Abstract

Title of dissertation: ROVER-II: A CONTEXT-AWARE MIDDLEWARE FOR PERVASIVE COMPUTING ENVIRONMENTS

Shivsubramani Krishnamoorthy
Doctor of Philosophy, 2012

Thesis directed by: Professor Ashok Agrawala
Department of Computer Science

It is well recognized that context plays a significant role in all human endeavors. All decisions are based on information which has to be interpreted in context. By making information systems context-aware we can have systems that significantly enhance human capabilities to make critical decisions.

A major challenge of context-aware systems is to balance usability with generality and extensibility. The relevant context changes depending on the particular application. The model used to represent the context and its relationship to entities must be general enough to allow additions of context categories without redesign while remaining usable across many applications. Also, while efforts are put in by application designers and developers to make applications *context-aware*, these efforts are customized to specific needs of the target application, and only certain common contexts like location and time are taken into account. Therefore, a general framework is called for that can (i) efficiently maintain, represent and integrate contextual information, (ii) act as an integration platform

where different applications can share contexts and (iii) provide relevant services to make efficient use of the contextual information. This dissertation presents:

- a generic and effective context model - *Rover Context Model (RoCoM)* that is structured around four primitives: entities, events, relationships, and activities; and practically usable through the concept of templates
- a flexible, extensible and generic ontology - *Rover Context Model Ontology (RoCoMO)* supporting the model, that addresses the shortcomings of existing ontologies,
- an effective mechanism of modeling the context of a situation, through the concept of *relevant context*, with the help of *situation graph*, efficiently handling and making best use of context information,
- a context middleware - Rover-II, which serves as a framework for contextual information integration, that could be used not just to store and compile the contextual information, but also integrate relevant services to enhance the context information; and more importantly, enable sharing of context among the applications subscribed to it,
- the initial design and implementation of a distributed architecture for Rover-II, following a P2P arrangement inspired from Tapestry [149].

The above concepts are illustrated through *M-Urgency*, a context-aware public safety system that has been deployed at the University of Maryland Police Department.

ROVER-II: A CONTEXT-AWARE MIDDLEWARE FOR PERVASIVE COMPUTING ENVIRONMENTS

by

Shivsubramani Krishnamoorthy

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Advisory Committee:
Professor Ashok Agrawala, Chair/Advisor
Professor Inderjit Chopra, Dean's Representative
Professor Atif Memon
Professor Aravind Srinivasan
Professor Bharat Jayaraman

Dedication

Dedicated to my most beloved ***Amma***, my Guiding Light.

Also dedicated to my parents and sisters.

Acknowledgments

There have been many who have inspired me, influenced me, guided me, advised me in various stages of my life. I owe my gratitude to all the people who have helped me mould myself into who I am today. I sincerely thank those who made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to express my heartfelt gratitude to my advisor, Professor Ashok Agrawala for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past four years. He had always made himself available to help me with the minutest of the problems, both academically and personally. I consider myself very privileged to have worked with a person of this stature.

I would also like to thank Dr. Atif Memon, Dr. Aravind Srinivasan, Dr. Inderjit Chopra and Dr. Bharat Jayaraman for agreeing to serve in my dissertation committee. I truly value their suggestions and advice which helped make my work stronger. I owe it to them for their continual support and guidance. Dr. Atif Memon was also a great moral support for me in my initial months at UMD. He has inspired and encouraged me in lot many ways. Dr. Bharat Jayaraman and Dr. Aravind Srinivasan have always provided me with ideas and suggestions which helped me firm my approach towards my research work. I have personally known Dr. Chopra and followed his research work. He has been an inspiration for me.

MIND Lab has been like home to me. We've always had a very cordial, understanding and a wonderful group. We have worked together on various projects and organizing many events. I would like to thank Preeti Bhargava, who has been working with me on

Rover-II. We have spent a lot of time sharing our ideas, views, brainstorming etc. I should thank Raghu Narasimhan, Dr. Matthew Mah, Arun Balasubramanian and Ravindra Passan who have been working with me on M-Urgency. I've learnt a lot from them. I thank Richard Baer for giving me views, from a non computer science perspective, on my work. I have had many memorable moments with Rincy Matthew, Arun and Raghu.

I should not fail to thank Dr. Balaji Vasan Srinivasan, Praveen Vaddadi, Karthick Annamalai, Rangarajan Padmanabhan, Dr. Kiran Somasundaram for all the bright moments I have spent with them during my Ph.D days.

I am highly indebted to Amrita Vishwa Vidyapeetham (Amrita University) for nurturing me as a growing researcher and as a responsible individual and for providing me the opportunity to pursue my Ph.D. I am grateful to Br. Abhayamrita Chaitanya, Dr. Venkat Rangan, Dr. K P Soman, Dr. Krishnasree Achuthan, Dr. Sriram Devanathan, Ajay Vadakkepatt, Dr. Vivek Menon and many more at Amrita University. The Amma family in the US has always made me feel at home in this country. I thank Br. Dayamrita Chaitanya, the Pooleris, the Stebens, Binitha Kalesan, Aditya Sabu, Dr. Prasanth Ariyannur Sambhu, Dr. Bhanu Krishnan, Dr. Glenn Manheim, Dr. Lynn Young, Ani Burk, just to list a few from the huge loving family.

I am what I am because of my parents Dr. S. Krishnamoorthy, Prasannakumari Krishnamoorthy and my sisters Radhika Visalam Krishnamoorthy and Dr. Ratna Krishnamoorthy. They have been beside me in every situation of my life, at every of my fall and rise.

Above all, it is HER infinite Grace that has guided me in every part of my work and, more importantly, my life. All I can do is offer my sincere and humble salutations at the

lotus feet of my most beloved Amma, Mata Amritanandamayi Devi, my Guiding Light.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

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 into account the context the designer considers during the design process. 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.

1.1 Motivation

Consider this scenario. Shiv is visiting a city for the first time and checks into a hotel. He pulls out his smart phone and opens up an app that gives him detailed information about the current weather and the extended forecast. He is hungry and brings up another app that not only shows him the restaurants nearby, but also considers that he is vegetarian and is fond of Indian food. More interestingly, it knows where the delivery is to be made; to the precision of the building and the room number he is in. While enjoying his favorite Mexican food, he gets a notification on his phone that he needs to renew his auto insurance within the next 24 hours. Later, as Shiv is about to fall asleep, he feels some uneasiness and gets breathless. He makes an 911 emergency to call a Public Safety Answering Point (PSAP). The emergency/police personnel receives not just a audio call but a live video

stream from his phone along with his precise location on a map interface. In addition, as they are automatically provided with Shiv's additional details including his medical history, interestingly they are also made aware of the fact that Shiv had just had his Indian dinner. This helps the emergency personnel easily understand that Shiv, who is allergic to garlic, is probably reacting to it.

These days, our day-to-day activities are dependent on computing devices. They influence even some of our small decisions. In the above scenario, we see that Shiv makes use of his mobile device to perform some of his usual activities. Interestingly, these activities catered to his needs and likes, making it convenient for him. Even an unusual activity like an emergency call significantly differed from the usual, to his advantage. The difference in each of the activities here is that as they happened, the individuals and his surroundings context was taken into account. In the background, there has been an integration point that compiled the context for the situation, helping the applications to be context aware and more importantly enable them to share the context among themselves. The same is complimented with services that enhance the quality of the contextual information by presenting them effectively.

Our work has been focussed on conceptualizing, designing and implementing:

- a context model that helps effectively capture as much context of a given situation as needed,
- an ontology that supports the model to organize and manage the context,
- an integration platform that not just compiles the context but also compliments it with services that help perceive the same effectively and enables sharing of context

among applications

In the most useful form of information, its context is available with it. Many effective decisions and actions require that the context be explicitly considered. Many computer query outcomes are significantly improved with the use of context information. Considering a simple example, when searching for a well-reviewed restaurant, a preference for vegetarian food will influence the selections a user wants displayed first. An excellent restaurant may have only limited vegetarian options; although this restaurant may be favorable to non-vegetarians, the context of the vegetarian user is important. Clearly, making applications context-aware will improve results. Although context-aware applications can be developed piecemeal, a common context-aware system enables easy integration and communication among applications. These applications can communicate contextual information with one another through such a system, which stores and handles context. Each application need only communicate through a common interface with the system to share context with other applications; there is no need to coordinate each pair of applications to share information.

1.2 Concepts

As we address these issues, it is essential that we familiarize ourselves with various concepts in the field of context-aware computing.

1.2.1 Context

Probably, the most used term in this dissertation is *context*. The term *context* finds definitions in different domains, right from the dictionary to linguistics to computer science. 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.

In the field of context-aware computing, one the most quoted definition of context is by Dey [36] - "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 applications themselves." The focus here is on adaptiveness of the application rather than change in its behavior.

Context is essentially a property or an attribute of any of the elements of the pervasive system. It can also be the ability of an entity in the system to carry out an activity or the role that the entity plays in an activity. It can have a type and description, and a value that can be discrete/nominal. It can also have a hierarchical structure. With this perspective, location, identity can all be considered as properties of an entity and hence part of its context. Consider a person with motor disability, this piece of information is also part of his/her context and can play an important role in situations that require him/her to perform certain tasks, which require movement.

We do not attempt to refine the term *context* in this proposal, but try to extend the definition taking into account the practical implementation of applications that are context-aware. Our definition, as found in [6] is:

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.

1.2.2 Context Model

In any given situation, there may be a lot of factors that influence the context. A framework is required that can recognizes these factors, represents the context information produced or influenced by these factors and handles the context information efficiently. This framework is called a context model. The context model is the crux of any context-aware system. The whole system is built around the context model. The context model must be general; at design time, we do not know what information will be relevant to all applications. New applications may require new types of context information.

In this research work, we present Rover Context Model (RoCoM), a generic, extensible and practically usable context model that is structured around four primitives: entities, events, relationships, and activities which we discuss in detail in Chapter 3.

1.2.3 Context Model Ontology

The context model depends on a mechanism to efficiently store and access the contextual information it represents. One of the most efficient and popularly adopted approach is an ontological approach. An *ontology* is an efficient way to store complex and densely interrelated pieces of information. It can be visualized as a dense graph structure, representing the pieces of information and their relationship.

We present our context model ontology - Rover Context Model Ontology (RoCoMO) in chapter 4, along with the advantages of this approach.

1.2.4 Context Aware Systems / Middleware

Dey et al [37] state - "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task." They focus on adaptiveness of the application rather than change in its behavior. They also specify certain features that a context-aware application should support: presentation of information and services to a user, automatic execution of a service for a user and tagging of context to information to support later retrieval. A context-aware middleware acts as an integration point that compiles the context for the given situation, enabling the applications, subscribed to it, to be context aware.

We present Rover-II, a framework for contextual information integration, that could not just be used to store and compile the contextual information, but also integrate relevant services to efficiently use the same for applications and, more importantly, enable sharing of context among the applications subscribed to it.

1.2.5 Situation

A situation is a temporal state describing activities and relations of detected entities in the given environment [25]. A situation can be understood as a snapshot of the all the possible context at a given point of time, with respect to an entity or application.

We introduce the concept of *relevant context*, the subset of the whole context with respect to the entity or the application that is pertinent to a given situation. We present *situation graph* that captures the relevant context, taking into account all the primitives of RoCoM. We discuss this in detail in chapter 6.

1.3 Objectives

We understand from the literature (presented in Chapter 2) that certain basic problems in the still emerging field of Context-Aware Computing are yet to be addressed. Our research work has been focussed on addressing the following:

- a generic context model that helps effectively capture the context of a given situation, considering all possible factors that influence the context other than just the entity of interest
- a generic, flexible and extensible ontological base for the model that addresses the shortcomings of the existing ones,
- an effective mechanism to model the context of a given situation
- an integration platform that not just compiles the context but enables:
  - sharing of context among applications, to make effective use of context

– enhancing the context information, providing relevant services to perceive the context information better.

1.4 Contribution

This dissertation presents:

- a generic, extensible and practically usable context model - *Rover Context Model (RoCoM)* that is structured around four primitives: entities, events, relationships, and activities; built using *Rover Context Model Ontology (RoCoMO)*.
- the concept of *relevant context* to model the context of a given situation, using a *situation graph*, for efficiently handling and making best use of context information.
- the design and implementation of a context middleware - Rover-II, based on our context model which serves as a framework for contextual information integration, that could be used not just to store and compile the contextual information, but also integrate relevant services to enhance the context information; and more importantly, enable sharing of context among the applications subscribed to it,
- the initial design and implementation of a distributed architecture for Rover-II, following a P2P arrangement inspired from Tapestry [149].
- M-Urgency, a context-aware public safety system built on Rover-II, as a practical proof of our concepts and ideas.

1.5 Organization of the Dissertation

The dissertation is organized into chapters as described as the following. In this chapter we introduced the concepts related to *context-aware computing*. We present our literature survey in chapter 2. We then introduce our context model - *Rover Context Model (RoCoM)* in Chapter 3. RoCoM is based on the ontology defined and developed by us called *Rover Context Model Ontology (RoCoMO)* which we discuss in chapter 4. We discuss the design and implementation of our context framework and middleware - *Rover-II* in chapter 5. We introduce the concepts of *relevant context*, *relevant filtering* and *situation graph* in chapter 6, that discusses the critical part of how the context information is actually handled based on a given developing situation. We then discuss our information model in Chapter 7 detailing how information flows in an event-driven system. Chapter 8 introduces M-Urgency, a full fledged public safety system developed on Rover-II based on the concepts proposed by us. Chapter 9 describes our initial efforts put towards designing a distributed architecture for Rover-II and discusses our initial results in utilizing the same for M-Urgency application. We finally discuss our concluding remarks, our contributions and future work in chapter 10.

Chapter 2

Literature Survey

Context itself requires extensibility as it can expand and contract with time. Context is always in relation to an element of a context-aware system and with respect to some situation or event. To be effective and efficient aid for decision making, context-aware systems should be broadly applicable and support a wide array of functionalities for using and passing around contextual information. The context model, which forms the underlying framework for representing context in the context-aware system, should be general, flexible and expressible.

The main focus of context-aware computing is assessing and understanding a given situation. The term *situation-awareness* is synonymous with context-awareness as found in the literature [25] [44]. Though situation awareness has been stressed in the literature, we were unable to find attempts to provide a generic mechanism to represent the context of a given situation. Our attempt in this dissertation is directed precisely towards that.

A major challenge of context-aware systems is to balance usability with generality and extensibility. The relevant context changes depending on the particular application. The model used to represent the context and its relationship to entities must be general enough to allow additions of context categories without redesign while remaining usable across many applications. Also, while efforts are put in by application designers and developers to make applications *context-aware*, these efforts are customized to specific

needs of the target application, and only certain common contexts such as location and time are taken into account. Therefore, a general framework is called for that can (i) efficiently maintain, represent and integrate contextual information, (ii) act as an integration platform where different applications can share contexts and (iii) provide relevant services to make efficient use of the contextual information. Such a context model and a context-aware framework is presented in this research work, which focuses on simple, extensible and general representations of both context and its relationship to different elements of the system. It is general enough to allow additions of context categories without redesign, while remaining usable across many applications. This chapter discusses some of the important terms in the field and relevant related work from the literature.

Several studies have been presented in the literature that emphasize the importance of context and situation awareness. We list a few in this section. Context Awareness was first introduced by Schilit et al [117] to develop a software adapting to location, nearby people and objects and changes in their information. An early work found in the literature is by Schmidt et al. [118] who specify that generic understanding of a situation requires more than just the location. Dey et al. [36] [38] define the term situation as an abstraction that describes the states of relevant entities with the help of widgets, aggregators and interpreters as part of their context toolkit. Beigl et al. [89] discuss context and situation-aware networking for performance efficiency. They consider the situation of an artefact (a mobile device) as a collection of contextual information that leads to adaptive decisions, including communications behavior for network optimization. They define four attributes for defining context - the type of situation, value of situation, time stamp of change in value and reliability of the value. Cook et al. [32] discuss a situation model that

contains many contextual features derived on the basis of general world knowledge and argue that the inferential information can be activated from the same without depending on pre-existing semantic relations between entities. Petersen et al. [71] have utilized Case Based Reasoning for user situations assessment for context-aware mobile systems[127]. Their focus is on a Multi Agent System, consisting of information suppliers, to provide the user with personalized and contextual information. Meissen et al. [88] argue that understanding the situation of a user allows the system to better target the information to be delivered. They present a model to handle various contexts and situations in information logistics. They define context as a collection of values usually observed by sensors, and a situation builds on this concept by introducing semantical aspects defined in an ontology [135]. Chen et al. [27] presented COBRA-ONT which is an ontology for their Context Broker Architecture (CoBrA) for meeting room domains. They also have another ontology SOUPA [30] which is a more comprehensive and exhaustive ontologies composed by combining several other ontologies.

Despite such a rich landscape in research related to context, context modeling and context-aware systems; a complete, generic and comprehensive model and system are missing. Most of the models and systems are either too domain specific or conceptual. Also, a number of models focus only on the user rather than the overall situation at hand. The solutions presented in this research work are aimed at addressing these concerns. The model we have is simple so that it is easy for designers to translate real world concepts to the modeling constructs, and it is general so that it can be used in a number of domains. Ontological base, using OWL, lends its flexible, expressive and extensible power to it. The system presented focuses not only on the user but also on different dimensions of

context such as location, time, activities etc. The design of Rover II is generic to be applicable in any domain. It provides a means to store and retrieve contextual information, and also facilitates providing relevant services to the applications so that the contextual information can be used more effectively. It also communicates with third party services and external databases for gathering contextual information, consolidating it and presenting it to client applications in a pertinent way. It has advanced context usage functionalities such as learning from past context(context history) and reasoning. A very important aspect of Rover II is that it is designed to serve as an integration platform for different mobile applications which share contextual information.

Below, we present a more specific discussion in the literature on the various concepts pertaining to our research work.

2.1 Aspects of Context

One way to describe the aspects of context of any primitive of a pervasive system, is by categorizing it into static and dynamic based on temporal characteristics [58]:

- *Static contextual information*: This includes those properties of the system that do not change with time. The identity of any entity such as a person or device is its static contextual information.

- *Dynamic contextual information*: This includes those properties of the system that vary with time. For example, the location of an moving entity is its dynamic contextual information.

Context can also be classified as *active context* and *past context*. The context of any primitive that holds true at a given point of time is called its *active context*. Due to the dynamic nature of context, it is important to maintain context history for a primitive. This is called its past context.

The above definitions capture the key idea of context but provide no insight into the actual implementation of a context-aware system. From the perspective of implementation, we require a more concrete representation of context. This can be achieved by a context model.

2.2 Context Model

Strang et al [128] identified the following categories of context models based on the representation of context:

- *Key value models* - These are the simplest models where a ⟨key,value⟩ structure is used to represent the functions of a service.
- *Markup scheme models* - These use a hierarchical data structure with tags, attributes and values.
- *Graphical models* - Models that use Unified Modeling Language (UML) to describe context
- *Ontology based models* - These use ontologies defining concepts and relationships between those concepts. These are by far the most popular context models.

- *Logic based models* - These primarily consist of facts, axioms and rules stored in a knowledge base. New rules can be added to the knowledge base using inference.

- *Object oriented models* - These use objects to represent context types and encapsulate their values. These follow the object oriented paradigm of reusability, encapsulation and inheritance.

A context model should incorporate the most useful features and characteristics of existing context models. As evident in survey conducted by Bolchini et al [24], the context model should serve efficiently in any domain and will be abstract enough to manage all the dimensions of context such as location, time, and user profile. It should be versatile enough to have a rich set of representation features such as flexibility, context granularity and constraints. It should also be advanced enough to incorporate a variety of context usage functionalities such as context construction and reasoning,context information and quality monitoring, automatic learning features, multi context modeling, contextual ambiguity and incompleteness management. Representation of context, which may be a context ontology [72], should be simple and expressible so that any complex piece of contextual information can be easily represented; flexible and extensible so that it can allow expansion and contraction of context; and general so that it can represent contextual information in any domain. We incorporate these requirements of context modeling in a simple, extensible, general and expressible context model, presented in chapter 3

As part of their survey, Bolchini et al [24] also identify five categories of models and context-aware systems based on the main focus of the model, the representation of context and the usage of context. A system which covers all these categories will address

the context problem as a whole and be applicable in any domain. Such systems will be abstract and generic enough to be applicable in any domain or environment. These categories are:

- Channel-device-presentation – This category of context-aware systems are application-centric, with limited management of location and time. They have limited flexibility and are characterized by centrally defined context.

- Location and environment – These models provide accurate location and time management, high degree of flexibility and centralized definition of context.

- User activity – This category of model focuses on the user and the user's activity as the main subject.

- Agreement and sharing of distributed context – These focus mainly on information and context sharing. Contextual information will be available from distributed sources and efficient sharing is required. Context quality monitoring and ambiguity resolving are the key issues here.

- Selecting relevant data, functionalities and services – These cater towards using context to determine which information, functionalities and services are relevant to the user. The application is the main subject of the model and context dimensions such as time and location are accurately provided.

The context model depends on a mechanism to efficiently store and access the contextual information it represents. One of the most efficient and popularly adopted approach is an ontological approach, which we have adopted in our research work.

2.3 Context Model Ontology

Ye et al [146] propose the following coarse-grained criteria to assess ontologies in pervasive computing [113] environments:

- Clarity and Coherence: Ontological concepts must be unique, unambiguous and distinguishable through their properties and constraints.
- Ontological commitment: Ontologies should make sufficient claims about the domain to support the intended knowledge sharing and reuse. If too many claims are made on a domain, the extensibility of ontologies is limited; however, if too few are made, the range of applications that can actually use the ontology will be reduced.
- Encoding bias: Ontologies should be specified at the knowledge level without depending on a particular symbol-level encoding.
- Extensibility: It should be easy to add new terms to ontologies without causing ambiguity.
- Orthogonality: General concepts should be defined as independent and loosely coupled atomic concepts.

Based on their analysis, they conclude that most of the context ontologies have several shortcomings and SOUPA [28] and CoBrA-Ont [27] are the only ontologies that come close to satisfying these requirements. However, we believe that their formulations are inadequate to address the general problem our study addresses. In addition, the problem with some of the general and exhaustive ontologies such as OpenCyc [97] is that they

become too cumbersome to use in a system designed for efficient and effective use in real time. We believe that it is not possible to enumerate all possible concepts, and the relationships between them, that could be used in a practical context-aware system.

A number of studies in the field of context-awareness ignore the details of how context is derived from sensors, and focus more upon modeling contextual information and delivering this information to applications. The goals of these works are similar to ours. We briefly describe them and their underlying ontologies and examine their shortcomings.

