavel Shvaiko, *Ontology Matching*. Berlin: Springer-Verlag, 2007.

[30] Facebook. (2010, January) Facebook. [Online]. http://www.facebook.com/

[31] Federal Communications Commission: Public Safety and Homeland Security Bureau. (2010, January) 9-1-1 Service. [Online]. http://www.fcc.gov/pshs/services/911-services/

[32] Federal Communications Commission: Public Safety and Homeland Security Bureau. (2009, December) Enhanced 9-1-1 - Wireless Services. [Online]. http://www.fcc.gov/pshs/services/911-services/enhanced911/reports.html

[33] Christiane Fellbaum, Ed., *WordNet: An Electronic Lexical Database*.: MIT Press, 1998.

[34] Roy T. Fielding et al., "Hypertext Transfer Protocol -- HTTP/1.1," Internet Engineering Task Force, RFC2616,.

[35] Andrew U. Frank, "Ontology for Spatio-temporal Databases," in *Spatio-temporal Databases*.: Springer-Verlag, 2003, ch. 2, pp. 9-77.

[36] Google. (2010, March) Android Developers. [Online]. http://developer.android.com/

[37] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, *Java Language*

Specification, 3rd ed.: Addison-Wesley Professional, 2005.

[38] Claudio Gutiérrez, Carlos A. Hurtado, and Alejandro A. Vaisman, "Temporal RDF," *Second European Semantic Web Conference (ESWC'05)*, pp. 93-107, 2005.

[39] Douglas Harper. (2001) Online Etymology Dictionary. [Online]. http://www.etymonline.com/index.php?term=context

[40] Lonnie Harvel et al., "Context Cube: Flexible and Effective Manipulation of Sensed Context Data," *Pervasive Computing*, vol. LNCS 3001, pp. 51-68, 2004.

[41] Mike Hazas, James Scott, and John Krumm, "Location-Aware Computing Comes of Age," *Computing*, vol. 37, no. 2, pp. 95-97, February 2004.

[42] Jeffrey Hightower and Gaetano Borriello, "A Survey and Taxonomy of Location Systems for Ubiquitous Computing," University of Washington, Technical Report UW-CSE 01-08-03, 2001.

[43] Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello, "The Location Stack: A Layered Model for Location in Ubiquitous Computing," *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pp. 22-28, June 2002.

[44] Pascsal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, "OWL 2 Web Ontology Language Primer," World Wide Web Consortium, W3C Recommendation REC-owl2-primer-20091027, 2009.

[45] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*, 5th ed.: Springer-Verlag, 2004.

[46] Denis Howe. (1993) Free On-Line Dictionary of Computing. [Online]. http://foldoc.org/context

[47] Denis Howe. (1996, December) Free On-Line Dictionary of Computing. [Online]. http://foldoc.org/context+switch

[48] Denis Howe. (1999, September) Free On-Line Dictionary of Computing. [Online]. http://foldoc.org/context-sensitive+menu

[49] Hwaci. (2010, January) SQLite. [Online]. http://www.sqlite.org/

[50] Jadwiga Indulska and Peter Sutton, "Location Management in Pervasive Systems," *Proceedings of the Australiasian Information Security Workshop*, pp. 143-151, 2003.

[51] International Organization for Standardization, "Data Elements and Interchange Formats -- Information Interchange -- Representation of Dates and Times," ISO 8601:2004, 2004.

[52] Internet Archive. (2010, January) Internet Archive. [Online]. http://www.archive.org/

[53] Kristian Ellebæk Kjær, "A Survey of Context-Aware Middleware," *Proceedings of Software Engineering*, 2007.

[54] Kimberle Koile, Konrad Tollmar, David Demirdjian, Howard Shrobe, and Trevor Darrell, "Activity Zones for Context-Aware Computing," *UbiComp 2003*, vol. LNCS 2864, pp. 90-106, 2003.

[55] Kenneth L. Kraemer, Jason Dedrick, and Prakul Sharma, "One Laptop Per Child: Vision vs. Reality," *Communications of the ACM*, vol. 52, no. 6, pp. 66-73, June 2009.

[56] Jeff Kramer, "Is Abstraction the Key to Computing?," *Communications of the*

ACM, vol. 50, no. 4, pp. 37-42, April 2007.

[57] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, 1978.

[58] Barry M. Leiner et al., "A Brief History of the Internet," vol. 39, no. 5, October 2009.

[59] Ted Lewis, "Living in real time, side A (What is the Info Age?)," *Computer*, vol. 28, no. 9, pp. 8-10, September 1995.