CoBrA-Ont [27] is a collection of ontologies for describing places, agents and events and their associated properties in an intelligent meeting-room domain. SOUPA [28] was developed to provide pervasive computing developers a shared and upper ontology that combines many useful vocabularies from different consensus ontologies such as FOAF, DAML-Time, RCC, BDI,and Rei policy ontology [96]. Other contemporary ontologies include CONON [55] where the context ontologies are divided into upper ontology and domain-specific ontologies; CoDAMoS [99], the context ontology is centered around four entities - user, environment, platform and service; ASC/CoOL [129] that enables context awareness and interoperability; Gaia [105] that incorporates ontologies for context awareness, service discovery and matchmaking, and interoperation between entities in a pervasive computing infrastructure mainly geared towards smart spaces; and GLOSS [35] which employs ontologies for the precise understanding of various contexts and services in smart spaces.

Three surveys mentioned earlier ([107], [79] and [19]) have asserted that of all the current ontologies used for context modeling, SOUPA is the most comprehensive ontology satisfying a majority of the requirements listed. However, both SOUPA and CoBrA-

Ont have no provision for provenance, quality of context and multiple representations. CONON enables provenance by using the concept of sensed, derived, aggregated or deduced context but lacks features such as comparability. Gaia takes on the challenge of modeling uncertainty and reasoning over it. However, their ontologies are restricted to the smart spaces domain and do not model provenance either. As mentioned above, even general and exhaustive ontologies such as OpenCyc [97] have shortcomings such as being too cumbersome to use in a system that is designed to be used efficiently and effectively in real time.

Hence, in our opinion, it is best to develop a generic base ontology and make it extensible for users just as most of the context model and ontologies like SOUPA have done. Our goal is to address all possible shortcomings that we discussed above. Our context model ontology - RoCoMO (Rover Context Model Ontology) is being developed in OWL2 DL [1] that is expressive, versatile and supports reasoning. It also satisfies the assessment criteria mentioned above and its concepts are unique, unambiguous and clearly defined; it is extensible and hence not restricted to a single domain; and it does not have any encoding bias. We present RoCoMO in chapter 4.

2.4 Situational Modeling

According to Endsley [44], situation awareness refers to "the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future". Over the years, guidelines and processes have been developed for building situation awareness models from

a goal-oriented perspective. Endsley [45] discusses how situational awareness is critical to convert data into information, so that the right and efficient use of the same can be made in systems. He introduced a three layered situation model of Perception, Comprehension and Projection [43] for representation, interpretation/assessment and prediction of future states respectively. Feng et al [48] extended this model, adding a terrain model for command and control domain in Context-Aware Decision Support system (CaDS). It incorporates a shared situation awareness model providing customized views and services through a group of entity agents. Jameson [63] explains how an adaptation decision should incorporate not just the information about the current situation but consider user's current state, users' behavior, long-term user properties along with the information about the current environment. Brdiczka et al [25] argue how context-aware services should be based on supervised learning of situation models through regular user feedback. They explain their approach with a scenario of an intelligent home wherein the various devices in the house operate based on the resident's feedback. We do not find in the literature much work wherein context-aware systems are extensively integrated with situational modeling. Though some of the approaches listed above have taken some steps in this direction, the focus has been on some rule based states to understand the context of a user, rather than a formal approach to context modelling. RoCoMO has been designed with an extensive ontological model driven foundation along with capabilities to model a situation ontologically.

2.5 Context Aware Systems

In section 2.2 we discussed the five categories of models and context-aware systems identified by Bolchini et al [24]. A context-aware system, that captures all the features of these aforementioned categories, will focus on the context problem as a whole, and will be abstract and generic enough to be applicable in any domain or environment. Some of the contemporary context models and context-aware systems, which fall into all the categories, include ACTIVITY [69], an Activity Theory based model which encapsulates context as a set of elements which influence users' intentions while doing some activities. The model appears to be in its nascent stage and the implementation details are not very lucid. Another such system called ConceptualCM is described in [34], which has a conceptual framework that considers context as a part of a process rather than a state. In this system, all possible contextual information for a scenario is contained in an information space modeled as a directed state graph. Each node represents a contextual piece of information and edges denote the conditions for changing context. In [143], Yang et al describe an ontology based context model which is specific to learning environments.

CASS [47] is a centralized server-based context management framework with distributed sensors. It consists of a sensor listener and a rule engine. CoBrA [28] consists of a Context Broker for sharing contextual information, a Context Knowledge Base, and a Context Acquisition Module, but is too specific to the domain of meeting management. In Context-ADDICT [23], a tree based structure called Context Dimension Tree is proposed, which can be used to represent context at different levels of granularity. However, the model lacks features like context history and reasoning. GraphicalCM [58], a theoret-

System	Space	Time	Relative/Absolute	Context History	Subject (User/Application)	User Profile (Role/Feature based)	Variable Context Granularity	Valid Context Constraints	Flexibility	Context Construction (distributed/centralized)	Context Reasoning	Learning Features	Multiple Context
ACTIVITY	+		A	+	U	F	+		+	C	+		+
CASS	+	+		+	U					D	+		
CoBrA	+	+	A		A	F			+	D	+	+	
ConceptualCM	+	+	R	+	A	R			+	C	+	+	+
Context-ADDICT	+	+	R/A		A	R	+	+	+	C			
GraphicalCM	+	+	R		A	F			+	C			+
SOCAM	+	+	R/A		A	F				D	+		
Context Toolkit	+	+	A		A	F	+		+	C	+		
Hydrogen Project	+	+	R		U	R			+	D			
Rover	+	+	R/A	+	U	F/R	+	+	+	C	+	+	+
Rover-II	+	+	R/A	+	U	F/R	+	+	+	C	+	+	+

Figure 2.1: Context Models, Context-aware Systems and Middleware - A Comparison

ical context model, focuses on context quality and its temporal aspects. Context Toolkit [111] is a context-aware system for distributed setting with a peer to peer architecture. It consists of distributed sensors and a centralized discoverer. The Hydrogen project [59] follows a completely decentralized architecture with two devices exchanging contextual information as soon as they discover that they are in close proximity. Bolchini et al [24] provided a detailed comparison between context models/ context-aware systems from the literature. We extend this comparison, considering additional criteria, and present the same in Figure 2.1. Importantly, we have highlighted the systems for which we found concrete implementation details in the literature. We compare these systems with our context-aware middleware, Rover-II (presented in chapter 5).

Most of the presented models in the literature are user centric, and the context of a situation is defined only with respect to the entities involved in it. Moreover, most of the

systems and middleware proposed are too specific to a particular domain in their implementation, or are general but only conceptual. We address these issues through Rover-II and RoCoM. RoCoM considers (in addition to the entities), events, activities, their properties, and the relationships between them. RoCoM is simple so that it is easy for designers to translate real world concepts to the modeling constructs, and it is general so that it can be used in many domains. It represents various dimensions of context such as location, identity, time, as well as the application and user. It is an ontological model and the modeling language being used for its implementation is the Web Ontology Language (OWL) which lends its flexible, expressive and extensible power to it. Rover II is being designed to support all the requirements of middleware in ubiquitous environments as mentioned in [102] - It supports integration of different mobile applications and collection and storage of information from heterogeneous services, provides support for inference, reasoning and learning. Ontological approach, using RoCoMO, allows a common semantic framework for representing context.

2.6 Context-awareness, Humanitarian Relief Planning and Social Computing

A lot has been discussed in the literature on how context of the situation and place is relevant when humanitarian relief is planned. Wendy et al [139] reflect on how local area details provide the context to which plans are tailored. Rainsford et. al. [31] highlights how information overloading is avoided if information is provided to the user in specific context. Barani et. al [13] specifically stress upon context-awareness when collabora-

tive planning is done between distributed planners. Kikiras et. al. [70] has proposed an integration platform for autonomic computing for disaster relief. The idea here is to provide a middleware for intelligent, collaborative and context aware services using rumor spreading techniques. Siren [142] provides a foundation for gathering, integrating, and distributing contextual data, such as location and temperature to support tacit communication between firefighters with multiple levels of redundancy in both communication and user alerts. Tomaszewski et. al. [131] argue that information foraging and sense-making with heterogeneous information are context-dependent activities. They attempt the challenges of constructing and representing context within visual interfaces that support analytical reasoning in crisis management and humanitarian relief. MacEachren et al [85] specifically discuss about geo-spatial information. They briefly put forth on how groups or groups of groups can work with geospatial information and technologies in crisis management.

Handling emergency situations is critical. Research works have been presented to make emergency situation handling more effective and efficient. The efforts range from improving the quality of the service to providing better information to the emergency responders. The focus of M-Urgency, a context aware public safety system built on Rover-II, has been on providing better contextual information to the emergency responders which enables them to be better prepared for the situation and can provide better efficient and effective service. We present M-Urgency, in detail, in chapter 8.

Connie White et. al. [140] recognize the extreme popularity of social network sites today and that they can be employed in the context of emergency service. Considering the open nature of the social networks, they can be effective in creating awareness, pre-

paredness and sharing of some critical information but contradict the nature of how 911 services work today where maximum confidentiality is maintained. Jiang et al [142] show how gathering, integrating and distributing contextual information enables tacit communication between fire fighters. NG-911 [60], the next generation 911 service operates on Internet protocol enabled emergency service communications network infrastructure. The focus here is to establish better interoperability among the emergency call centres, fire and rescue unit, law enforcement department and other emergency services. A major accomplishment of NG 911 has been in establishing a centralized approach for receiving an emergency from anywhere in the USA and routing the call to the appropriate call center based on the location provided by the mobile vendor. E-911 [39], enhanced 911 has enabled making emergency calls through voice over IP (VoIP), providing the location information and call back number. There has been attempt to make use of video stream to improve situational awareness during emergencies, as presented by Bergstrand et al [18]. The initial study proves the effectiveness of video communication in such situations. Kraut et al [73] have also shown how visual information can be used in effectively accomplishing a collaborative physical task in a given situation. They show how the impact of the amount and the quality of visual information on the performance. There have been other attempts to provide location services during emergency calls as in [114] and [148].

Our efforts [77] align with the overall objectives of 911.gov, where the focus is on forming Community Response Grid (CRG) [124][62][61][141]. The Community Response Grid would enables residents to report and receive information about emergent events such as fires, floods, tornados, hurricanes, community health concerns, and ter-

rorist attacks through web-enabled computers and mobile devices, as well as cell phones providing text messages, photos, or videos that could be shared with community officials and other citizens. The central idea of a CRG is to provide ICT-empowered resident-to-resident (R2R) assistance when it is needed. The Nation of neighbors [94] is another effort in effectively involving social contributions to handle crimes. It's Reporting System enables citizens to report incidents which may go through a member moderation process before it is being broadcast and utilized by law enforcement groups and/or other citizens.

We did not find in the literature efforts taken to handle all the three critical aspects which we try to address through M-Urgency: (1) utilizing technology, enabling social contribution towards public safety, (2) utilizing technology for effective communication (audio, video, real-time location etc.) and more importantly (3) real-time/immediate reporting of the emergency, directly to the PSAP.

Chapter 3

Rover Context Model - RoCoM

The context of any entity may include a very large number of items, at any given time. But depending on the current situation and goals, only a few of these items may be pertinent. This defines the *relevant context*. Note that the relevant context is a subset of the overall context, and is likely to change as the situation changes and even as additional information becomes available. To process relevant context in a context-aware system, a framework for storing and representing context is necessary. This framework is called a context model. The context model must be general; at design time, we do not know what information will be relevant to all applications. New applications may require new types of context information. In this chapter we discuss our context model - Rover Context Model (RoCoM).

We first discuss about the hierarchical approach of handling contextual information and then present our Context Model - RoCoM and the ontology we developed for the same - RoCoMO.

3.1 Hierarchical Storage

While the context of the *entities* are stored statically in Rover II, the context of the events, activities and relationships is considered dynamically. Contextual information storage and representation on Rover II is adapted from the RDF approach. We attempt to

extend the approach and believe that it would be more effective if the relationship is represented hierarchically. For example, to say that Mary is related to Bob and Mary is also married to Bob, the RDF approach requires two triplet entries. This incurs redundancy in storage. Thus, we propose to represent and store the information hierarchically. This also enables access to the contextual information at various abstract levels. Figure 3.1 shows the relationship between A and B as represented in Rover II. Say the three relationships we would like to store are: A is related to B, A is friend of B and A is classmate of B. Using RDF, we should need to store three entries, but when we represent the relationship hierarchically as shown in Figure 3.1, only two entries are required. When A is friends with B or when A is a classmate of B, it implies both A and B are definitely related. An additional entry can be avoided in this case. When we deal with exhaustive information being stored about every entity, this would be significantly reduce the number of records.

Every entity has a hierarchical tree representation, with leaf nodes containing either a *literal* or another *entity* while the internal nodes are essentially the contextual information of the entity. Every entity is recognized with a unique identifier. Every non-leaf node is represented with a tag, as shown in the example in Figure 3.1. Each tag may have a set of properties *properties* associated with it. For example, a tag <Height> can have a property that specifies the units of the value stored as the leaf of the tag.

The hierarchical representation and storing of contextual information has its advantages as listed below:

- *Multi level relationship* - Multiple levels of properties that an entity may have can be easily represented here

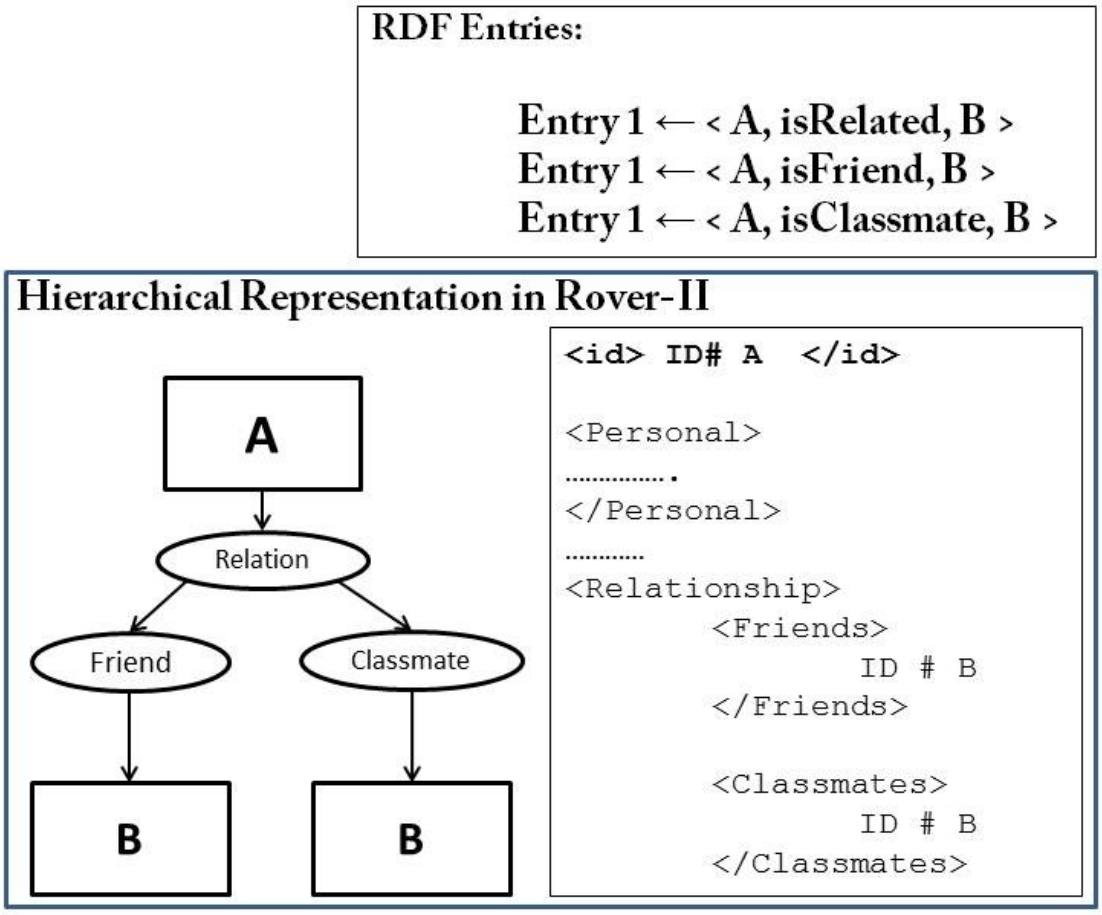

Figure 3.1: Hierarchically representing the relationship while storing the contextual information

- *Abstraction* - Levels of abstraction of information are possible in this case. If a list of all the friends of A is required, it can be easily accessed here. Additionally, if all those related to A (not just friends, but also siblings) are required to be listed, it can be accomplished efficiently here.

- *Specificity* - Going down the hierarchy, very specific information can be efficiently accessed here.

- *Exhaustive* - Information regarding an entity can be represented and stored without any limitations. All the information will be available through the unique ID of the entity.

- *Extensible* - Additional information can be easily added in new branches without affecting the existing information.

- *Generic* - Any entity can be represented with this approach with ease; be it an individual, an object or even an organization.

- *Non-redundant* - Information maintained here will avoid redundancy as compared to a tuple record based approach. For example, the very information of the unique ID of an entity has only once instance here as compared to a tuple record based approach, where every record needs to store the ID.

The following shows a partial ontology description of a "Person" entity.

```
⟨owl:Class rdf:ID="Entity"/⟩
⟨owl:Class rdf:ID="Person"⟩
  ⟨rdf:subClassOf rdf:resource="#Entity"/⟩
⟨/owl:Class⟩
```

⟨per:Person⟩

⟨per:Identification rdf:type=“Person” rdf:ID=”Per1234567890”/⟩

⟨per:Personal rdf:ID=“PersonalInfo”⟩

⟨per:FirstName rdf:datatype=“String”⟩first⟨/per:FirstName⟩

⟨per:LastName rdf:datatype=“String”⟩last⟨/per:LastName⟩

⟨per:Gender rdf:datatype=“String”⟩Male⟨/per:Gender⟩

⟨per:DOB rdf:datatype=“Date”⟩01-01-1900⟨/per:DOB⟩

⟨per:ContactNo rdf:datatype=“String”⟩1234567890⟨/per:ContactNo⟩

⟨per:Address rdf:datatype=“String”⟩lane⟨/per:Address⟩

⟨per:Personal⟩

⟨per:Professiona rdf:ID=“ProfessionalInfo”⟩

⟨per:Designation rdf:datatype=“String”⟩random⟨/per:Designation⟩

⟨per:URL rdf:resource=“http://xyz.org/ first¡/per:URL”/⟩

⟨per:email rdf:resource=“mailto:first@xyz.org/⟩

⟨per:Professional⟩

⟨per:Medical rdf:ID=“MedicalInfo”⟩

⟨per:Blood rdf:ID=“BloodInfo”⟩

⟨per:Group rdf:datatype=“String”⟩A+⟨/per:Group⟩

⟨per:Pressure rdf:ID=“BloodPressure”⟩

⟨per:systolic rdf:datatype=“number”⟩110⟨/per:systolic⟩

⟨per:diastolic rdf:datatype=“number”⟩110⟨/per:diastolic⟩

⟨/per:Pressure⟩

⟨/per:Blood⟩

⟨per:History rdf:ID=“medicalHistory”⟩

⟨per:dataSource1 rdf:resource=http://128.8.126.21/ds1.php/⟩

⟨per:dataSource2 rdf:resource=http://128.8.126.21/ds2.php/⟩

⟨/per:History⟩

⟨/per:Medical⟩

⟨per:Qualification rdf:ID=“qualifications”⟩

⟨per:UnderGrad rdf:ID=“UnderGrad”⟩

⟨per:SchoolName rdf:ID=“UGSchoolName rdf:datatype=“String”⟩

Springfield College⟨/per:SchoolName⟩

⟨per:GPA rdf:ID=“UGGPA” rdf:datatype=“number”⟩4.0⟨/per:GPA⟩

⟨/per:UnderGrad⟩

⟨per:Grad rdf:ID=“Grad”⟩

⟨per:SchoolName rdf:ID=“UGSchoolName rdf:datatype=“String”⟩

Big State University⟨/per:SchoolName⟩

⟨per:GPA rdf:ID=“GPA” rdf:datatype=“number”⟩4.0⟨/per:GPA⟩

⟨/per:Grad⟩

⟨/per:Qualification⟩

⟨per:Relations rdf:ID=“Relations”⟩

⟨per:Blood rdf:ID=“BloodRelations”⟩

⟨per:Parents rdf:ID=“ParentsRelationship”⟨

⟨per:Father rdf:ID=“Father” rdf:resource=”#234567890/⟩

⟨per:Mother rdf:ID=“Mother” rdf:resource=”#2343427890/⟩

⟨per:Parents⟩

⟨per:Siblings rdf:ID=“SiblingRelationship”⟩

..

..

⟨/per:Siblings⟩

⟨/per:Blood⟩

⟨per:Business rdf:ID=“businessRelationship”⟩

..

..

⟨/per:Business⟩

⟨per:Friends rdf:ID=“friendRelationship”⟩

..

..

⟨/per:Friends⟩

⟨/per:Relations⟩

⟨per:Person⟩

The hierarchical representation of information can be implemented using an ontology language such as Web Ontology Language (OWL). We have implemented our ontology - RoCoMO using OWL which we discuss in chapter 4. OWL has several advantages over traditional modeling languages [27]:

- It is more expressive and hence allows more versatile knowledge representation.
- It is specifically an ontology language and hence, has many predefined classes and properties for modeling ontologies.
- It is a standard and is endorsed by the well known World Wide Web Consortium. As a result, a number of development and integration tools are available for it.

3.2 Rover Context Model - RoCoM

An *ideal* context model is one which incorporates the most useful of the above features and characteristics. As identified by the survey conducted by Bolchini et al [24], the context model should serve efficiently in any domain and should be abstract enough to manage all the dimensions of context such as location, time, and user profile. It should be versatile enough to have a rich set of representation features such as flexibility, context granularity and constraints. It should also be advanced enough to incorporate a variety of context usage functionalities such as context construction and reasoning, context in-

formation and quality monitoring, automatic learning features, multi context modeling, contextual ambiguity and incompleteness management. Representation of context should be simple and expressible so that any complex piece of contextual information can be easily represented; flexible and extensible so that it can allow expansion and contraction of context; and general so that it can represent contextual information in any domain. We incorporate these requirements of context modeling in a simple, extensible, general and expressible context model, RoCoM

Most of the proposed models in the literature are user centric, and the context of a situation is defined only with respect to the entities involved in it. RoCoM considers (in addition to the entities), events, activities, their properties, and the relationships between them. Moreover, most of the systems and middleware proposed are too specific to a particular domain in their implementation, or are general but only conceptual. To address these shortcomings, we have developed RoCoM. RoCoM is simple so that it is easy for designers to translate real world concepts to the modeling constructs, and it is general so that it can be used in many domains. It represents various dimensions of context such as location, identity, time, as well as the application and user. It is an ontological model and the modeling language being used for its implementation is the Web Ontology Language (OWL) which lends its flexible, expressive and extensible power to it.

The subsections below discuss the various aspects of RoCoM.

3.2.1 Primitives

RoCoM is an ontological model built around four primitives that can used to describe a situation and its associated context. These primitives are the building blocks of every context-aware system built on RoCoM. Each piece of contextual information is associated with at least one of these primitives. The primitives are:

1. *Entity* - An individual element of the context-aware system, such as a person, a place, an organization, or a computing device. The properties or attributes of an entity constitute its context. An entity can be classified as physical or virtual; permanent or transient; single or group. Typically, entities would be specific to a situation. For instance, in the case of an accident, an entity involved can be a person, place, car, building, etc.

   An *entity* is of the form

   entity $\langle$ **n, C** $\rangle$

   where **n** is the name of the entity

   and **C** is its context, a set of attributes.

2. *Activity* - An activity occurs for a fixed time and causes a change in context. Every activity is driven by a desired outcome or an implicit *goal* that can be long term or short term. The goal ceases to exist once the activity to achieve it has been performed. There can be interaction or coordination between different activities to achieve the common goal. Every activity is performed by one or more entities and derives a part of its context from those entities. It should also have some executable or action associated with it, which is required to carry out the operations neces-

sary for the activity to achieve the goal. An instance of an activity can be calling Emergency Response Service to dispatch help to an accident victim.

An *activity* is of the form

$$\textbf{activity} \langle\ \textbf{n, } \mathbf{t}_s\textbf{, } \mathbf{t}_e\textbf{, G, E, C}\ \rangle$$

where **n** is the name of the activity,

$\mathbf{t}_s$ is the starting time associated with the activity,

$\mathbf{t}_e$ is the ending time associated with the activity,

G is the goal driving with the activity,

E is the set of entities associated with the activity,

and **C** is its context, a set of attributes.

3. *Event* - An event has one or more entities involved in it and can consist of one or more activities. Every event will have a start time and/or end time, along with a duration, associated with it. An event catalyzes the context-aware system and sets the implicit goal for it. This goal can then be further broken down into its *subgoals* for each of the activities. An event has its own properties or context, and inherits the context of entities and activities involved in it. For example, a road accident can be considered as an event that catalyzes the context-aware system to launch an emergency response.

   An *event* is of the form

$$\textbf{event} \langle\ \textbf{n, E, A, C}\ \rangle$$

where **n** is the name of the event,

E is the set of entities associated with the event,

A is the set of activities associated with the event,

and **C** is its context, a set of attributes.

The event and the activities are driven by a *goal*, represented by a condition. As the event triggers a situation, it sets a overall goal to be addressed in the situation. The activities will have the sub-goals respectively and cease to exist once the condition is satisfied. As the goal corresponding to the event is satisfied, the situation ceases to exist.

4. *Relationship* - A relationship describes how two primitives relate to each other. A relationship can also have context. Relationships can be derivative or transitive i.e. if primitive A has a relationship with primitive B, which in turn has a relationship with primitive C, then A may also have a relationship with C.

   A *relationship* is of the form

   $$\textbf{relationship} \langle\, \mathbf{P}_1, \mathbf{n}, \mathbf{P}_2 \,\rangle$$

   where $\mathbf{P}_1$ and $\mathbf{P}_2$ are primitives

   and **n** is the name of the relationship.

3.2.2 Templates

An important element of our model design, which makes it practical and implementable, is the concept of a *Template*. A template is a predefined default structure for

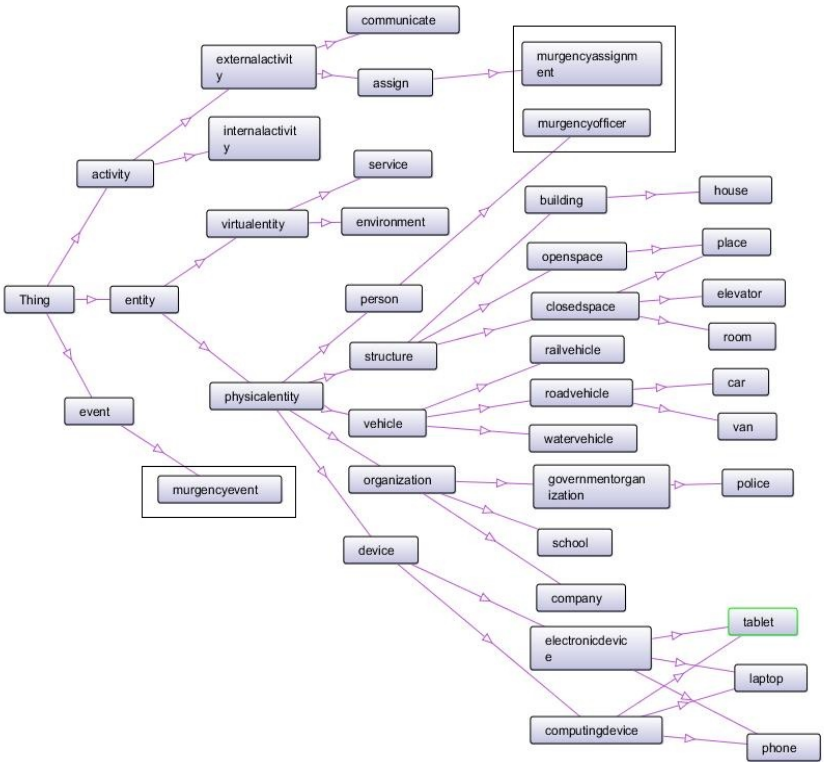

Figure 3.2: Partial description of the RoCoM Ontology

any primitive. It can take the form of an information model in case of an entity, describing what contextual information the entity can possibly have. It can also take the form of a sequence of actions and executables for an activity or an event. For a relationship, it can simply describe what primitives are part of the relationship. The template for a primitive contains elements of its complete context. The information for some of the elements may not be available, in which case the template can have placeholders that will be replaced when the information becomes available.

RoCoMO is being implemented in OWL using the Protege-OWL editor [100]. The templates for primitives are essentially formalized descriptions of the top level classes and derived sub-classes in RoCoMO. An *individual* instance of a primitive can be obtained by instantiating a template. The attributes of an individual represent its contextual information. The RoCoM Ontology includes the following top level classes:

- entity - This class can have several sub-classes to represent individual entities in a domain. For instance,“ person” is a subclass of “physicalentity” which is a subclass of entity. The following shows a fragment of the “person” template including the object property ‘daughter’ and two data properties - ‘likesfood’ and ‘likespets’, an individual named “xyz” of that class with those properties specified.

⟨Class rdf:about=”&person;person”⟩

⟨ rdfs:label xml:lang=”en”⟩person⟨/rdfs:label⟩

⟨rdfs:subClassOf rdf:resource=”http://www.semanticweb.org/prets/ontologies/rover/rdfxml/physicalentity#physicalentity”/⟩

⟨/Class⟩

⟨ObjectProperty rdf:about=”&person;daughter”⟩

⟨rdf:type rdf:resource=”&owl;FunctionalProperty”/⟩

⟨rdf:type rdf:resource=”&owl;InverseFunctionalProperty”/⟩

⟨rdfs:label xml:lang=”en”⟨daughter⟨/rdfs:label⟩

⟨rdfs:subPropertyOf rdf:resource=”&person;contact”/⟩

⟨inverseOf rdf:resource=”&person;father”/⟩

⟨inverseOf rdf:resource=”&person;mother”/⟩

⟨/ObjectProperty⟩

⟨DatatypeProperty rdf:about=”&person;likesfood”⟩

⟨rdfs:label xml:lang=”en”⟩likesfood⟨/rdfs:label⟩

⟨rdfs:subPropertyOf rdf:resource=”&person;likes”/⟩

⟨/DatatypeProperty⟩

⟨DatatypeProperty rdf:about=”&person;likespets”⟩

⟨rdfs:label xml:lang=”en”⟩likespets⟨/rdfs:label⟩

⟨rdfs:subPropertyOf rdf:resource=”&person;likes”/⟩

⟨/DatatypeProperty⟩

⟨NamedIndividual rdf:about=“&person;xyz”⟩

⟨rdf:type rdf:resource=“&person;person”/⟩

⟨rdfs:label xml:lang=“en”⟩xyz⟨/rdfs:label⟩

⟨person:likespets rdf:datatype=“&xsd;string”⟩dogs⟨/person:likespets⟩

⟨person:likesfood rdf:datatype=“&xsd;string”⟩indian⟨/person:likesfood⟩

⟨/NamedIndividual⟩

- event - Several subclasses can be derived from this top level class to represent specific events such as “accident”, “wedding” etc.
- activity - This top level class can be used to derive several subclasses like “call”, “assign” etc.
- relationship - A relationship can be between two classes (subclass/superclass), a class and an individual (member) or a specific relationship between two individuals (object properties).

Figure 4.2 shows a partial description of the RoCoM Ontology. The encircled concepts show the application specific (application being M-Urgency in this case) concepts

that can be derived from the top level core concepts.

As mentioned earlier, *relevant context* is the context of a primitive that is pertinent to the event or activity. For any entity, it will be the intersection of the entity's context and the context of the activity or event, with only direct properties of the primitives being taken into consideration. It can expand or contract depending on the situation, activity, event and other primitives involved in it. For example, when an event such as an accident takes place, the event's relevant context includes the context of the automobiles, the victims and the place. As the rescue and clean up starts, the context starts contracting until all the activities are completed. Context merging is aggregating the relevant context of all the primitives involved in an activity or an event.

3.2.3 Rules

Context reasoning involves using the contextual information in an intelligent manner. It is used in checking the consistency of context and also in deriving high level (implicit) context from low level (explicit) context [138]. One of the simplest techniques of reasoning that will be carried out in Rover II is rule based inference. A rule describes a change in context for any of the primitives and can be used to infer new contextual information from already existing contextual information. Rules can be system specific or application specific. For instance, when an emergency call is made on M-Urgency, the change in context of the caller can be described in terms of the following rule:

$$\text{currState}\langle \text{ dispatcher,available } \rangle \wedge \text{onCall } \langle \text{ dispatcher,true } \rangle \rightarrow \text{currState}\langle \text{ dispatcher,busy } \rangle$$

More complex techniques for reasoning can also be adopted such as probabilistic reasoning or first order logic. While the system specific rules will be built into the reasoning module at the time of implementation, the application specific rules will have to be provided by application developers.

Chapter 4

Rover Context Model Ontology - RoCoMO

Recent years have witnessed rapid advances in enabling technologies for pervasive computing environments - an important step being context-awareness in systems. Context awareness enables a new class of applications in pervasive computing that can help users navigate through unfamiliar territory, find preferred restaurants nearby, receive messages in the least unobtrusive manner, get extra sleep when meetings are canceled, find people with similar interests, and so on. The use of context information in these applications reduces the amount of human effort and attention an application needs to service the user's requests.

Dey and Abowd [3] describe a context-aware system as one that "uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." Moreover, in pervasive computing environments, various entities often have to cooperate and integrate seamlessly to achieve a common objective. Thus, "Situation awareness is the capability of the entities in pervasive computing environments to be aware of situation changes and automatically adapt themselves to such changes to satisfy user requirements, including security and privacy." and a *Situation* can be described as "a set of contexts in the application over a period of time that affects future system behavior." [144]. Thus, a situation can be considered as an amalgamation of the context of several entities interacting and coordinating with each other, and often performing one or more

activities.

The context model forms the underlying framework for modeling and representing context in the pervasive computing environment and context-aware systems. To support situation-awareness and adaptation of the entities in pervasive computing environments, it is necessary to model and specify context and situations in a way such that the contextual information can be easily exchanged, shared and reused. As discussed in several papers including Chen et al [28] and Krishnamoorthy et al [78], ontologies are a powerful tool for modeling context and the encompassing situations in context-aware systems because they promote knowledge sharing and reuse across different applications and services interacting in a pervasive computing environment, thus, enhancing the interoperability. They allow context-aware systems to use existing logic reasoning mechanisms to deduce high-level, conceptual context from low-level, raw context, and handle uncertainty and inconsistency in context. They can be combined to form a more complex ontology and save the effort of starting from scratch.

We introduced a general and intelligent context-aware middleware called Rover II and its general, flexible and extensible context model for context and situation modeling, called Rover Context Model (RoCoM), in Krishnamoorthy et al [78]. As discussed, the model has four Primitives - Entity, Event, Activity and Relationship. These Primitives are the building blocks of every context-aware system or middleware built on this model. Any piece of contextual information in the system should be attached to one of these primitives. Any situation can be modeled in terms of these primitives. RoCoM is an ontological model and its underlying ontology is called Rover Context Model Ontology (RoCoMO). In this chapter, we describe this ontology and illustrate its benefits for context

and situation modeling in pervasive computing environments as well as the utility of its capabilities such as support for provenance, traceability, Quality of Context and multiple representations of context etc. RoCoMO consists of a Core Ontology which has generic concepts and an Application Ontology that extends the Core Ontology to domain specific concepts. The purpose of our ontology is to lay the groundwork of what we hope could become a common standard for context-aware services, applications, middleware and systems development.

4.1 Context and Situation Modeling Case Study - Fire Incident

We discuss a practical case study here in order to illustrate the varied nature of context and the capabilities that context models and ontologies should possess in order to represent and model this situation in real time pervasive computing systems and environments. We will return to this case study in Sections 4.4 and 4.5 to illustrate RoCoMO's modeling capabilities. For this situation, we have selected the domain of rescue and evacuation but this, by no means, restricts RoCoMO's applicability and generality. RoCoMO can be used in other domains as well.

> A fire incident takes place in a room on the fourth floor of a building on a university campus. Fire fighters and responders are using a context-aware system to coordinate the rescue efforts. A responder using the system gets updated readings from two temperature sensors in the room on fire. Using this contextual information, the system can determine the time that responders have to evacuate the building before the whole building is engulfed into

flames. The temperature information also has a quality measure attached to it to convey any inaccurate or incomplete information. Another responder is accessing the system to get confidential floor maps of the building and also determine the evacuation route people should take based on the floormaps and the time remaining.

Representing and modeling this situation in a context-aware system is not trivial. This situation involves interaction between several entities such as responders which perform one or more activities. The entire situation is catalyzed by an event like the fire incident and the goal of the situation is to evacuate the building. Each activity being performed by the system or an entity is driven by a sub-goal. Some of the activities can occur simultaneously, for instance, calculating the time remaining for evacuation and accessing the floor maps of the building while the evacuation routes can be determined only when the floor maps are available. The room has contextual information that needs to be modeled and represented such as the temperature readings of the room. To avoid ambiguity, the information must be clearly marked with its source (which sensor it is coming from) as well as the encoding format (whether it is in Celsius or Fahrenheit). This requires support for encoding format as well as provenance. The model should also support attachment of quality attributes to contextual information such as probability or certainty. Also, since the readings are getting updated at a fixed time interval, they should be timestamped to determine the most recent reading. Another aspect of the model and the system is security - only authorized personnel such as responders have access to the floor maps of the building. We discuss these multiple facets of context and the required supported capabilities

of the context model in the next section.

4.2 Characteristics of Context, and Requirements of Context Models and Ontologies

4.2.1 Characteristics of Context

The most quoted definition of context is by Dey [36] - "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 applications themselves." Though this definition suffices to describe what context is, it doesn't highlight the varied nature of context. Henricksen et al. [58] have summarized the following characteristics of context:

- Temporal Characteristics - Contextual information can be static i.e. those aspects of a pervasive system that are invariant, such as a person's date of birth. However, the majority of contextual information is dynamic, such as location, with its persistence being highly variable.

- Imperfection in Contextual Information - Contextual Information may be incorrect, inconsistent, redundant or incomplete. This is mainly because pervasive computing environments are highly dynamic and hence the contextual information may become redundant quickly. Also, inferring high level context from raw sensor input, such as inferring a person's activity from location and sound level, may introduce errors and uncertainty. In some cases the contextual information may be incorrect

due to a faulty sensor or incomplete due to lack of sufficient input from the source.

- Heterogeneity - Often the sources of context are distributed and the contextual information supplied by them requires processing and aggregation in order to be useful to the context consumer.

- Multiple Representations - Contextual information usually originates from sensors where it is present in its raw form. For instance, a person's weight can be represented in pounds or kilograms. It may then need to be transformed into a logical form that an application or user can understand. Moreover, requirements can vary between applications and hence the different representations of context should be preserved.

- Contextual information is Highly Interrelated - High level contextual information may be derived from low level raw sensor data or other high level contextual information by certain rules. For instance, concluding that a person is sleeping based on his/her phone activity and current time.

- Granularity of Context - Contextual information can be represented at different levels of abstraction. For instance, location of a user can be represented at a fine-grained level in terms of latitude/longitude and at a coarse-grained level in terms of the name of a city or a building.

4.2.2 Requirements for Context and Situation Models

These varied characteristics of context require support at both the context modeling as well context-aware system level. To justify this argument, we found that several surveys of context models including those by Reichle et al [107], Krummenacher and Strang [79] and Bettini et al [19] have concluded that context modes should support:

- Ease of development: Model-Driven Development should be followed which provides appropriate development support to the software developers to ease their tasks with respect to the whole development process and incorporates all views and aspects.

- Heterogeneity and Mobility: Pervasive computing environments are characterized by distribution, heterogeneity, unpredictability and unreliable communication links and the context model should have support for them.

- Machine-interpretable representation of contextual information: The model must employ machine-interpretable representation of context to tackle heterogeneity by using semantic annotations that enable automatic exploitation and transformation of information in distributed context sharing scenarios as well as automatic context reasoning.

- Quality of Context: Unlike data in conventional database systems, context data is characterized by properties such as incompleteness, ambiguity, uncertainty, inaccuracy, and temporal nature. The context model should be able to model and represent this imperfection.

- Contextual information partitioning: Because of the distributed nature of ubiquitous computing environments, it is possible that the sources of contextual information are partitioned. The context models should enable the aggregation and merging of the data when needed. This is related to the Heterogeneity and Mobility requirement.

- Evolvability, Extensibility, Flexibility and Applicability: Context models should not be rigid but flexible and extensible. Thus, it should be able to support new and varied application domains, and represent the contextual information required for them. It should evolve with the applications and their context needs.

- Comparability: Contextual information sources constitute a variety of sensors and devices which often use different measurement and encoding systems thus resulting in a heterogeneous set of values describing the same entities. Hence, it is necessary to provide means to compare values with different units and encodings.

- Traceability and Provenance: In order to provide adequate control and interpretation of contextual information, the model should provide means to determine the source of data and the transformations made to it.

- Timestamping: This helps determine the 'freshness' as well as versioning of the information which further enables resolution of conflicts and ambiguity.

- Satisfiability: This deals with the conformance of measured or derived information to the defined model. The model should define the range a context value can take, or forbid a particular set of values to co-exist.

- Relationships and dependencies: The model should capture various relationships that exist between types of contextual information to ensure correct behavior of the application such as modeling a dependency where one piece of contextual information may depend on other contextual information. Some of the interrelationships between context can be captured at the ontology level through defined axioms and property characteristics like transitivity. However, most of the reasoning needs to be carried out at the system level with the aid of a reasoner.

- Usability of modeling formalisms: The important features of modeling formalisms are the ease with which designers can translate real world concepts into modeling constructs and the ease with which the applications can use and manipulate contextual information at runtime.

- Efficient context provisioning: The model should provide efficient access paths to contextual information. These access paths represent dimensions along which applications often select contextual information, typically supported by indexes. These dimensions are often referred to as primary context, in contrast to secondary context which is accessed using the primary context. Commonly used primary context attributes are the identity of context objects, location, object type, time, or activity of user.

Other requirements are those that are usually fulfilled at the system implementation rather than modeling level. These include support for context history and prediction as well as reasoning and inference of higher-order context from lower-order context produced by sensors. The former can be implemented by logging timestamped contextual

information and applying learning algorithms like Hidden Markov Models, classification or clustering techniques to it. Logging can be done with varying degrees to handle large volumes of contextual information. The latter can be implemented using a reasoning engine built on top of a rule based engine like Jess [67].

4.2.3 Supported capabilities for Context Model Ontologies

Since ontologies are the most powerful and common tool for modeling context and are used extensively in context models, Krummenacher and Strang [79] and Ye et al [146] proposed the following criteria to assess ontologies in context modeling, some of which are similar to the requirements for models:

- Reusability: Reusability implies maximizing the usage of the ontology among several independent modeling tasks.
- Flexibility and extensibility: It should be possible to add new definitions to the ontology without altering the existing ones.
- Generality: Ontologies that are applicable across several domains are referred to as upper ontologies and belong to the most general category of ontologies. Thus, the context ontology should not restrict the application domain.
- Granularity: A fine-grained ontology defines concepts for closely related objects in a specific domain, while a coarse-grained ontology has more general and distinguishable terms. Thus, granularity is related to the diversity and coverage of individual concepts. Upper ontologies are often coarse-grained, while application

ontologies are more fine-grained.

- Consistency: The ontology should be consistent and explicit or implicit contradictions in the represented ontological content should not exist.
- Completeness: An ontology is complete if it covers all possible aspects or concepts of a restricted target domain.
- Language and Formalism: This criterion looks at the language used to model the ontology and its expressiveness. Some of the popular choices are OWL and UML.
- Clarity, Coherence and Redundancy: Ontological concepts must be unique, unambiguous and distinguishable from one another through their properties and constraints.
- Encoding bias: Ontologies should be specified at the knowledge level without depending on a particular symbol-level encoding, such as the representation of date in a particular format. This is akin to the support for Multiple representations of context or Comparability requirements mentioned earlier.
- Orthogonality: General concepts should be defined as independent and loosely coupled atomic concepts.

One key criterion missing here is interoperability. Since several existing projects use standard upper ontologies, every generic ontology should be interoperable that is, its term definitions must be consistent with other standard generic and consensus ontologies such as SOUPA [28]. Also, security and privacy are of prime concern in context modeling for pervasive computing environments [146].