[60] H. Lieberman and T. Selker, "Out of Context: Computer Systesm that Adapt to, and Learn from, Context," *IBM Systems Journal*, vol. 39, no. 3-4, pp. 617-732, July 2000.

[61] Linux Kernel Organization. (2008, February) The Linux Kernel Archives. [Online]. http://www.kernel.org/

[62] LiveCast Medi. (2010, March) LiveCast. [Online]. http://www.livecast.com/

[63] Peter Lyman and Hal R. Varian. (2003) How Much Information. [Online]. http://www.sims.berkeley.edu/how-much-info-2003

[64] Natalia Marmasse and Chris Schmandt, "Location-Aware Information Delivery with ComMotion," *HUC 2000*, vol. LCS 1927, pp. 157-171, 2000.

[65] Dave McComb, "Semantics in Business Systems," 2004.

[66] Merriam-Webster. (2009) Merriam-Webster Online Dictionary. [Online]. http://www.merriam-webster.com/dictionary/context

[67] David Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," Internet Engineering Task Force, RFC 2030, 1996.

[68] Minnesota Burea of Criminal Apprehension. (2010, January) Minnesota Crime Alert Network. [Online]. http://www.crimealert.state.mn.us/amberalert/AA_ActiveAlerts.asp

[69] Makoto Miyake, "A Survey on Progress and Standardization Trends in Wireless Communications," *Journal of Communications*, vol. 4, no. 7, pp. 509-520, August 2009.

[70] National Aeronautics and Space Administration, "Mars Climate Orbiter: Mishap Investigation Board: Phase I Report," 1999.

[71] Giles John Nelson, "Context-Aware and Location Systems," University of Cambridge, Clare College, PhD Dissertation 1998.

[72] New Vision Television - KIMT. (2010, January) 3kimt.com - Iowa, Minnesota, Together. [Online]. http://www.kimt.com/mostpopular/story/Bogus-Amber-Alert-Message/M7qSdGuz4kiiMt0gYelSuQ.cspx

[73] Annu-Maaria Nivala and L. Tiina Sarjakoski, "Need for Context-Aware Topographic Maps in Mobile Devices," *ScanGIS'2003*, pp. 15-29, 2003.

[74] Nokia. (2005) maemo. [Online]. http://maemo.org/

[75] Donald A. Norman, "The Way I See It: Signifiers, Not Affordances," *interactions*, vol. 15, no. 6, pp. 18-19, November-December 2008.

[76] Object Management Group, "OMG Unified Modeling Language (OMG UML), Infrastructure," formal/2009-02-04, 2009.

[77] Palm. (2010, March) Palm Developer Center. [Online]. http://developer.palm.com/

[78] Jason Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers,"

pp. 92-99, 1998.

[79] Charles Petzold, *Code: The Hidden Language of Computer Hardware and Software*. Redmond: Microsoft Press, 1999.

[80] Joseph Phelps, Glen Nowak, and Elizabeth Ferrell, "Privacy Concerns and Consumer Willingness to Provide Personal Information," *Journal of Public Policy & Marketing*, vol. 19, no. 1, pp. 27-41, Spring 2000.

[81] David S. Platt, *Introducing Microsoft.NET*, 2nd ed. Redmond, Washington: Microsoft Press, 2002.

[82] Jon Postel, "Transmission Control Protocol," Internet Engineering Task Force, RFC 793, 1981.

[83] Jon Postel, "User Datagram Protocol," Internet Engineering Task Force, RFC 768, 1980.

[84] Michael Pressley, Joel R. Levin, and Mark A. McDaniel, "Mnemonic and Contextul Approaches," in *The Nature of Vocabulary Acquisition*, Margaret G. McKeown and Mary E. Curtis, Eds.: Lawrence Erlbaum Associates, 1987, ch. 7, pp. 107-127.

[85] Hung Keng Pung et al., "Context-Aware Middleware for Pervasive Elderly Homecare," *IEEE Journal on Selected Areas in Communication*, vol. 27, no. 9, pp. 510-524, May 2009.

[86] Python Software Foundation, "The Python Language Reference," 2010.

[87] Python Software Foundation, "The Python Standard Library," 2010.

[88] Red Hat. (2008, May) Fedora Project. [Online]. http://fedoraproject.org/

[89] Tom Rodden, Keith Chervest, Nigel Davies, and Alan Dix, "Exploiting Context in HCI Design for Mobile Systems," *Workshop on Human Computer Interaction with Mobile Devices*, 1998.