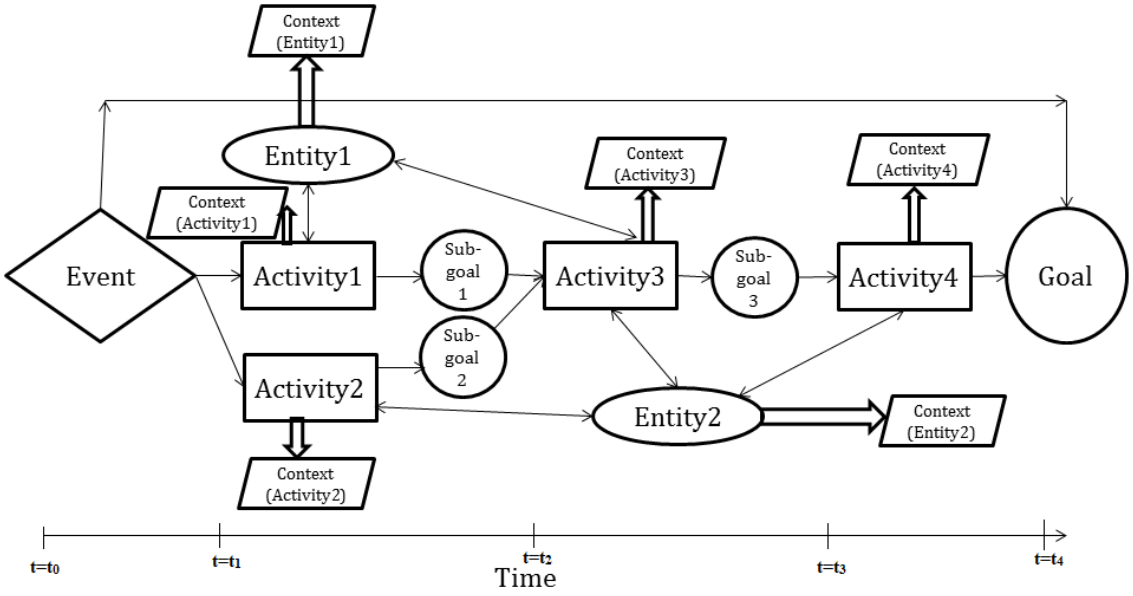

Figure 4.1: Interaction between the primitives

4.3 Design and Implementation of RoCoMO

RoCoMO is currently being developed in OWL2 DL [1] and has two components:

- RoCoM Core Ontology which is divided into two upper level ontologies:

  1. RoCoM Domain ontology - This ontology includes concepts that characterize the knowledge of the domain i.e. *Entities* and *Events*.

  2. RoCoM Task ontology - This ontology characterizes the problem solving structure of the domain and provides primitives for describing the problem solving process i.e. *Activities*.

- RoCoMO Application Ontology which has concepts that will extend the core ontology concepts but are specific to the applications.

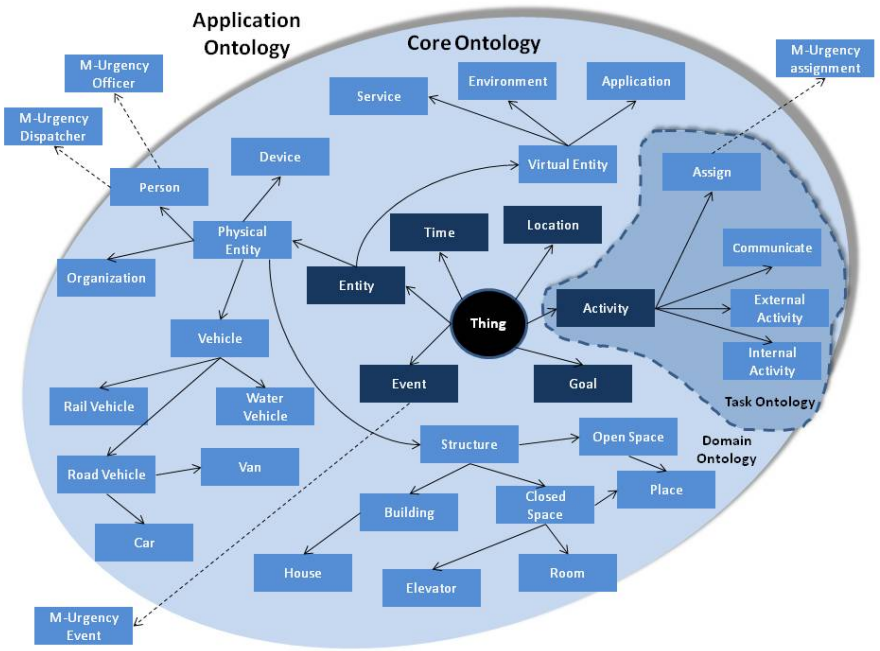

Figure 4.2: Partial view of The RoCoM Ontology Version 1.0

Figure 4.1 shows the interaction between the different primitives that represent a situation. The *Event* catalyzes the context-aware system and sets the *Goal* for it. To achieve the goal, different *Entities* - Entity1 and Entity2 perform a sequence of *Activities*. Every activity is driven by its own goal (which can be the goal that is set by the event for the context-aware system or a sub-goal). It has a start time, an end time and a duration associated with it (as shown on the Time scale). It has a pre-condition and a post-condition which are essentially goals that should have been met before the activity starts and once the activity ends. The activity can be atomic and non-interruptable. Every entity and activity has contextual information associated with it such as location, identity etc. The actual representation of these primitives and their contextual information is explained in more detail in Sections 4.4 and 4.5.

Figure 4.2 shows a partial view of the RoCoM Core and Application Ontologies.[1] The three primitives of RoCoM - 'entity', 'event' and 'activity' along with 'location', 'time' and 'goal' are derived from the OWL class 'Thing' and form the top level classes in the RoCoM Core Ontology. Each of the classes derived from these top level classes represents a different, unique and unambiguous concept in the ontology. The fourth primitive 'relationship' can be represented in OWL in many ways: between two classes (as a subclass/superclass), a class and an individual (as a member) or a specific relationship between two individuals (as object properties). The context of any element - entity, event or activity is represented using datatype properties in OWL. The following shows an OWL code snippet for the description of an individual instance named "xyz" of class 'person' which has a relationship with another entity (object property termed 'daughter') and contextual information such as food preference (datatype property termed 'likesfood').

⟨Class rdf:about="&person;person"⟩

⟨ rdfs:label xml:lang="en"⟩person⟨/rdfs:label⟩

⟨rdfs:subClassOf rdf:resource="http://mind7.cs.umd.edu:8134/Rover/physicalentity#physicalentity"/⟩

⟨/Class⟩

⟨ObjectProperty rdf:about="&person;daughter"⟩

⟨rdf:type rdf:resource="&owl;FunctionalProperty"/⟩

⟨rdf:type rdf:resource="&owl;InverseFunctionalProperty"/⟩

⟨rdfs:label xml:lang="en"⟩daughter⟨/rdfs:label⟩

⟨rdfs:subPropertyOf rdf:resource="&person;contact"/⟩

⟨inverseOf rdf:resource="&person;father"/⟩

⟨inverseOf rdf:resource="&person;mother"/⟩

⟨/ObjectProperty⟩

⟨DatatypeProperty rdf:about="&person;likesfood"⟩

⟨rdfs:label xml:lang="en"⟩likesfood⟨/rdfs:label⟩

⟨rdfs:subPropertyOf rdf:resource="&person;likes"/⟩

[1]The complete ontology can be found at http://mind7.cs.umd.edu:8134/Rover

⟨/DatatypeProperty⟩

⟨NamedIndividual rdf:about="&person;xyz"⟩

⟨rdf:type rdf:resource="&person;person"/⟩

⟨rdfs:label xml:lang="en"⟩xyz⟨/rdfs:label⟩

⟨person:likesfood rdf:datatype="&xsd;string"⟩indian⟨/person:likesfood⟩

⟨/NamedIndividual⟩

All the concepts/classes in the RoCoM Core Ontology, excluding those derived from 'activity' class, form the RoCoM Domain Ontology while 'activity' and its derived classes form the RoCoM Task Ontology. The core ontology can be extended to concepts specific to an application, such as M-Urgency [84], to form a part of the Application Ontology. M-Urgency is a public safety application that enables mobile users to stream live video from their devices to local PSAP (Public Safety Answering Point) along with the audio stream, the real time location information and any personal relevant information about the caller. A simple M-Urgency scenario can involve the entities (corresponding RoCoMO classes in parentheses): caller ('person'), dispatcher ('murgencydispatcher') and responder ('murgencyofficer'). For instance, because of an accident that is an event ('murgencyevent'), a series of activities follow such as, the caller calls the police, the dispatcher accepts the call, the dispatcher assigns ('murgencyassignment') a responder or officer to the call etc. This is only an illustration of how the concepts in the core ontology can be extended to model concepts specific to an application.

4.4 Analysis and Evaluation of RoCoMO

As discussed in Section 4.2, there are several critical requirements for context models and context model ontologies to function in pervasive computing environments. Some of

these criteria are similar or related. Here, we assess RoCoM and RoCoMO on the basis of those criteria that are most representative and explain how it addresses them:

- *Representation of static and dynamic information*: Every element of the RoCoM ontology, beginning with the top level classes like 'entity', 'activity' and 'event', have their contextual information separated into two hierarchies - static and dynamic. This enables distinguishing between contextual information that is persistent over a long period of time (static) and that which needs to be updated frequently based on its freshness (dynamic).

- *Representation of temporal characteristics of primitives*: For every primitive such as an entity, activity or event , we have defined a time class that records properties like its *start time* - time at which the event/activity started or the entity came into being, *end time* - time at which the event/activity ended or the entity ceased to exist (it is equivalent to the current time if the individual still exists), *duration* or life time of an individual (it is the difference of the start time and the end time) and *recurrence* - frequency of repetition.

⟨Class rdf:about="http://mind7.cs.umd.edu:8134/Rover/time#time"⟩

⟨/Class⟩

⟨owl:DatatypeProperty rdf:about="http://mind7.cs.umd.edu:8134/Rover/time#startTime"⟩

⟨rdfs:domain rdf:resource="http://mind7.cs.umd.edu:8134/Rover/time#time/⟩

⟨rdfs:range rdf:resource="&xsd;dateTime"/⟩

⟨rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/⟩

⟨/owl:DatatypeProperty⟩

.....

⟨owl:DatatypeProperty rdf:about="http://mind7.cs.umd.edu:8134/Rover/time#repetition"⟩

⟨rdfs:range:resource="&xsd;string"/ ⟩

⟨rdfs:subPropertyOf rdf:resource=”&owl;topDataProperty”/ ⟩

⟨ /owl:DatatypeProperty ⟩

- *Timestamping*: Timestamping the dynamic contextual information allows the system to determine the freshness of the contextual information. In RoCoMO, the contextual information is timestamped at two levels; (i) *fine-grained level* - timestamping every dynamic contextual information of an individual instance of a primitive to keep track of when it was last modified and by which entity and (ii) *coarse-grained level* - timestamping the individual instance itself to determines when it was modified and by which entity. The ontology snippet below shows how the *hasMood* context of an individual *xyz* of the *person* class in the ontology is assigned a value *happy* and is timestamped to determine when the value got updated.

⟨Axiom⟩

⟨annotatedTarget rdf:datatype=”&xsd;string”⟩"happy"⟨/annotatedTarget⟩

⟨rocomo-schema:timeStamp rdf:datatype=”&xsd;dateTime”⟩2012-09-18T14:00:00⟨/rocomo-schema:timeStamp⟩

⟨annotatedProperty rdf:resource=”&person;hasMood”/⟩

⟨annotatedSource rdf:resource=”&person;xyz”/⟩

⟨/Axiom⟩

- *Machine-interpretable representation of contextual information, Efficient context provisioning and Granularity of context*: RoCoMO is implemented in OWL2 DL which is expressive and allows more versatile knowledge representation. In OWL, context can be represented as annotated semantics via data properties and relationships between different elements can be represented via object properties. It also enables automatic context reasoning. Also, OWL represents information hierarchi-

cally which allows efficient provisioning of context and representation at multiple levels of abstraction or granularity.

- *Multiple representation of context, Encoding bias and Comparability*: We annotate any measurable contextual information with an annotation property, called 'unit', defined by us in a RoCoMO schema[2]. This removes the model's dependency on any particular encoding or measurement unit and also facilitates comparison or conversion from one unit to another. For instance, a person's context can include his/her height which can be in feet or meters or any other unit. Thus, for person "xyz", we can represent height and its measurement unit as:

⟨Axiom⟩

⟨annotatedTarget rdf:datatype="&xsd;float"⟩6.0⟨/annotatedTarget⟩

⟨rocomo-schema:unit rdf:datatype="&xsd;string"⟩feet⟨/rocomo-schema:unit⟩

⟨annotatedProperty rdf:resource="&person2;height"/⟩

⟨annotatedSource rdf:resource="&person2;xyz"/⟩

⟨/Axiom⟩

- *Quality of Context (QoC)*: The intended purpose of our model and ontology is to not deal with raw sensor data directly but rather with the high-level context that has been obtained from it. Hence, it becomes extremely important to annotate the contextual information with some QoC attributes to determine imperfection or uncertainty in it that might have been introduced. Several papers including Gray and Salber [111] introduced the notion of attaching information quality attributes to every piece of sensed context. On the same lines, we have defined seven QoC attributes that model imperfection in contextual information - *accuracy* to represent

[2]The schema can be found at http://mind7.cs.umd.edu:8134/Rover/rocomo-schema

correctness, *probability or confidence* to represent the certainty, *coverage* to represent the range, *resolution* to represent the smallest perceivable element, *meanError* to represent average error, and *recurrence* to measure repeatability. These annotations are defined in the RoCoMO schema and can be attached to the appropriate contextual information or a relationship and can be propagated to applications. For instance, a person's context can include his/her weight which can be in kgs, pounds or any other unit. Also, the weight measure can have a mean error attached to it depending on the sensitivity of the instrument. Thus, for person "xyz", we can represent weight, its measurement unit and its mean error as:

⟨Axiom⟩

⟨annotatedTarget rdf:datatype="&xsd;float"⟩55.0⟨/annotatedTarget⟩

⟨rocomo-schema:unit rdf:datatype="&xsd;string"⟩kgs⟨/rocomo-schema:unit⟩

⟨rocomo-schema:meanError rdf:datatype="&xsd;float"⟩1.0⟨/rocomo-schema:meanError⟩

⟨annotatedProperty rdf:resource="&person2;weight"/⟩

⟨annotatedSource rdf:resource="&person2;xyz"/⟩

⟨/Axiom⟩

- *Provenance and Traceability*: Provenance and traceability in RoCoMO is done at a coarse-grained level where we store when the contextual information of any instance or an individual was created, when was it last modified, the last modification made to the instance and the entity by which it was made. However, we are not tracking every single modification made to every unique attribute or contextual information since this is too cumbersome at the modeling level. This can be achieved at the system level by logging context history and transformations. For instance, in our case study, we require the temperature of a room along with the source of

the reading, its measurement unit, its time stamp and its probability. Thus, the following OWL snippet represents an instance of a temperature reading of 100 degree Fahrenheit with probability 0.9, created by a sensor instance 'sensor1' at 2 pm on September 18th 2012.

```
⟨owl:NamedIndividual rdf:about="&environment;envreading1"⟩
    ⟨rdf:type rdf:resource="&environment;environment"/⟩
    ⟨rdfs:label xml:lang="en"⟩envreading1⟨/rdfs:label⟩
    ⟨environment:temperature rdf:datatype="&xsd;float"⟩100.0⟨/environment:temperature⟩
    ⟨entity:createdBy rdf:resource="http://mind7.cs.umd.edu:8134/Rover/sensor#sensor1"/⟩
⟨/owl:NamedIndividual⟩
⟨Axiom⟩
    ⟨rocomo-schema:probability rdf:datatype="&xsd;float"⟩0.9⟨/rocomo-schema:probability⟩
    ⟨rocomo-schema:timeStamp rdf:datatype="&xsd;dateTime"⟩2012-09-18T14:00:00⟨/rocomo-schema:timeStamp⟩
    ⟨owl:annotatedTarget rdf:datatype="&xsd;float"⟩100.0⟨/owl:annotatedTarget⟩
    ⟨rocomo-schema:unit rdf:datatype="&xsd;string"⟩Fahrenheit⟨/rocomo-schema:unit⟩
    ⟨owl:annotatedSource rdf:resource="&environment;envreading1"/⟩
    ⟨owl:annotatedProperty rdf:resource="&environment;temperature"/⟩
⟨/owl:Axiom⟩
```

- *Heterogeneity, Mobility and Context Information Partitioning*: RoCoMO is an ontological model and promotes knowledge sharing and reuse across distributed systems and applications in pervasive computing environments. Hence, even if the sources of context are heterogeneous, distributed and partitioned, the contextual information can be shared and aggregated across environments.

- *Ease of development*: RoCoMO is developed on the principle of Model-Driven Development as we designed RoCoM first. The ontology is also available publicly. Hence, developers have adequate support for development and implementation.

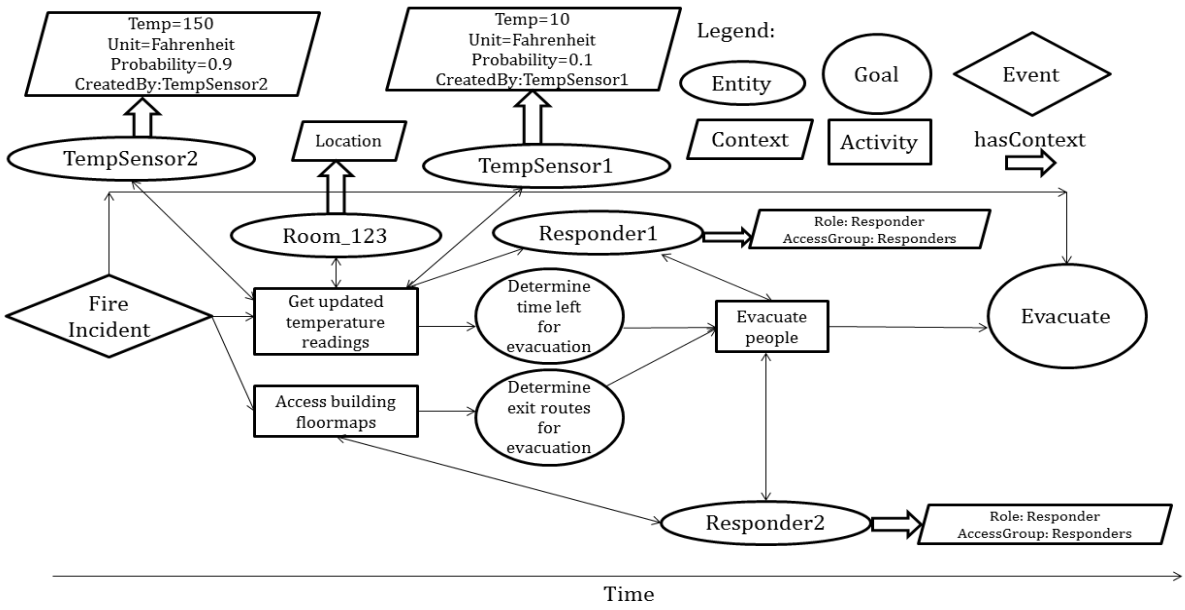

Figure 4.3: Case Study

- *Flexibility, extensibility, applicability, generality, evolvability and completeness*: RoCoMO is structured in a modular fashion with clear distinction between the Core and Application ontologies. Also, as the applications evolve, more concepts can be added to it. Thus, it is easily extensible, flexible and evolvable. It does not target any specific domain and is intended to be general and applicable across several applications and domains. As a result, we do not claim that the ontology is complete.

- *Interoperability*: It is highly desirable than an ontology be interoperable so that it can be flexible and accommodative. It also enables reuse of domain knowledge [55]. We have designed RoCoMO to be interoperable with other ontologies. For example, RoCoMO is interoperable with SOUPA [28], via the *equivalentClass* and *equivalentProperty* OWL statements. The example below shows that the RoCoMO *person* class is defined equivalent to the *person* class in SOUPA and the *dateofbirth*

property is defined equivalent to *birthDate* property in SOUPA.

```
⟨Class rdf:about="&person;person"⟩
    ⟨rdfs:label xml:lang="en"⟩person⟨/rdfs:label⟩
    ⟨equivalentClass rdf:resource="http://pervasive.semanticweb.org/ont/2004/06/person#person"/⟩
    ⟨rdfs:subClassOf rdf:resource="http://mind7.cs.umd.edu:8134/Rover/physicalentity#physicalEntity"/⟩
⟨/Class⟩
⟨DataProperty rdf:about="&person;dateofbirth"⟩
    ⟨rdfs:label xml:lang="en"⟩dateofbirth⟨rdfs:label⟩
    ⟨rdfs:subClassOf rdf:resource="&person;personalinfo"/⟩
    ⟨equivalentProperty rdf:resource="http://pervasive.semanticweb.org/ont/2004/06/person#birthDate"/⟩
⟨/DataProperty⟩
```

- *Security and Privacy*: These are implemented in RoCoMO using the concept of groups and members. A group is an instance of type 'accessgroup' class. This class has object properties like 'groupmember' which includes the entities like users or devices that can be assigned to an instance of the group. The 'accessgroup' class also has an object property called 'privileges' which defines the permissions that the group can have. These permissions can be in the form of an 'activity' that the group is allowed to perform. Every entity can belong to multiple access groups while each access group can have multiple entities and privileges. This form of Role Based Access Control (RBAC) obtained by assigning users to groups and granting privileges to groups rather than individual users reduces the number of associations involved that must be managed. Hence, it is easier to define security policies around this framework.

4.5 Fire Incident Case Study revisited

In this section, we revisit the Fire Incident case study from Section 4.1 and illustrate how RoCoMO can be used to model it. Figure 4.3 shows the situation modeled in RoCoMO. The *Fire Incident* Event triggers the situation that follows and sets the Goal *Evacuate*. This goal can be subdivided into smaller sub-goals which can be performed by one or more activities. Entity *Responder1* performs the Activity *Get updated temperature readings* of getting the context information of the room - the updated temperature readings from the temperature sensors *TempSensor1* and *TempSensor2*. The Goal for this activity is to *Determine time left for evacuation*. Since the information is timestamped, the system can refresh it periodically based on its freshness and this resolves any ambiguity. The contextual information also has a probability measure attached to it. As shown in the figure, the probability of temperature from *TempSensor1* is 0.1 which means it is highly unreliable and that the sensor could be faulty. This is also evident by the fact that it shows a reading of 10 deg Fahrenheit while the other sensor shows a reading of 150 deg Fahrenheit. Also, the contextual information has source or provenance information attached to it and so this determines which reading came from which sensor. Based on the temperature reading and its encoding (Fahrenheit in this case), the system can calculate how much time it will take till the temperature reaches the value at which the building bursts into flames. This is the amount of time that the responders have for evacuation. Simultaneously, another entity *Responder2* is performing the activity *Access building floormaps* with the Goal of *Determine exit routes for evacuation*. Since the responders belong to the AccessGroup *responders*, the system checks their privileges (same as the privileges

granted to the ‘responders’ accessgroup) and grants access to the temperature readings from the sensor and building floor plans. Once these two activities have achieved their goals, both the responders start the activity *Evacuate people* and achieve the goal set by the event.