[90] Bill Schilit, Norman Adams, and Roy Want, "Context-Aware Computing Applications," *First International Workshop on Mobile Computing Systems and Applications*, pp. 85-90, 1994.

[91] Bill N. Schilit and Marvin M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network*, pp. 22-32, September/October 1994.

[92] Albrecht Schmidt et al., "Advanced Interaction in Context," *First International Symposium on Handheld and Ubiquitous Computing (HUC99)*, vol. LNCS 1707, pp. 89-101, 1999.

[93] H. Schulzrinne, V. Gurbani, P. Kyzivat, and J. Rosenberg, "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)," RFC 4480, 2006.

[94] H. Schulzrinne and H. Tschofenig, "Location Types Registry," RFC 4589, 2006.

[95] Henning Schulzrinne, Hannes Tschofenig, John B. Jr. Morris, Jorge R. Cuellar, and James Polk, "Geolocation Policy: A Dcoument Format for Expressing Privacy Policies for Location Information," Internet Engineering Task Force, Internet-Draft draft-ietf-geopriv-policy-21, 2009.

[96] David A. Schum, *The Evidential Foundations of Probabilistic Reasoning*. Evanston: Northwestern University Press, 1994.

[97] Claude E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379-423, 623-656, July, October 1948.

[98] Ben Shneiderman, *Designing the User Interface*.: Addison-Wesley, 1998.

[99] Ben Shneiderman, "Universal Usability: Pushing Human-Computer Interaction Research to Empower Every Citizen," University of Maryland, College Park, College Park, HCIL Technical Report Number 99-17, 1999.

[100] Biplav Srivastava and Jana Koehler, "Web Service Composition - Current Solutions and Open Problems," *Workshop on Planning for Web Services (ICAPS'03)*, pp. 28-35, 2003.

[101] Robert J. Sternberg, "Most Vocabulary is Learned from Context," in *The Nature of Vocabulary Acquisition*, Margaret G. McKeown and Mary E. Curtis, Eds.: Lawrence Erlbaum Associates, 1987, ch. 6, pp. 89-105.

[102] Thomas Strang and Claudia Linnhoff-Popien, "A Context Modeling Survey," *Workshop on Advanced Context Modeling, Reasoning, and Management*, 2004.

[103] H. Sugano et al., "Presence Information Data Format (PIDF)," RFC 3863, 2004.

[104] Telecommunication Standardization Sector of the International Telecommunication Union, "Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model," International Telecommunication Union, ITU-T Recommendation X.200, 1994.

[105] Christopher Gordon Timpson, "Quantum Information Theory and the Foundations of Quantum Mechanics," University of Oxford: The Queen's College, Thesis for the Doctor of Philosophy 2004.

[106] Alvin Toffler, *Future Shock*.: Random House, 1970.

[107] Itamar Turner-Trauring. (2003, December) TwistedJava: Perspective Broker for

Java. [Online]. http://itamarst.org/software/twistedjava/

[108] Twisted Laboratories, "Twisted Documentation," 2009.

[109] Twitter. (2010, January) Twitter. [Online]. http://www.twitter.com/

[110] United States Department of Justice, "National Information Exchange Model (NIEM) User Guide," 2007.

[111] Hal R. Varian, "Universal Access to Information," *Communications of the ACM*, vol. 48, no. 10, 2005.

[112] VideoLAN. (2008, February) VideoLAN - VLC Media Player. [Online]. http://www.videolan.org/

[113] Hans Christian von Baeyer, *Information: The New Language of Science*. Cambridge, Massachusettes: Harvard University Press, 2003.

[114] Mark Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94-95, 98-102, 104, September 1991.

[115] Mark Weiser, "The World is Not a Desktop," *Interactions*, vol. 1, no. 1, pp. 7-8, January 1994.

[116] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fifth Edition," REC-xml-20081126, 2008.

[117] World Wide Web Consortium, "RDF Primer," REC-rdf-primer-20040210, 2004.

[118] World Wide Web Consortium, "XML Schema Part 0: Primer Second Edition," REC-xmlschema-0-20041028, 2004.

[119] Kurt Zeilenga, "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map," The Internet Society, RFC 4510, 2006.

[120] ZigBee Alliance, "ZigBee Specification," 053474r17, 2008.

[121] Andreas Zimmerman, Andreas Lorenz, and Reinhard Oppermann, "An Operational Definition of Context," *LNCS 4635/2007*, pp. 558-571, 2007.