Here are selected OWL snippets that represent this situation (in addition to the ones mentioned in Section 4.4). The activity, its goal and its start time can be represented as :

⟨owl:NamedIndividual rdf:about=”#determineEvacuationTime”⟩

⟨rdf:type rdf:resource=”&goal;goal”/⟩

⟨rdfs:label xml:lang=”en”⟩determineEvacuationTime⟨/rdfs:label⟩

⟨goal:hasDescription rdf:datatype=”&xsd;dateTime”⟩Determine time for evacuation⟨/goal:hasDescription⟩

⟨/owl:NamedIndividual⟩

⟨owl:NamedIndividual rdf:about=”#getTempReadings”⟩

⟨rdf:type rdf:resource=”http://mind7.cs.umd.edu:8134/Rover/activity#activity”/⟩

⟨rdfs:label xml:lang=”en”⟩

getTempReadings⟨/rdfs:label⟩

⟨time rdf:resource=”#getTempReadingsTime”/⟩

⟨/owl:NamedIndividual⟩

⟨owl:NamedIndividual rdf:about=”#getTempReadingsTime”⟩

⟨rdf:type rdf:resource=”&time;time”/⟩

⟨rdfs:label xml:lang=”en”⟨getTempReadingsTime⟩/rdfs:label⟩

⟨time:startTime rdf:datatype=”&xsd;dateTime”⟨2012-09-21T14:32:12⟩/time:startTime⟩

⟨/owl:NamedIndividual⟩

We can define an instance ‘Responder1’ of type ‘person’ and an instance ‘Responders’ of type ‘accessgroup’. The ‘Responders’ access group has access to the perform the activity of determining the evacuation time by accessing the updated temperature readings:

⟨owl:NamedIndividual rdf:about=”#Responder1”⟩

⟨rdf:type rdf:resource=”&person2;person”/⟩

⟨rdfs:label xml:lang=”en”⟩Responder1⟨/rdfs:label⟩

⟨physicalentity:memberOf rdf:resource=”#Responders”/⟩

⟨/owl:NamedIndividual⟩

⟨owl:NamedIndividual rdf:about=”#Responders”⟩

⟨rdf:type rdf:resource=”&accessgroup;accessgroup”/⟩

⟨rdfs:label xml:lang=”en”⟩Responders⟨/rdfs:label⟩

⟨accessgroup:privileges rdf:resource=”#getTempReadings”/⟩

⟨/owl:NamedIndividual⟩

Chapter 5

Rover-II - the Middleware

Middleware is considered an indispensable component of context-aware environments. Ranganathan et al [102] argue that ubiquitous computing environments must provide support for middleware. This is because the middleware would provide uniform abstractions and reliable services for common operations and would simplify the development of context-aware applications. It would be agnostic to hardware, operating system and programming language. It would also allow us to compose complex systems based on the interactions between a number of heterogeneous and distributed context-aware applications. More importantly, it would provide support for complex tasks such as acquisition of contextual information, reasoning about context using mechanisms like rule based or temporal or spatial reasoning as well as learning from context using mechanisms like Bayesian networks, neural networks, reinforcement, supervised and unsupervised learning and modifying its behavior based on the current context. It would also define a common model of context which will ensure that different applications in the ubiquitous environment have a common semantic understanding of contextual information. They also specify certain requirements for middleware for context-aware systems in ubiquitous environments, which in today's terms mean:

1. It should support collection of context information from heterogeneous sensors and services and the delivery of appropriate context information to different applica-

tions.

2. It should support inference of higher level contexts from low level sensed contexts

3. It should provide tools for different kinds of reasoning and learning mechanisms

4. It should allow applications to behave differently in different contexts easily

5. It should enable syntactic and semantic interoperability between different applications and services (through the use of ontologies)

In this chapter, we present the design and architecture of a context-aware middleware, Rover II, which serves as an integration platform for mobile and desktop applications. It provides a means to store and retrieve contextual information, and also facilitates providing relevant services to the applications so that the contextual information can be used more effectively. It communicates with third party services and external databases for gathering contextual information, consolidating it and presenting it to client applications in a pertinent way. It is also designed to have advanced context usage functionalities such as learning from past context (context history) and context reasoning. It is based on the our underlying ontological context model, Rover Context Model (RoCoM), which enables knowledge representation, sharing and reuse.

Rover II is designed to support human decision making, keeping three essential features of context-aware systems in mind, namely customization, adaptability and interactivity [6]. The context model described above allows the customization through user-specific context. With the idea of relevant context comes adaptability. It that seeks to provide a mechanism to aid communication of heterogeneous information to all inter-

ested and authorized entities. Additionally, entities utilizing it would themselves serve as contextual information sources. Since, the system is designed to aid a human decision maker, it constantly interacts with him/her through a UI console.We have designed the architecture of Rover II keeping three essential features of context-aware systems in mind, namely customization, adaptability and interactivity [6]. The system will support human decision making. Thus, the final decision of taking any action, based on the contextual information provided by Rover II, rests with the human decision maker.

Rover II is a major advancement based on the underlying concepts of Rover [6], a context-aware server that caters to the development of context-aware mobile applications. Rover II extends the Rover concept with RoCoM and features such as context reasoning and learning. Rover did not have an explicit context model and followed the RDF approach [2] for context representation and storage where every piece of contextual information was represented as a triplet:

$$\text{triplet} \leftarrow \langle \text{ subject, predicate, object } \rangle$$

Most of the proposed systems in the literature are user centric, and the context of a situation is defined only with respect to the entities involved in it. Rover considers (in addition to the entities), events, activities, their properties, and the relationships between them. Moreover, most of the systems and middleware proposed are too specific to a particular domain in their implementation, or are general but only conceptual. To address these shortcomings, we have developed Rover II which is being designed to support all the requirements of middleware in ubiquitous environments as mentioned in [102] - It supports integration of different mobile applications and collection and storage of in-

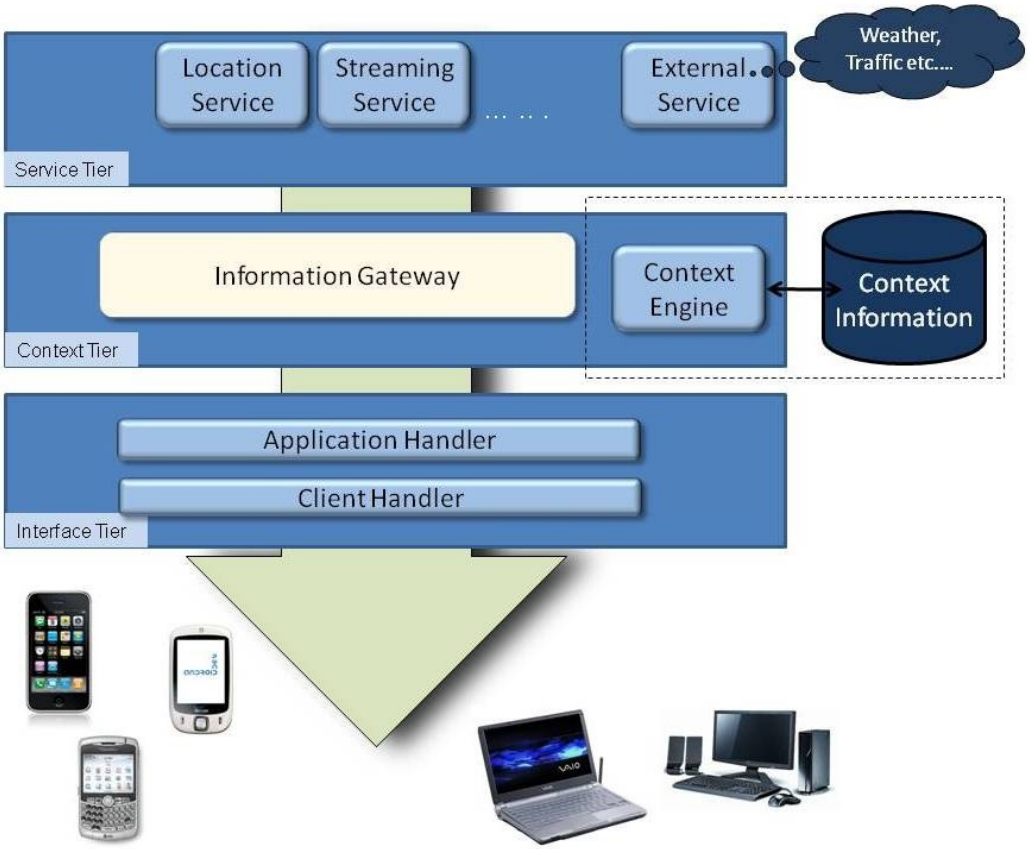

Figure 5.1: Rover-II ecosystem architecture

formation from heterogeneous services, provides support for inference, reasoning and learning and uses an ontological context model (RoCoM) to allow a common semantic framework for representing context.

Figure 5.1 provides a high level layout of the Rover II ecosystem architecture. An ecosystem here is a logical view of how different entities interact with the context-aware system. This architecture is centered on the information flow between the various sources and the end application or the user. The architecture is organized in three tiers:

- Service tier,

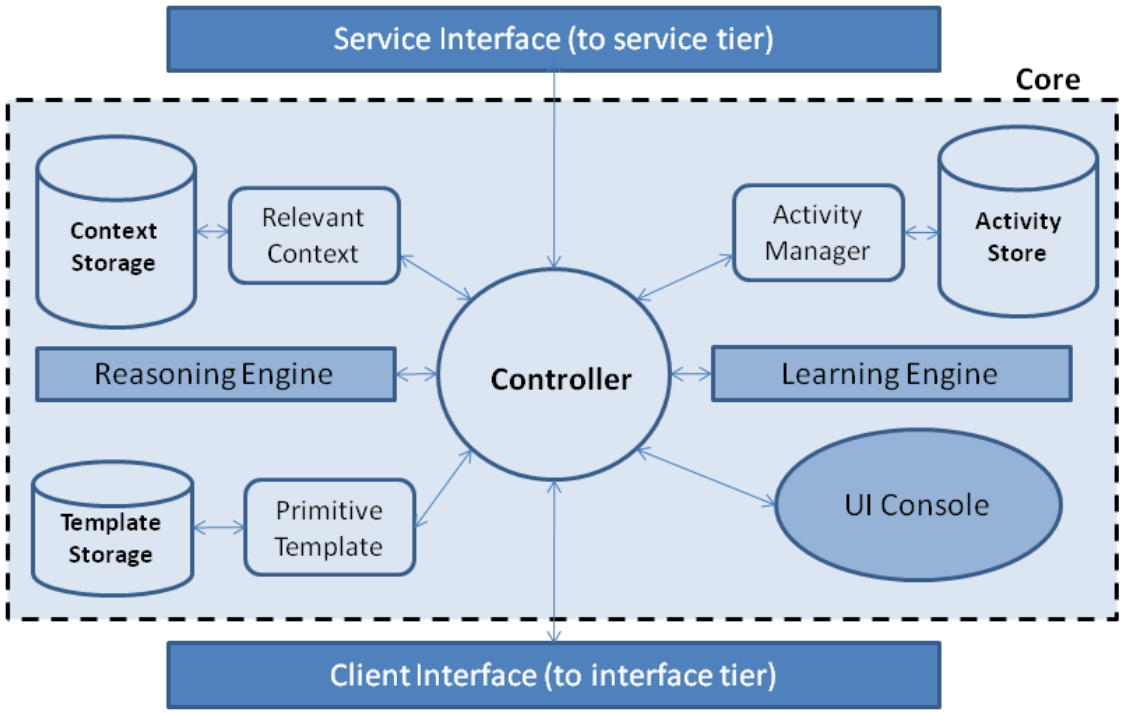

Figure 5.2: Rover-II server architecture overview, detailed design

- Context tier - the Core, and
- Interface tier.

Each of the tiers of the Rover-II architecture are explained in detail in the following sections.

5.1 Context Tier - the core

This is the critical part of the system which makes use of the contextual information of each user to mediate the flow of information from the source to the client/user. Based on the context of the user, along with the general context of the environment, the information is filtered, consolidated and/or rearranged, and some services enabled.

Figure 5.2 depicts the detailed design of the Rover-II Context Tier Core. The three layers are:

1. *Service Interface* - This layer interfaces with third party services that could be Web Services, REST based services, etc. It will also provide CRUD operations for external databases.

2. *Client Interface* - This layer interfaces with the client applications. The interaction can be through TCP sockets or through Remote Procedure Calls.

3. The Rover Core - This layer forms the core layer of the Rover Server. It consists of several modules that handle and propagate the context:

   (a) The *Controller* is the main kernel, which schedules different processes running inside the Rover Core and passes around the context between modules.

   (b) The *Context Engine* determines the relevant context for each primitive, based on predefined primitives and context templates stored in the Template Store.

   (c) The *Template Manager* is used to fetch the context and primitives templates from the Template Store.

   (d) The *Reasoning Engine* module will perform reasoning about complex situations and inference based on pre-defined rules.

   (e) The *Learning Engine* learns about application and user behavior from past context (context history) in order to determine patterns and predict user behavior.

(f) The *Activity Manager* is responsible for the execution of all the activity(s) that form an event, until the Goal of the event or activity has been achieved.

(g) The *UI Console* is the user interface for a human entity, for making decisions based on the contextual information provided by the system.

(h) The *Context Store* contains the aggregation of context for every instance of primitive whether an entity or relationship, etc.

(i) The *Template Store* contains the primitives and context templates.

When a client application or device initiates a session with the Rover Core; an Event or Activity primitive is instantiated, from its corresponding template, for that client depending on the application signature. The Event template will contain the Goal for that event. The Controller obtains an instance of the template for the primitive (activity or event) from the Primitives Templates module which fetches it from the Template Store and instantiates it. The template may require other primitives such as sub-activities, entities and relationships, which form a part of the event or activity, to be instantiated as well. Once the instances have been created, the Activity Manager module is triggered. The Activity Module starts managing the activities and the entities involved in those activities to complete the designated goal as defined by the Event. The Activity Manager invokes the Relevant Context module and obtains the Relevant Context for each of the primitives in the activity from the Context Store. Though the *Learning Engine* has not been currently implemented, the Rover-II design and implementation has been executed taking this module into account. We discuss below, more in detail, the functionality and implementation of the important components in the Rover-II core.

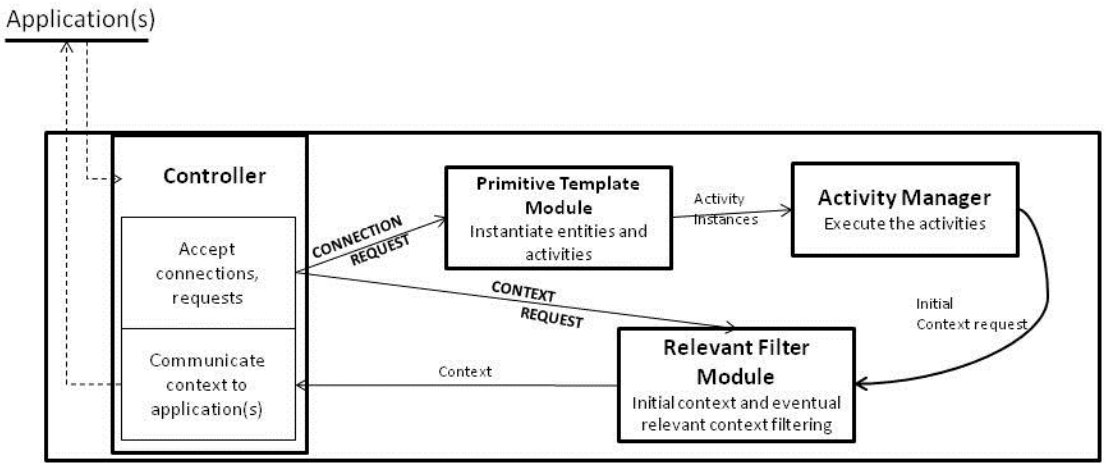

Figure 5.3: A typical sequence in Rover-II core

5.1.1 Controller

The *controller* is the central point of Rover-II core. Its role ranges from accepting the incoming connections from client applications, through the *interface tier*, to coordinating between the *Activity Manager*, *Primitive Template Module* and the *Relevant Context Module* in order to cater to facilitate the exchange and reuse of context. Though the sequence of activities would differ for each application, Figure 5.3 shows the sequence in a typical case. The Controller receives a connection request from the application and retrieves the initial set of activities to be performed for the client application to achieve the set Goal. It invokes the required sub-modules in Primitive Template module to instantiate and buffer the contextual information from the store for quick access. As the situation evolves, it invokes required sub-modules in the Relevant Context module to appreciate the contextual information accordingly.

5.1.2 Activity Manager

The *activity manager* maintains a list of activities that take place as a consequence of an event, and to achieve a desired goal. It also maintains a list of entities involved in those activities. The machine understandable instructions are maintained as part of the Rover Context Model Ontology (RoCoMO) [20] [21]. This module executes the sequence of activities such as facilitating connections between client applications and third party services that are a part of the service tier. It also handles the individuals of the involved entities and activities that are are instantiated by the Primitive Templates module.

5.1.3 Primitives Templates Module

This module receives a list of entities that are involved in the trigger event as well as the subsequent activities that will be performed by the Activity Manager, and instantiates them. To make data access efficient and quick, along with instantiating the entity, the primitive template module instantiates all the entities directly related to the same. The instantiated contextual information are indexed with the session ID associated with the application. Thus, the instance of a person connecting through application A will not interfere the context of application B, involving the same person.

5.1.4 Relevant Context Module

In this section we discuss the conceptual basis and implementation of *relevant context*, taking context of the context into account. An exhaustive storage of contextual information regarding every entity will be available in Rover II. Not every piece of contextual

information will be relevant in every situation. For example, the professional information of a person involved in a road accident is of no value; rather, the person's medical record and personal information will be relevant context in that situation.

The *relevant context module* is instrumental in performing the process of *relevance filtering* as a situation evolves. Based on the feedback from the applications, sensors and other data sources, Rover-II keeps track of how a situation, involving the application, evolves. As these feedbacks are received, appropriate relevance filtering is performed on the available pool of context to produce the *relevant context* to the given situation. Relevance filtering is performed with the help of *relevant filters*, comprising of appropriate *relevant predicates*, involving filtering, augmentation, aggregation, transformation etc. We discuss this module in detail in chapter 6.

5.1.5 User Interface Console

The User Interface console acts as the front end for Rover-II, as it functions. It updates the user with the state of the middleware textually and with some basic visualizations. Figure 5.4 shows a screenshot of the Rover-II UI console. It consists of a text area, on the left, that is updated with all the sequences of execution taking place in the Rover-II core. Each module in the core updates the console with their respective actions. The drop down box, on the top of the console, enables the user to select a particular session ID. Every event has an associated session, which is typically initiated by an application. All the contextual information associated with that event are indexed by the session ID. If more than one application or users are associated with a session, the contextual information

is shared between them. Thus, when a session ID is selected from the drop down box, the *situation graph* of that particular session is displayed in the pane on the center of the UI console. Details of situation graph are discussed in chapter 6. This pane provides an interactive visualization of the situation graph, color coding the Event/Activity graph nodes and Relevant Context Graph (also discussed in chapter 6) nodes, to differentiate between them. The user is provided with the pan and zoom capabilities on the graph visualization along with the capability to reorganize the graph through drag operations. This gives the user flexibility to understand the situation, better. The text area on the right side of the console provides the list of users currently connected to Rover-II. The application ID and the session ID associated with that user are also listed.

We made use of Prefuse - interactive information visualization toolkit [132] to develop the visualization of the situation graph. The UI, otherwise, was developed using Java Swing components. We have provided only a basic user interface here and believe an extensive UI would more effective.

5.2 Service Tier

We believe the abundant contextual information available will be effective only when it has clarity and is presented efficiently. For example, location information with a lat/long value text may not make much sense to a user, but can make a lot of sense is the same is translated to a pointer on a geographical map interface. The idea behind the service tier to to provide and make use of such services that help enhance the contextual information, adding clarity to it, by combination of information and by effective presen-

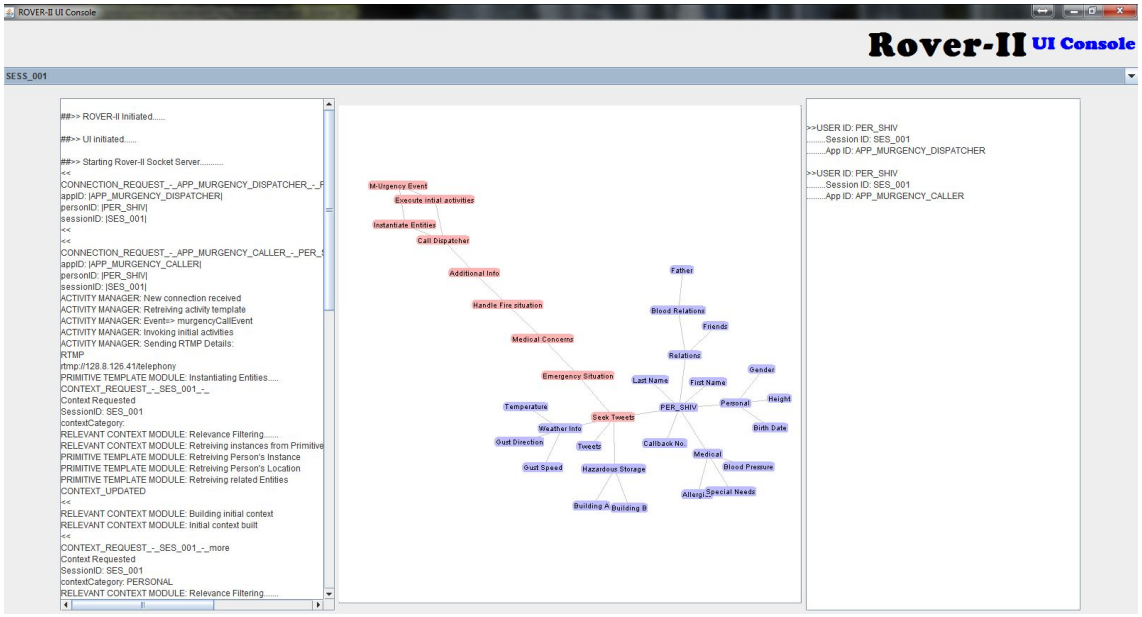

Figure 5.4: Screenshot of the Rover-II User Interface Console

tation/visualization. This tier consists of several services, developed by us or by external/third party, with which the system can communicate to obtain possibly unstructured information. Figure 5.1 shows some of the services that have been implemented in the system to date. We discuss below some of the main services the tier comprises of. We are in the process of adding more services to this tier.

5.2.1 Location Service

Location is increasingly important for mobile computing, providing the basis for services such as navigation and location-aware advertising. The most popular technology for localization is GPS, which provides worldwide coverage and accuracy of a few meters depending upon satellite geometry and receiver hardware. Its major shortcoming is that it is reliable only in outdoor and environments with direct visibility to at least four

GPS satellites. For indoor environments, alternative technologies are required. The holy grail in indoor location technology is to achieve the milestone of combining minimal cost with accuracy, for general consumer applications. A low-cost system should be inexpensive both to install and maintain, requiring only available consumer hardware to operate and its accuracy should be room-level or better. To achieve this level of accuracy, current systems require either extensive calibration or expensive hardware. Most of them are based primarily on either time or signal strength information. A third alternative, angle-of-arrival information is useful in outdoor environments but is not generally helpful indoors due to obstructions and reflections. Time-based systems require hardware support for timestamping that is not available in consumer products.

The use of wireless received signal strength indicators or RSSI values for localization of mobile devices is a popular technique due to the widespread availability of wireless signals and the relatively low cost of implementation.(We use RSSI and signal strength interchangeably). Its simplest version involves the mobile device measuring the signal strengths of existing infrastructure points such as Wi-Fi access points (APs) or mobile phone base stations and reporting the origin of the strongest signal it can hear as its location. This technique may be applied both to short range communications technologies such as RFID or Bluetooth as well as longer range technologies such as Wi-Fi or mobile phones but its performance is directly linked to the density of reference points. The precision of signal strength approaches is improved to meter-level by fingerprinting techniques, such as those in RADAR [11] or Horus [147], that use pre-measured fingerprinting radio maps. There are commercial solutions available as well such as Ekahau [42] that achieves a high precision of 1 to 3 m but requires proprietary hardware. However, a

major drawback of fingerprinting is the cost of recording the radio map; a large amount of human effort is required to record the signal strength at each desired location using a receiver. Also, if the infrastructure or environment changes significantly, for instance the locations of APs are changed, furniture is moved around, the number of people occupying the closed space increases dramatically, or the test site is changed; the radio map must be remeasured to maintain performance [12].

Systems that don't use fingerprinting techniques often suffer from very low precision. These include Active Campus [54], that uses an empirical propagation model and hill-climbing algorithm to compute location with a location error of about 10 meters, and a ratio based algorithm proposed by Li [81], which produces median errors of roughly 20 feet (6.1 m) by predictively computing a map of signal strength ratios. Lim et al [82] proposed an automated system for collection of RSSI values between APs and between a client and an AP to determine the client's location with an error of 3m. They do not create a radio map but require initial AP calibration and its modification for continuous data collection.

However, more important than the raw error in distance is the computation of the correct floor in indoor multi-story environments. Even a most modest error in altitude can result in an incorrect floor leading to a high location error as determined by human walking distance. Identifying the exact floor is also more difficult because there are multiple APs on each floor and a device can receive signals from APs across floors. To address this, we explain a heuristics based indoor localization, tracking and navigation system for multi-story buildings - *Locus* [22], that determines a device's floor and location by using the locations of infrastructure points but without the need of radio maps. It can enable

indoor location based services and applications such as a smartphone application that automatically downloads a map of a building when a user enters it, tracks his approximate current position on a floor map, and provides indoor navigation directions for destinations such as restrooms, offices, or conference rooms. It is also essential for situations like search and rescue operations where knowledge of the exact floor and location of a device/person on that floor is crucial for timely assistance.

Our system is calibration-free and is an inexpensive solution suitable for localization with minimum setup, deployment or maintenance expenses. By avoiding the dependence on radio maps, it is readily deployable and robust to environmental change. It relies on existing infrastructure and mobile device capabilities, and requires no proprietary hardware to be installed. Initial experimental results with commercial tablet devices in an indoor space spanning 175,000 square feet across multiple floors, show that our system can determine the floor with 99.97% accuracy and the location with an average location error of 7m, and with very low computational requirements. Though our system has a higher location error as compared to fingerprinting techniques, we believe it still serves as a competitive alternative particularly in scenarios where extensive fingerprinting is not feasible or affordable or it is preferable to trade a little precision for saved human effort. To the best of our knowledge, our system is the first calibration-free system for floor as well as location determination in multi-story buildings. Active Campus [54] has options for user adjustments to correct the computed floor while in [81], the testbed is assumed to be on a single floor. Skyloc [136] uses GSM based fingerprinting for floor determination only and determines it correctly in 73 % of the cases while FTrack [145] uses an accelerometer to capture user motion data to determine floor but requires user input for

initial floor.

Locus is discussed in detail in [22].

5.2.2 Media Service

The media server enables applications to avail live streaming services of audio and video. We have have made available two services provided by Adobe - Flash Media Server and Flash Media Gateway.

5.2.2.1 Flash Media Server

Flash Media Server (FMS) [122] is a proprietary data and media server from Adobe Systems. This server works with the Flash Player runtime to create media driven, multiuser RIAs (Rich Internet Applications). The server uses ActionScript, an ECMAScript based scripting language, for server-side logic. A newer version of the same was released recently, called Adobe Media Server. Applications connect to the hub using Real-Time Messaging Protocol. The server can send data to and receive data from many connected users. A user can capture live video or audio using a camera and microphone attached to a computer running Adobe Flash Player or Adobe AIR. Users connect to the server, mediated by Rover-II .

An application that runs on Flash Media Server has a client-server architecture. The client application can be developed in Adobe Flash or Adobe Flex and run in Flash Player, Adobe AIR, or Flash Lite. It can capture and display audio and video. It manages client connections and permissions, writes to the servers file system, and performs other tasks.

The client must initiate the connection to the server. Once connected, the client can communicate with the server and with other clients. More specifically, the client connects to an instance of the application running on the server.

FMS is available in different server editions. We have installed Flash Media Development Server.

5.2.2.2 Flash Media Gateway

Flash Media Gateway (FMG) [50] is a real-time server platform that enables Adobe Flash and Adobe AIR applications to connect with traditional communication devices via SIP. FMG Telephony SDK plugs in seamlessly with existing Adobe Flash Media Server server-side applications to allow quick integration of telephony related workflows, therefore, giving full control to customize the level of features and policies that application may require.

5.2.3 Mapping Service

Location is one of the basic contexts associated with any situation. We could say that literally any mobile application makes use of the current location to define the situation and its context. A map interface enables efficient reflection the current location of a user/application. The University of Maryland is developing a very detailed GIS system and interface for the campus, taking into account information ranging from buildings to even individual plants/trees.

Applications can avail these services through Rover-II. They can makes use of ESRI

[46] maps and the ArcGIS APIs to work on the same.

5.2.4 Data Service

5.2.4.1 Tweet Service

With the growing popularity of social networks, a lot of information can be gathered regarding an event/incident, to get a better idea of a given situation. We have implemented a very simple service that can read Twitter [133] tweets and gather any possible information about a given situation. Say, an emergency call is being made (as explained in chapter 8) reporting a fire incident in a particular location. The twitter data service can scan for tweets made in relation to the incident (through simple reasoning techniques based on contexts like the location, keywords etc.) and make the same available. This may help the emergency dispatcher/responder understand the situation better and act accordingly. We made use of *twitter4j* API [134], which helped us integrate our Java application with Twitter service.

5.2.4.2 Weather Service

Yet another simple service we have implemented is to gather the weather information, provided a location. Applications, that are influenced by weather, could make use of this service. We made use of WeatherBug API [9] to implement this module, gathering the weather information from the WeatherBug server.

There is a scope for many more data services that could tap useful information from the many available data sources to understand a situation better.

5.3 Interface Tier

This tier defines the application interface for clients/applications with the system. Applications could make use of defined communication mechanisms and APIs to connect/disconnect, communicate and avail the various services provided by Rover-II.

The primary mode of communication between the applications and Rover-II happens through HTTP socket communication. Applications typically begin with *connectionRequest* and end with *disconnectionRequest*. Applications maintain the communication meanwhile through *contextRequest*, *serviceRequest* etc. The media service is available through a RTMP connection. The Location Service and the Mapping service are available through HTTP connections. The compiled context information is made available to the applications through HTTP service.

Interface tier is also instrumental in the inter-Rover communication in our distributed approach. We discuss our initial study towards a distributed architecture of Rover-II in chapter 9. We performed out experiments by simulating the interface tier and designing and assessing the behavior of the system.

5.4 Implementation Details

As discussed before, Rover-II is an integration platform with the core managing the context and the service layer encompassing as many services possible. Different components were developed on a suitable platform and coded using appropriate languages. The whole idea of Rover-II is to integrate the various components into a single system. Table 5.1 lists the language used in developing the various components of Rover-II.

Rover-II (core)	Java [64]
HTTP server	Apache [123]
HTTP server scripting	PHP [98]
Handling ontologies	Java (Jena API) [65]
Tweet Service	Java (twitter4j)
Weather Service	Java (WeatherBug API)
Media Server scripting	Adobe Flex ActionScript [4]
Location Server	Java
UI Console	Java Swing
UI Visualization	Prefuse visualization toolkit

Table 5.1: Languages used to develop the various components of Rover-II

5.5 Failures, Uncertainty and Security

Though the main focus in developing Rover-II has been in handling the context in the core and the service tier yet, some basic measures have been put in place to address issues of security, failures etc.

- *Security*: Rover II addresses the security issues such as levels of access, authentication and authorization. Applications go through an authentication process to connect to Rover-II. Based on the role of the user, with respect to the application, only appropriate authorization and access to data is permitted. Appropriate firewall mechanisms have been put in place. The security measures are not very extensive, but sufficient enough to provide a basic level of security. Consider M-Urgency (chapter 8) system, developed on Rover-II. It is a public safety system deployed at the University of Maryland Police Department which demands that security issues be addressed.

- *Failure*: Failures, with respect to Rover-II can be considered at various levels:

  - The case where the Rover-II server itself fails. We have made initial efforts in designing and implementing a distributed architecture for Rover-II which we discuss in detail in chapter 9. This addressed the issue of load balancing and fault tolerance in the distributed setup. This attempt minimizes to a significant extent the consequences of server failures.

  - Failures can happen for individual applications, at the activity level. This can be handled when defining the activity template. An alternate or default

mechanism can be specified if the activity fails to complete successfully. For example, if the M-Urgency (chapter 8) IP-based streaming service fails, M-Urgency initiates a direct 911 audio call through the phone network. It is the responsibility of the application developer to envisage possible failures and propose alternates or default mechanisms to handle them.

- *Uncertainty*: The issues of uncertainity and the like are handled at the ontology level in RoCoMO, as discussed in chapter 4. This enables Rover II to associate an uncertainty measure with contextual information in the system. The data and uncertainty measure are propagated together within the system. For example, when a caller makes an emergency call using the M-Urgency system (section 8), the caller sends both his location and whether the location was acquired from GPS or Wi-Fi. Adopting a simple probabilistic approach, Rover II can communicate the location of the caller with an uncertainty value attached to it that is reflected on the map interface with a circle of uncertainty around the location point.

Chapter 6

Relevant Context - Context of Context

In this chapter, we discuss the concept and implementation of *relevant context*, taking context of the context into account. An exhaustive storage of contextual information regarding every entity will be available in Rover II. The critical question to answer here is, "Is all that contextual information required at every situation?". The answer is "no". Not every piece of contextual information will be relevant in every situation. For example, the professional information of a person involved in a road accident is of no value; rather, the person's medical record and personal information will be relevant context in that situation.

Figures 6.1, 6.2, 6.3 and 6.4 depict some situations and points out the relevant context of a person in those situations. The relevant context of the person is highlighted in each of the situations. As in figure 6.5, when considering the weather information, the static context of the person is of no importance. Just the dynamic context like the person's location needs to be considered in this case.

6.1 Relevance Filtering and Context Templates

Relevant context is different according to the situation in which the application is executed. Each application must be able to specify its own relevant context. We introduce *context templates*, with the help of which the application developers can define the relevant context for their application. The context templates for all the applications are stored

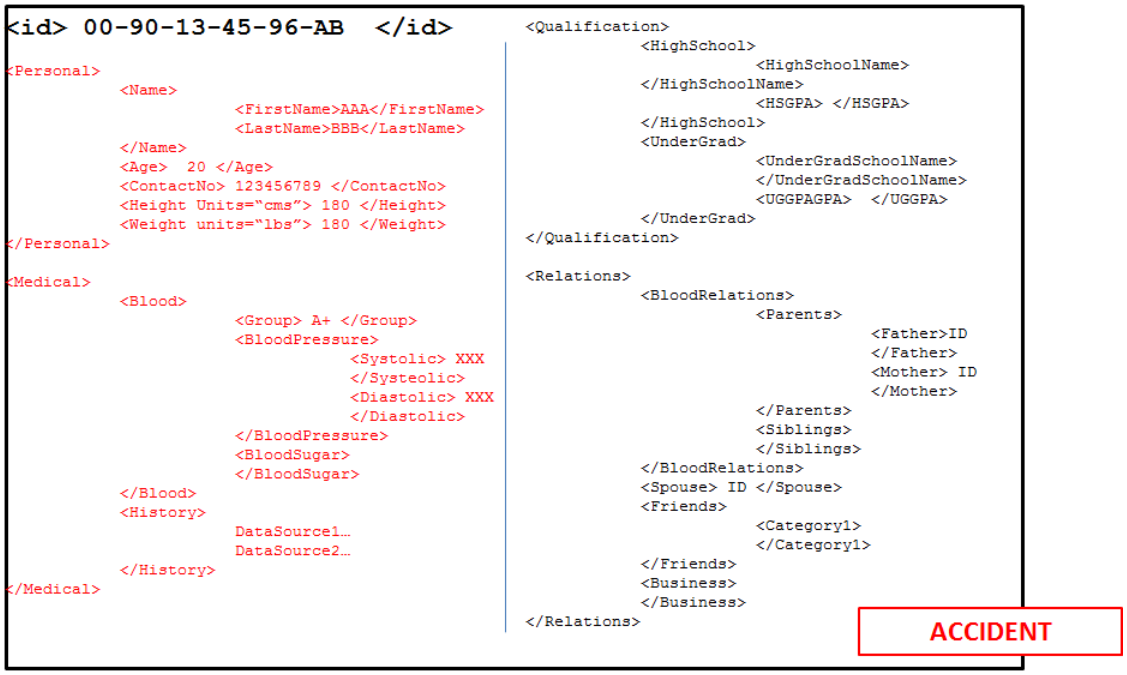

Figure 6.1: Relevant context in the situation of an accident

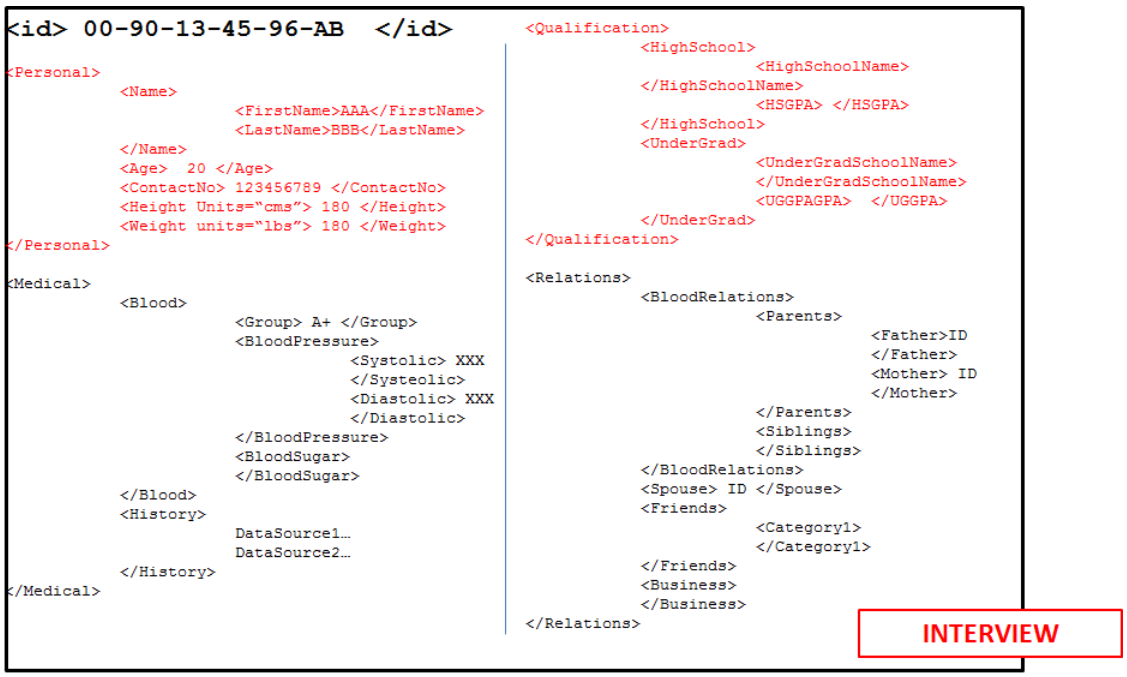

Figure 6.2: Relevant context in the situation of an interview

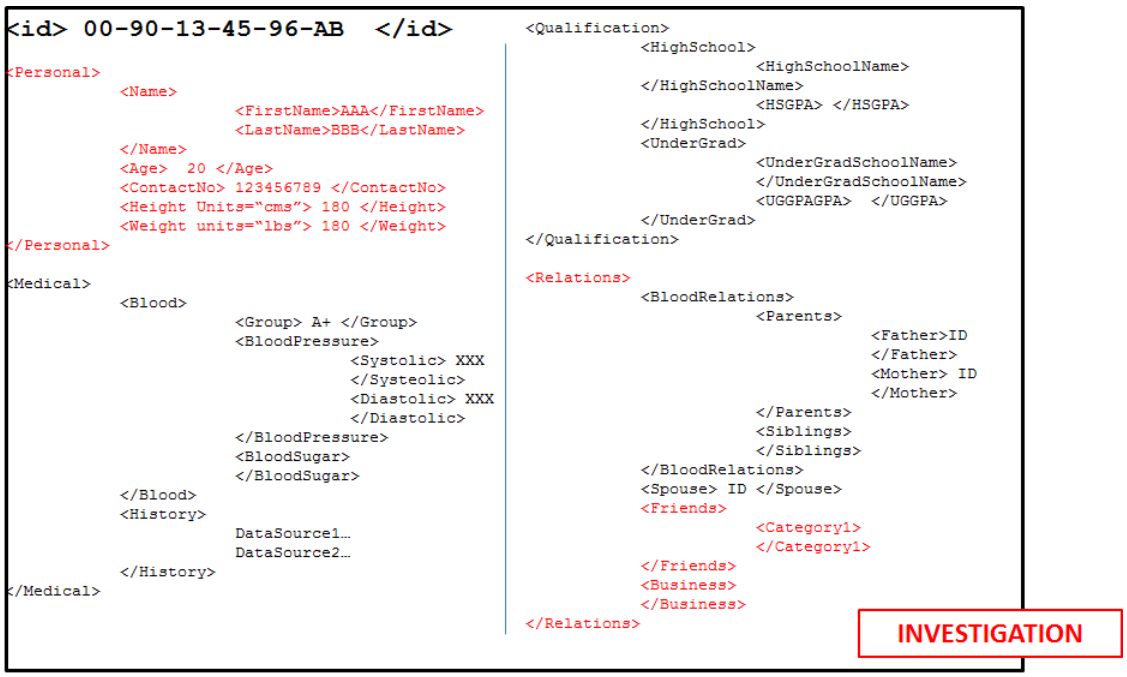

Figure 6.3: Relevant context in the situation of an investigation

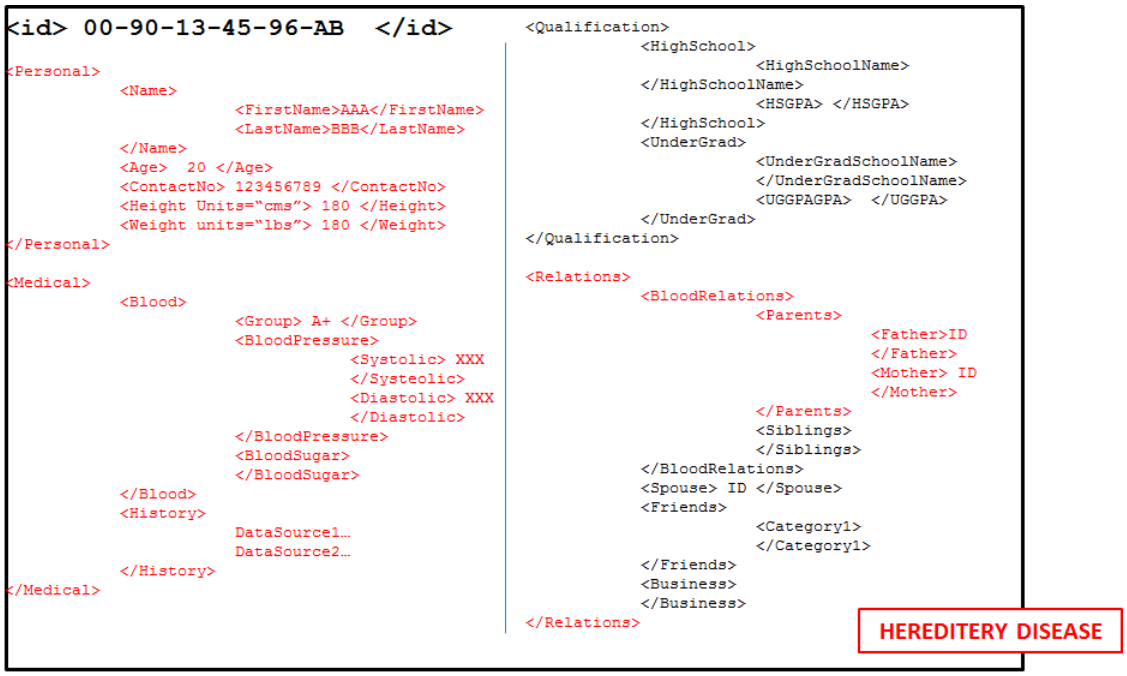

Figure 6.4: Relevant context in the situation of an hereditary disease diagnosis

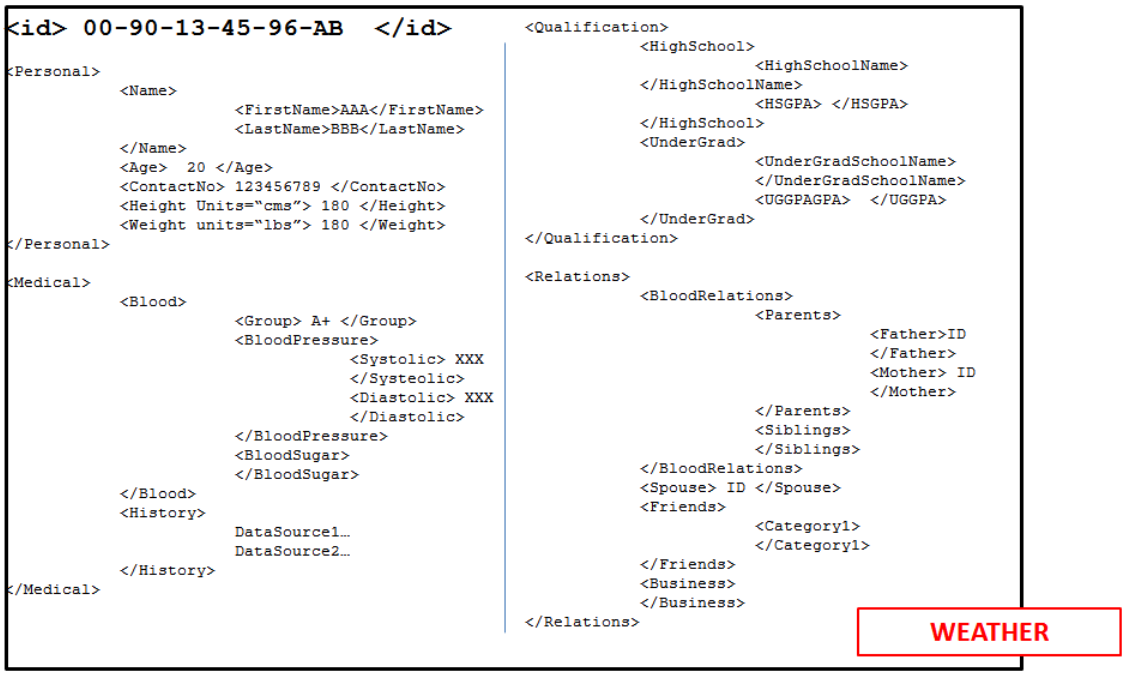

Figure 6.5: Relevant context when just considering the weather

in the *Template Store* in the context-tier of Rover II, as shown in Figure 5.2. We term the process of filtering the available contextual information into relevant context as *relevance filtering*.

A context template consists of:

- *Application ID*: The ID of the application, the context template is associated with. This template is instantiated once the application establishes a connection with Rover II.

- *Relevance Filter*: The relevance filter is the critical part of the context template where we specify the relevant context for the situation for the application. We can use one or more relevance filters.

- *Aggregation*: Once each of the relevance filters has generated a set of contextual

information, we specify the aggregation mechanism that is to be performed between each of the sets, like a union, intersection, etc.

6.1.1 Relevance Filter

Each context template consists of one or more *relevance filter(s)*. A relevance filter typically consists of four parts as listed below:

- *Source*: The contextual information of an entity to be considered for the situation.
- *Filter*: The contextual information used to filter the source.
- *Criteria*: The criteria for filtering. We have defined rules for criteria like "MATCH", "NEARBY" etc. "MATCH" which specifies that the information from the filter should match with the source; "NEARBY" typically looks for the difference between the two values being under a specified threshold. We will eventually incorporate other filtering criteria.
- *List*: The contextual information to be listed once the filtering is performed.

The role of each of the above components in the relevance filter is explained with an example later in the section. Relevance filters can be of two types:

- *Implicit Relevance Filter*: Some information is always required for an application. For example, when a caller makes an emergency call using the M-Urgency application, his/her personal mobile phone number must be accessed.
- *Explicit Relevance Filter*: Other information must be dynamically accessed based on the situation. For general M-Urgency calls, medical information is not required,

```
<appID> M-Urgency </appID>


<relFil id="relFil1">
      <src>user/personal</src>
      <filter>NONE</filter>
      <criteria>NONE </criteria>
      <list>user/personal</list>
</relFil>


<relFil id="relFil2">
      <src>medCenter/general/address/city<src>
      <filter type="explicit">LOCATION</filter>
      <criteria>MATCH</criteria>
      <list>medCenter/general</list>
</relFil>


<relFil id="relFil3">
      <src>medCenter/bank/blood</src>
      <filter>user/medical/blood/group</filter>
      <criteria>MATCH</criteria>
      <list>medCenter/general</list>
</relFil>
<operation> relFil1 UN relFil2 IN relFil3 </operation>
```

Figure 6.6: A sample context template

but for a road accident, the medical information about the person may need to be accessed.

6.1.2 Relevance Predicates

The *source*, *filter* and *list* are specified in the form of relevance predicates (RP). With the help of each RP, we can specify which particular property of the contextual information is sought by the application. In the example below, we specify two relevance predicates. The first seeks all the entity's personal information, and the second specifically seeks the age of the entity.

⟨ user/personal ⟩

⟨ user/personal/age ⟩

A sample context template is shown below. It contains the application ID that specifies the application associated with the context template and also contains three relevance filters (relFil). The first relFil accesses the user's personal information with no filtering because no filter is specified. The information to be displayed/listed is the user's personal information. The second filter finds medical centers close to the explicitly specified location of the application device. The criterion is to find exact matches and once a match is found, all the general information about the medical center is to be provided. Similarly the third filter looks for medical centers that have the user's blood type available in their blood banks. Finally, the aggregation mechanism is specified. The set of medical centers obtained in the second and third filters are intersected to find medical centers that have emergency casualty services along with availability of user's blood type in the blood

bank. The user information is combined with the resultant medical center set.

```
⟨rel:ContextTemplate rdf:application="#M-Urgency" rdf:ID="ContextTemplate1"⟩

  ⟨rel:RelFil rdf:ID="RelFil1" rdf:type="implicit"⟩
    ⟨rel:Src rdf:resource="#Person1" rdf:hierarchy="personal"/⟩
    ⟨rel:Filter rdf:resource="#"/⟩
    ⟨rel:Criteria rdf:Resource="MATCH"/⟩
    ⟨rel:List rdf:resource="#Person1" rdf:hierarchy="personal"/⟩
  ⟨/rel:Relfil⟩

  ⟨rel:RelFil rdf:ID="RelFil2" rdf:type="explicit"⟩
    ⟨rel:Src rdf:resource="#MedicalCenter1" rdf:hierarchy="personal/address/city"/⟩
    ⟨rel:Filter rdf:resource="@location"/⟩
    ⟨rel:Criteria rdf:Resource="NEARBY"/⟩
    ⟨rel:List rdf:resource="#Person1" rdf:hierarchy="general"/⟩
  ⟨/rel:Relfil⟩

  ⟨rel:RelFil rdf:ID="RelFil3" rdf:type="explicit"⟩
    ⟨rel:Src rdf:resource="#MedicalCenter1" rdf:hierarchy="bank/blood"/⟩
    ⟨rel:Filter rdf:resource="#Person1" rdf:hierarchy="medical/blood/group"/⟩
    ⟨rel:Criteria rdf:Resource="MATCH"/⟩
    ⟨rel:List rdf:resource="#MedicalCenter1" rdf:hierarchy="general"/⟩
  ⟨/rel:Relfil⟩

  ⟨rel:Aggregation⟩
    ⟨rel:Aggregation rdf:operation="UN" rdf:resource="#relFil1"/⟩
    ⟨rel:Aggregation rdf:operation="UN" rdf:resource="#relFil2"/⟩
    ⟨rel:Aggregation rdf:operation="IN" rdf:resource="#relFil3"/⟩
  ⟨/rel:Aggregation⟩

⟨/rel:ContextTemplate⟩
```

We do not intend to replace query languages with the *relevance filters*. Since the information is finally stored in a database, the relevance filters only complement SQL. Relevance filters are designed to be compatible with the ontological approach and convenient for the application developers to construct and specify a rule to access contextual information. The context engine internally converts the relevance filter into an appropriate query.

6.2 Context Engine

The context engine plays the role of identifying the application that has established a connection with Rover II and selecting and identifying the appropriate context template. It instantiates and parses the context template. It plays the role of the middleware between the application, the appropriate context template and the actual context store to retrieve the relevant context as shown in Figure 6.7.

6.3 Situation Graph

Situation graph, which we believe is an effective way of presenting the context of a given situation to the user. We shall explain it with respect to a situation related to M-Urgency application (Chapter 8).

The situation graph is constituted of all the four primitives, as in RoCoM (Chapter 3). It comprises the *event/activity graph* and the *relevant context graph*. The situation graph provides not just the relevant contextual information of a given situation but also the information about the activities that have taken place prior to the situation and those that

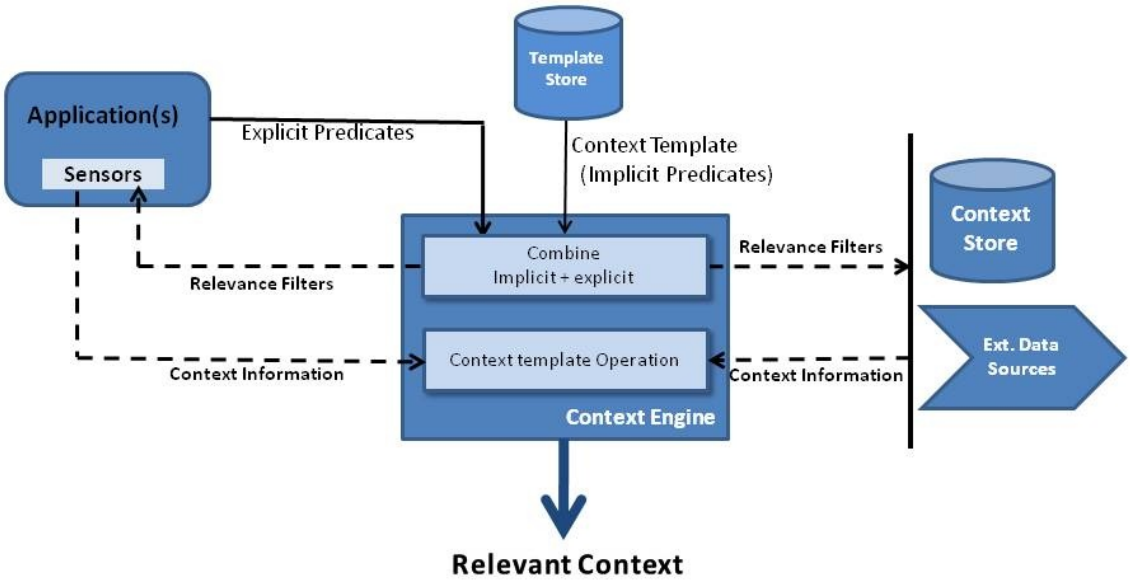

Figure 6.7: Depiction of the process of generating the relevant context by the context engine

would take place immediately after the situation. This helps the users best understand the situation and make decisions accordingly. In applications/systems like M-Urgency, decisions about potentially life-threatening and urgent situations arise. Thus, providing the best picture of a given situation becomes very important.

6.3.1 Relevant Context Graph

Rover II maintains exhaustive contextual information associated with entities. As discussed above, for a given situation only a subset of the exhaustive contextual information is relevant. The relevant context graph represents the relevant context of a given situation, comprising the entities, their relevant contextual information and the relationships between the entities.

Relevant context graph is generated as the result of relevance filtering. The graph is

initially formed as a result of the implicit relevance filters. The graph then expands or contracts as the explicit relevance filters are added to the context template as a result of the various activities.

A relevant context graph can be represented as

$$\text{relevant context graph } \mathbf{G = (V, E)}$$

$$\text{where,}$$

$$\mathbf{V} \rightarrow \mathrm{e} \bigcup \mathrm{p}$$

$$\mathbf{E} \rightarrow \mathrm{r}$$

where the vertices could be either an entity **e** or one of its properties **p** **r** is a relationship between individuals or properties of the individuals; such that r could be either of the following:

$$\mathrm{r}(\mathrm{e}_n, \mathrm{e}_m)$$

$$\mathrm{r}(\mathrm{e}_n, \mathrm{p}_m)$$

$$\mathrm{r}(\mathrm{p}_n, \mathrm{p}_m)$$

Figure 6.8 through 6.11 provide an examples of how a relevant context graph forms and contracts/expands as the context template is updated. These examples are explained assuming an emergency call in the M-Urgency system about a car accident where the caller is an accident victim.

We shall first look into the graph produced by each individual relevance filter. As an M-Urgency call is made, irrespective of why it is made, the caller's personal information is made available to the dispatcher. Thus, the context template has an implicit relevance

filter as shown in Figure 6.8. It is later recognized that the call is made because of a road accident and thus an explicit relevance filter is added as shown in Figure 6.9, seeking information about medical center near the place of accident. Later, another relevance filter is incorporated that specifies the blood group of the victim that should be available in the medical center. Medical centers A, B and D have the blood group in stock and thus included in the graph as shown in Figure 6.10. Eventually another explicit relevance filter is added to the context template seeking the people under the relationship hierarchy of the victim's contextual information who have the same blood group as his. The resultant graph is shown in Figure 6.11.

The final relevant Context Graph is shown in Figure 6.12. The resultant graphs of the relevance filters are combined together through the aggregation process - relFil1 $\cup$ (relFil2 $\cap$ relFil3) $\cup$ relFil4.

6.3.2 Event/Activity Graph

As pointed out in our context model, we believe that a situation is triggered by an *event*. Applications would establish a connection with Rover II as a consequence of an event and the situation thus defines the context. In the example with the M-Urgency application, the accident event triggered the caller application to establish a connection with Rover II.

The event is the root of the event/activity graph, and the triggered activities, whether sequentially or concurrently executed, form the other nodes of the graph. Typically each node of the graph is a *goal*, as discussed in chapter 4 (RoCoMO). The goal of the event

```
<relFil id=relFil1>
   <source> user/personal</source>
   <filter> NONE </filter>
   <criteria> NONE </criteria>
   <list> user/personal </list>
</relFil>
```

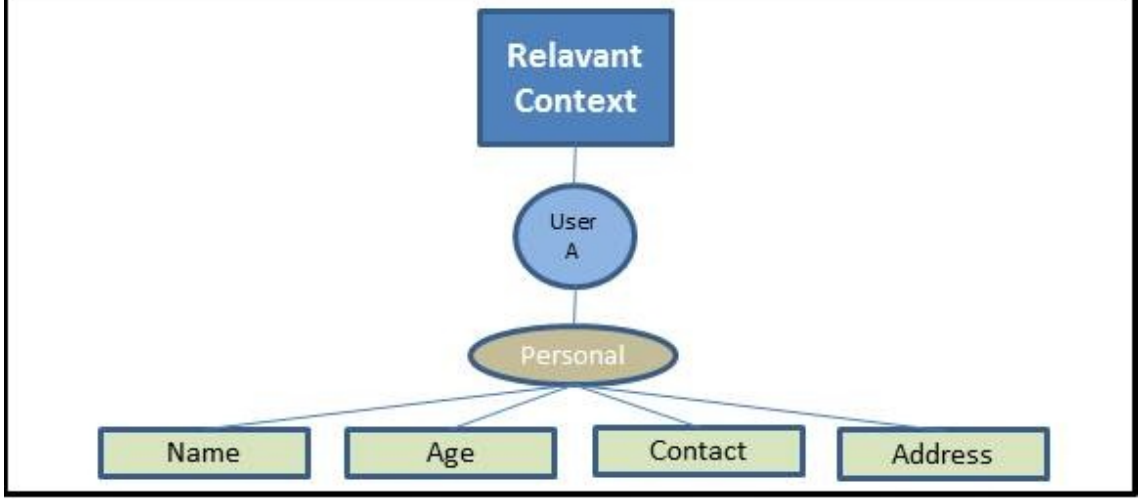

Figure 6.8: Graph generated by relevance filter seeking personal information of the caller

```
<relFil id=relFil2>
   <source> medCenter/general/address/city </source>
   <filter type="EXPLICIT"> LOCATION </filter>
   <criteria> MATCH</criteria>
   <list> medCenter/general</list>
</relFil>
```

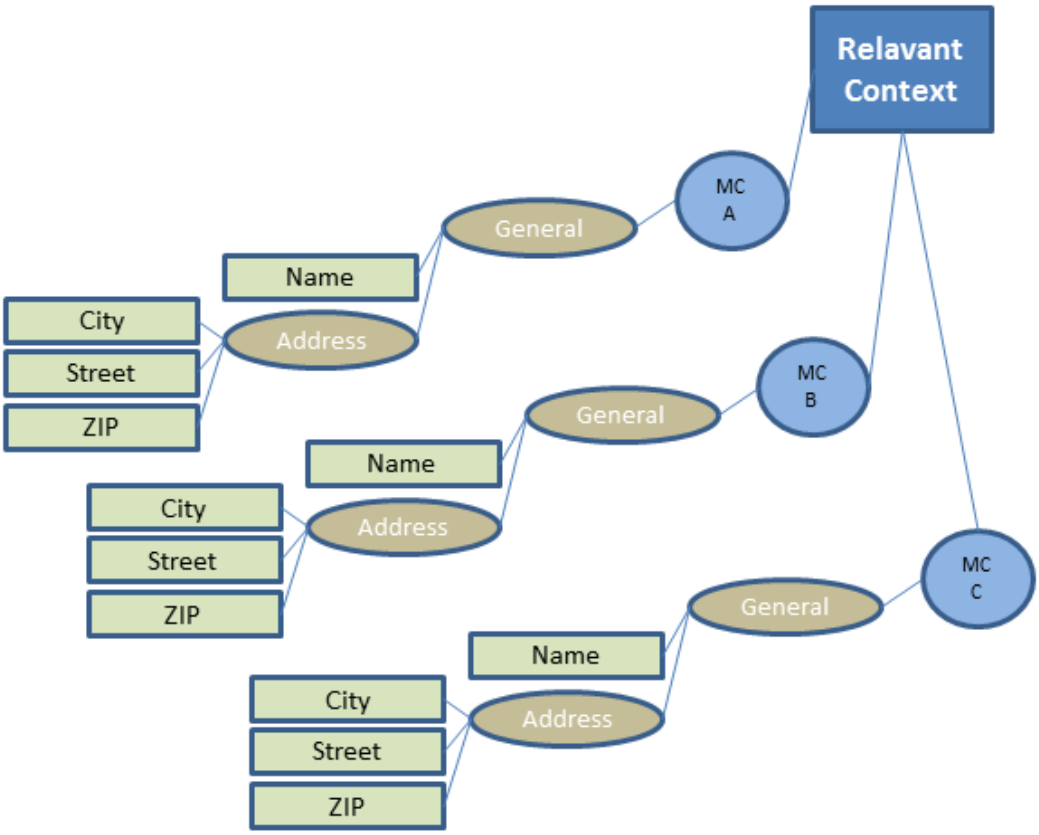

Figure 6.9: Graph generated by relevance filter seeking Medical centers near the accident spot

```
<relFil id=relFil3>
   <source> medCenter/bank/blood </source>
   <filter> user/medical/blood/group </filter>
   <criteria> MATCH</criteria>
   <list> medCenter/general</list>
</relFil>
```

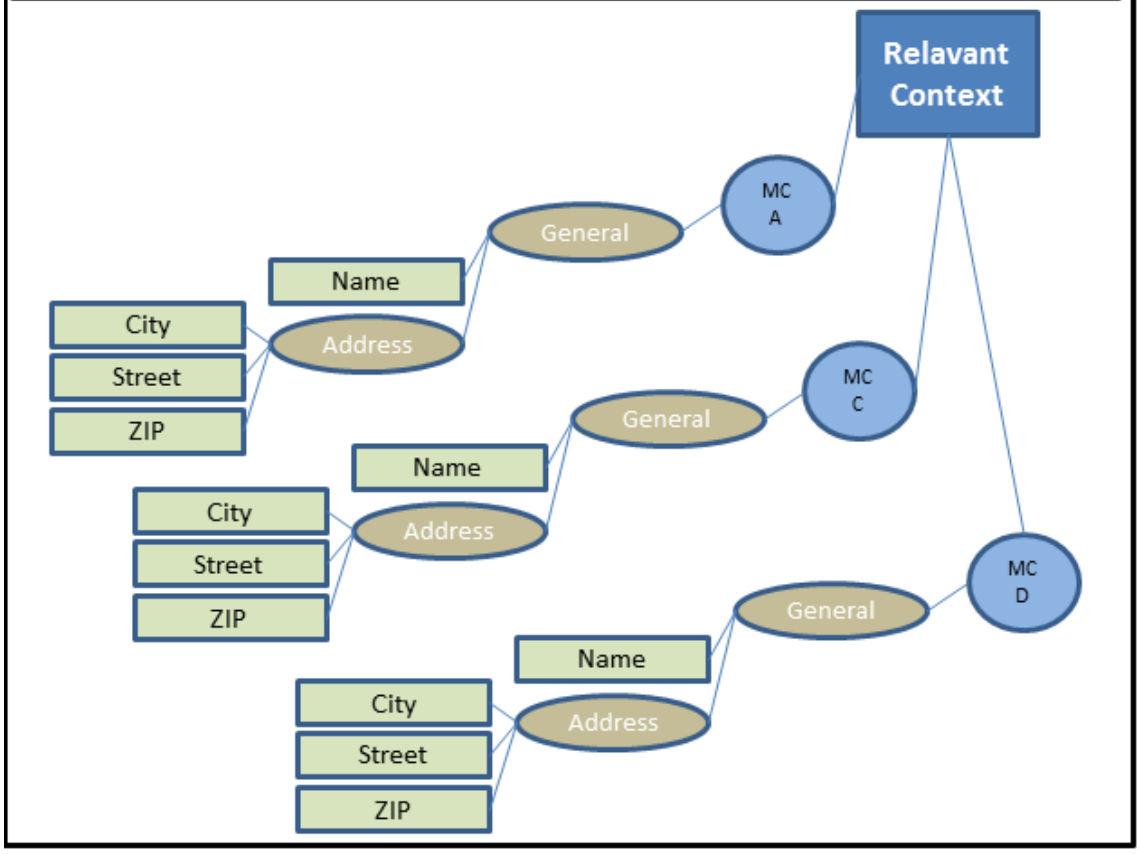

Figure 6.10: Graph generated by relevance filter Medical centers with the appropriate blood group

```
<relFil id=relFil4>
   <source> user/relation/family/…/blood/group </source>
   <filter> user/medical/blood/group </filter>
   <criteria> MATCH</criteria>
   <list> user/relation/family/…/personal  </list>
</relFil>
```

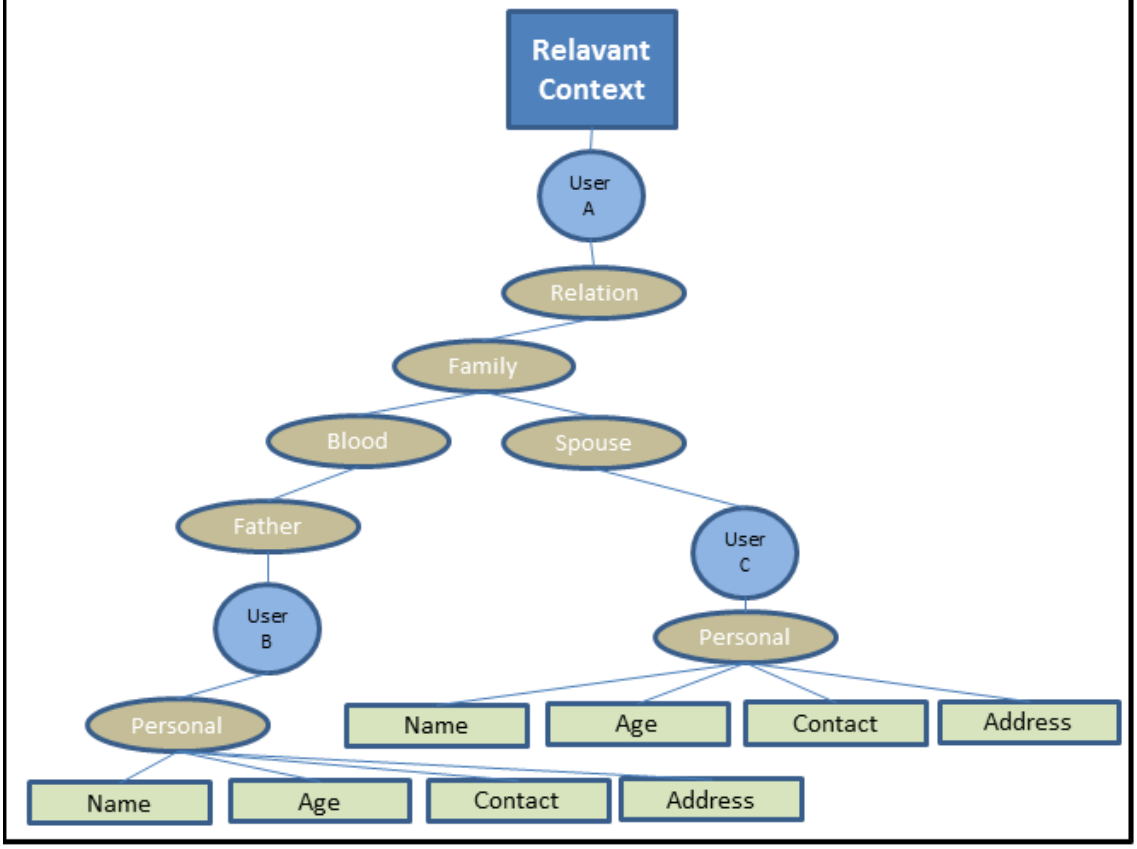

Figure 6.11: Graph generated by relevance filter seeking seeking relations with same blood group

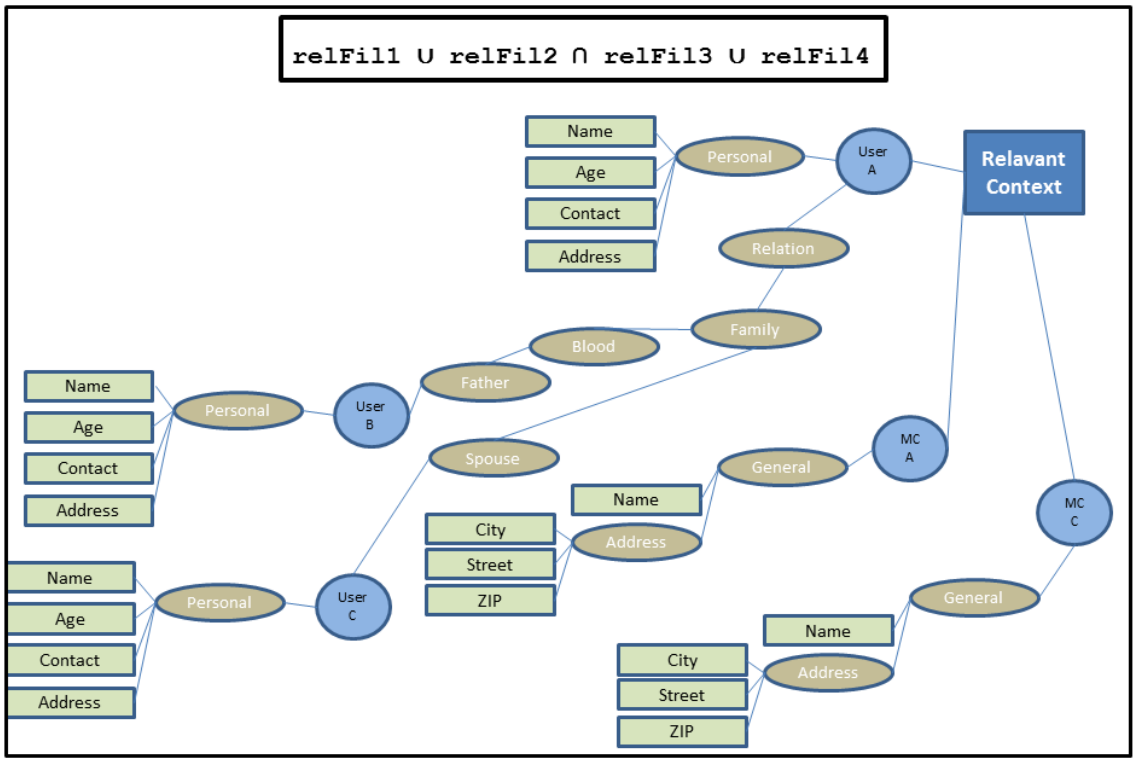

Figure 6.12: Relevant Context Graph

drives the whole situation, resulting in activities and each activity is driven by individual goals. The goal act as the post-condition for a given activity and the pre-condition for the following activity. Thus, the event/activity graph can be represented as:

event/activity graph **G=(V, E)**

where,

$$\mathbf{V} \rightarrow \mathrm{e} \bigcup \mathrm{g}_n$$

$$\mathbf{E} \rightarrow \mathrm{a}_n$$

where **e** is the event that triggers the whole situation,

$\mathbf{g}_n$ is the goal of n^{th} activity,

and $\mathbf{a}_n$ is n^{th} activity, as the result and following the event.

The event/activity graph typically begins from a single node, being the event **e** and would converge back into a single node $\mathbf{g}_e$, which is the overall goal of the event. Once the overall goal of the event is achieved, the situation ceases to exist.

Figure 6.13 shows a possible event/activity graph of an accident event and the subsequent M-Urgency call. As the accident happens, the M-Urgency call is made to the emergency dispatcher through Rover II. The dispatcher gathers information about the event and assigns a police officer to the incident and a paramedic team to the event. The officer and the paramedic would carry out their respective duties. Each activity may spawn more activities. For example, after an officer assesses the situation, he may call for a vehicle towing service if vehicles are badly damaged. The graph thus expands as different activities are carried out and finally contracts as the overall *goal* of the event is achieved. The event/activity graph may be much more complex in real world scenario.

6.3.3 Combining the two graphs

As mentioned earlier, Event/Activity graph combined with the Relevant Context Graph forms the *situation graph*. A third dimension is added to the graph wherein, each node of the event/activity graph acts as a root and spans a relevant context graph.

The *situation graph*, combining the event/activity graph and relevant context graph, can be represented as:

situation graph **G=(V, E)**

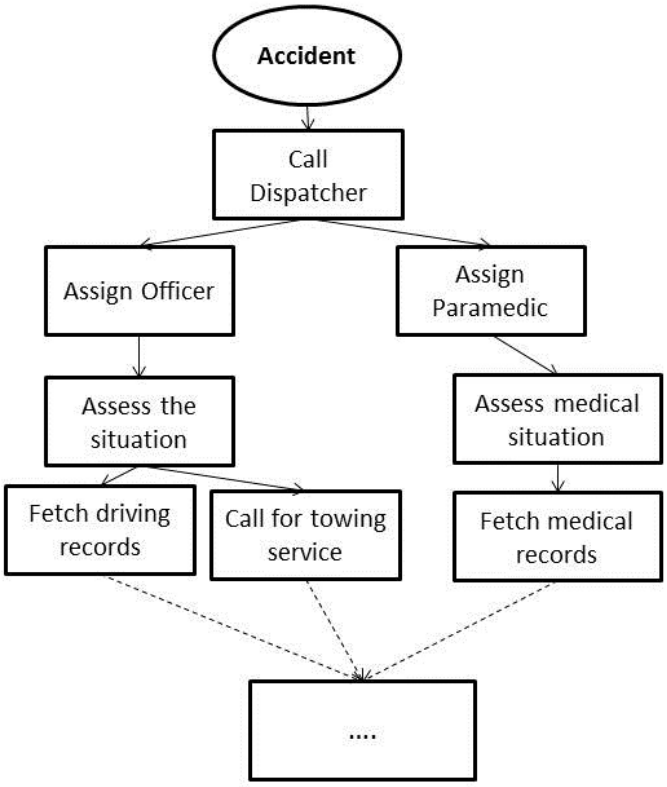

Figure 6.13: A typical Event/Activity Graph

where,

$$V \rightarrow e \bigcup g_n \bigcup RCG$$

$$E \rightarrow a_n \bigcup r$$

where,

e is the event that triggers the situation,

$\mathbf{g}_n$ is the goal that drives the n^{th} activity, triggered by and following the event,

RCG is a relevant context graph,

$\mathbf{a}_n$ is the n^{th} activity following and triggered by the event,

and **r** is a relationship.

The *event* and the *activities* each define a situation and influence its context. In other words, each of them may contribute a new relevance filter to the context template. Thus, each of the nodes in the Event/Activity Graph span a Relevant Context Graph. The node "Relevant Context" in the relevant context graph will be replaced by the nodes from the event/activities graph. As we progress through the Event/Activity Graph, the Relevant Context Graph expands and contracts as discussed in section 6.3.1. Figure 6.14 shows an instance of the situation graph when the information regarding the available donors is fetched, as seen in the example explaining relevant context graph. Thus at each instance of the situation graph, a complete idea about the situation can be realized, thus best understanding the situation or being effectively context-aware.

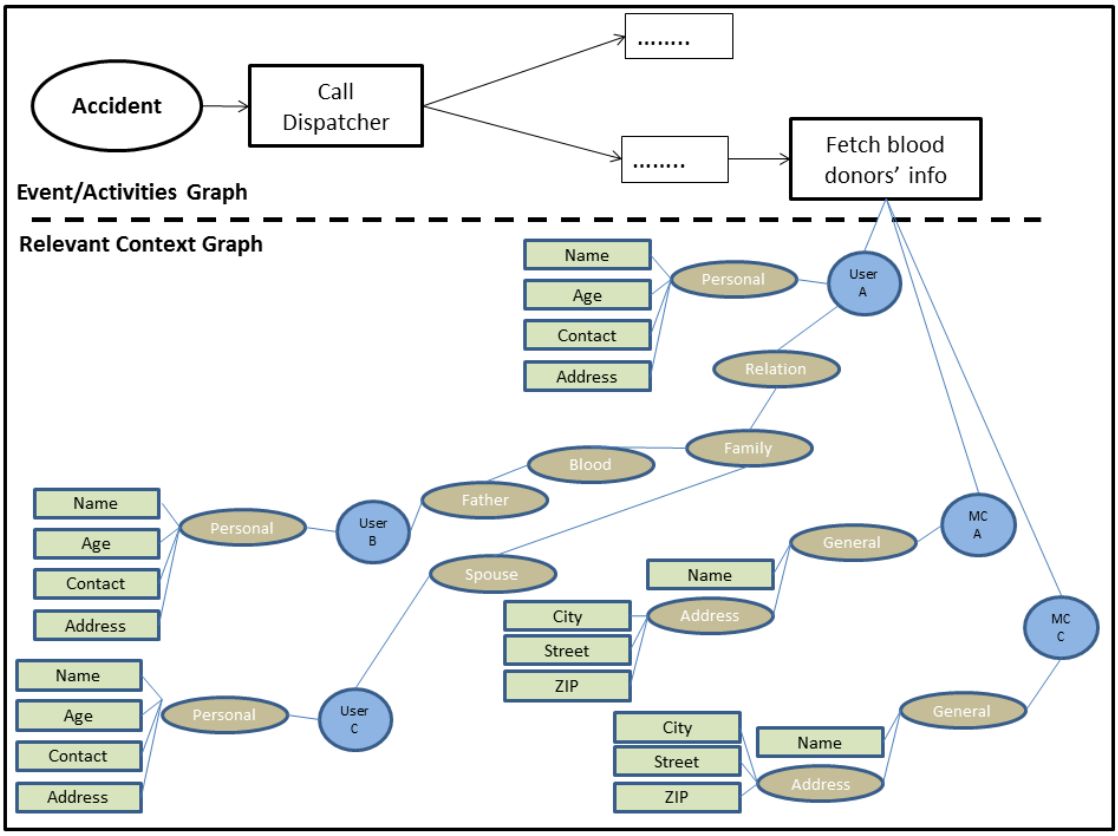

Figure 6.14: An instance of the situation graph

Chapter 7

Information Model and Information Brokering

As discussed in our proposed context model in Chapter 3, *events* act as a trigger to the functioning of a system involving some *entities* sharing *relationships* within themselves and *activities* as a consequence of the event. An context-aware application should be designed to perform these set of activities taking into account the context of each of the entities involved and activities. In this chapter we briefly discuss about *event-driven systems* and introduce our *Information Model*.

7.1 Event-Driven Systems

An event can be defined as ”a significant change in context of an entity. For example, when a consumer purchases a car, the car’s context changes from ”for sale” to ”sold”. A car dealer’s system architecture may treat this state change as an event to be produced, published, detected and consumed by various applications within the architecture.

This approach may be applied by the design and implementation of applications and systems which transmit events among loosely coupled software components and services. An event-driven system typically consists of event emitters (or agents) and event consumers (or sinks). Sinks have the responsibility of applying a reaction as soon as an event is presented. The reaction might or might not be completely provided by the sink itself. For instance, the sink might just have the responsibility to filter, transform and forward the

event to another component or it might provide a self contained reaction to such event. The first category of sinks can be based upon traditional components such as message oriented middleware while the second category of sinks (self contained online reaction) might require a more appropriate transactional executive framework.

Building applications and systems around an event-driven architecture allows these applications and systems to be constructed in a manner that facilitates more responsiveness, because event-driven systems are, by design, more normalized to unpredictable and asynchronous environments.

7.2 Our Information Model

Getting into the details of event-driven systems are beyond the scope of our work. Our focus is on applications that can benefit the most from Rover-II framework which are basically driven by one or more events. We find it best to explain our Information Model with an example. Let us assume that an application is used as a consequence of an emergency event that takes place, like a natural calamity or a fire incident. Before an event occurs, all the resources that may be needed to address the situation are located at their normal locations for storage. The situation on the ground varies widely in the area and so do the needs for help, whether it is for putting out fire, rescue from collapsed building, flood, trapped in vehicle etc., emergency medical evacuation support, transportation food shelter, etc. All such needs are only known in the local area and the information about the need, in context of the local area and its timely-ness have to be available for coordination.

In most calamities the resources needed far exceed the requirements and most of the

resources are not at the places where they are needed. In order to effectively organize the support operation the first requirement is that of the ground truth reflecting not only the needs at various locations but also the context including the urgency of support. The knowledge about the resources of various kinds is also needed. Then only any suitable plan can be prepared and executed in a coordinated manner. When such steps are not taken chaos often results. We note that getting the right information from various locations to the location where coordination can be carried out requires the movement of a lot of information. As the infrastructure for communication is often affected by such calamities, wireless communications which can continue to function play a very significant role.

Further, most of the help is provided to people affected by the calamity and the individuals not only need help but also information about the situation around them as well as about their loved ones. Thus, we believe that critical information provided by/to the people in case of emergencies or crises should not just be timely, but should be relevant to the person. This can be done only if the context of the person is understood along with the context of the situation/environment around him. Timely and critical decisions are possible based on how effective the piece of provided information is.

A lot has been discussed in the literature on how context of the situation and place is relevant when handling event. We discuss about some of the work from the literature specifically discussing emergency situations. The paper [116] reflects on how local area details provide the context to which plans are tailored. Rainsford et. al. [31] highlights how information overloading is avoided if information is provided to the user in specific context. Bahrani et. al [13] specifically stress upon context-awareness when collaborative planning is done between distributed planners. Kikiras et. al. [70] has proposed an

integration platform for autonomic computing for disaster relief. The idea here is to provide a middleware for intelligent, collaborative and context aware services using rumor spreading techniques. Siren [142] provides a foundation for gathering, integrating, and distributing contextual data, such as location and temperature to support tacit communication between firefighters with multiple levels of redundancy in both communication and user alerts. Tomaszewski et. al. [131] says information foraging and sensemaking with heterogeneous information are context-dependent activities. They attempt the challenges of constructing and representing context within visual interfaces that support analytical reasoning in crisis management and humanitarian relief. The paper [85] specifically discuss about geo-spatial information. They briefly put forth on how groups or groups of groups can work with geospatial information and technologies in crisis management.

7.2.1 Information Integration Point

When an event happens, we require an ”Information Integration Point” (IIP) where *information brokering* is performed. Information is acquired from varied sources can be compiled meaningfully together and effectively provided to the different categories of people - be it the general public which is getting effected or the personnel who are addressing the situation of calamity from the administration side.

We propose an Information Model as shown in figure 7.1, which depicts a continuous series of operations, or process, whose performance is enhanced at each node, and collectively through mutual access to information and collaborative utilization of information.

The ”I” in the information model represents the information pool that is generated

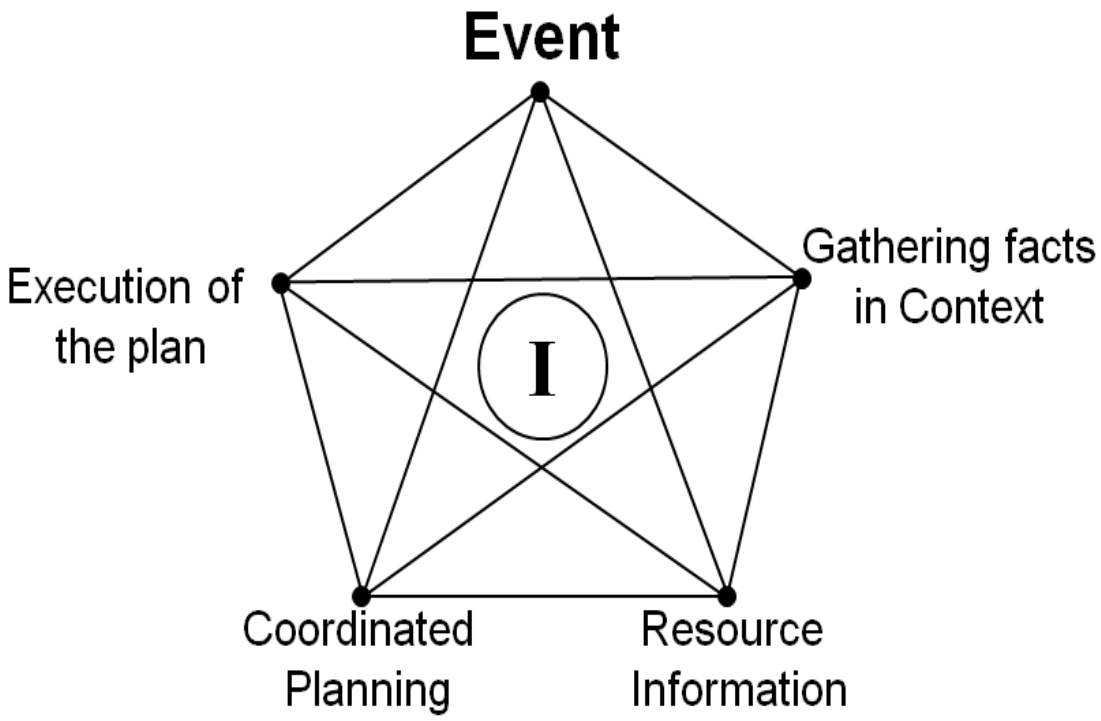

Figure 7.1: Information Model for effective information management during an event

as the consequence of an event. By event we mean a natural calamity or a situation of humanitarian crisis. A series of activities or processes follow the event, which are represented by the rest of the nodes. The nodes revolve around information, either utilizing it or contributing to it. The nodes are inter-related with each other through the information they share, one process complementing the other.

As the event takes place, all the information regarding the event can be collected and pooled at the IIP. It is critical that that information is gathered in context, based on the situation and relevance. For example, when an earthquake tremor is felt in a particular city, information such as the weather, traffic in major roads, information related to the population of the city may all be relevant. But information such as the literacy rate or birth rate with respect to the city does now have much relevance in addressing the situation at hand.. Thus, the facts are to be gathered in context at the IIP. The gathered information could be of two types: ” Static Contextual Information: Information that would remain unchanged over the period the relief work is on. For example, the information related to the offices and how many people work at each, the location and normal capabilities of the hospitals, etc.. ” Dynamic Contextual Information: Information that change rapidly as the various activities are initiated. For example, the road traffic situation, weather etc.

The next step would be to gather the information about all the various resources available that need to be utilized as part of the relief work. A coordinated planning follows based on the information available about the event, the contextual related information regarding the event and the information about the resources. There may be many bodies/organizations involved in the relief work. Thus, a coordinated effort has to be facilitated by the IIP. IIP would be the core of the activities where the key decisions are

made by personnel based on the different forms of information available. Then, the IIP is responsible in initiating the execution of the plan.

This model is incomplete with just a single iteration of the above mentioned steps. As discussed before, an event will lead to more consequential events for which the whole cycle has to be repeated again. As discussed in the beginning of this section, though the earthquake was the major event, it incited more events like situations of major traffic jam, water pipeline damage which require a full iteration in the information model each.

We also note that in a relief operation there may be multiple IIPs such that each may have some dedicated responsibilities and there may be one IIP which coordinates the efforts of other IIPs.

Chapter 8

M-Urgency - an illustrative application

In this chapter, we present the application M-Urgency [76][74][77][75][78][84], built for the Rover platform. We explain the application with respect to our context model and show how the application will make use of our context architecture, in Rover-II. M-Urgency is a public safety system that significantly advances how emergency calls are handled. M-Urgency enables a person to connect with an emergency dispatcher and establish an audio and video stream. We believe that this example application is a strong proof of how effective and efficient emergency handling can become if the right contextual information is available at the point of decision making and if convenient methods are provided to them to initiate the activities.

M-Urgency presents information that facilitates time-critical decisions by the dispatcher and responder. The dispatcher and the responder are provided with contextual information such as:

- Relevant information about the user, e.g. disability or special needs, gender etc. so that they can dispatch aid accordingly.
- The users real time location that is reflected on a map interface even when he or she is mobile.
- Additional information, such as weather and traffic, to gather the context of the incident scene.

There have been attempts to provide location services during emergency calls [115][148]. We attempt to take it a step further by providing real-time location as the incident/user may not be static. As noted above, we consider many variables, in addition to the location, to define context pertaining to the user and the incident. The context can be better defined as more services are added to the service tier of Rover-II.

Since the emergency personnel are provided with all the contextual information, they are in a position to efficiently provide service accordingly. Further, M-Urgency enables the dispatcher to forward the stream and the contextual information to first responders like a police officer, ambulance, fire engine etc., assigned to handle the call, by a gesture as simple as drag and drop. In this way the responder precisely knows the latest situation even before he gets to the incident scene.

8.1 Components

There are various components of M-Urgency system that communicate with Rover-II. Figure 8.1 shows a high level architecture of the M-Urgency system. When an M-Urgency call is made from a mobile device, the audio/video stream is mediated by the Adobe Flash Media Player, forwarding it to the Dispatcher application. An option is available to split the audio from the stream and send it as a SIP call to the existing E-911 system at the dispatcher's end. The dispatcher can assign one or more emergency responders to the incident and can forward the audio/video streams, along with all other pieces of contextual information to the responders so that they have a clear idea about the scene even before getting there physically.

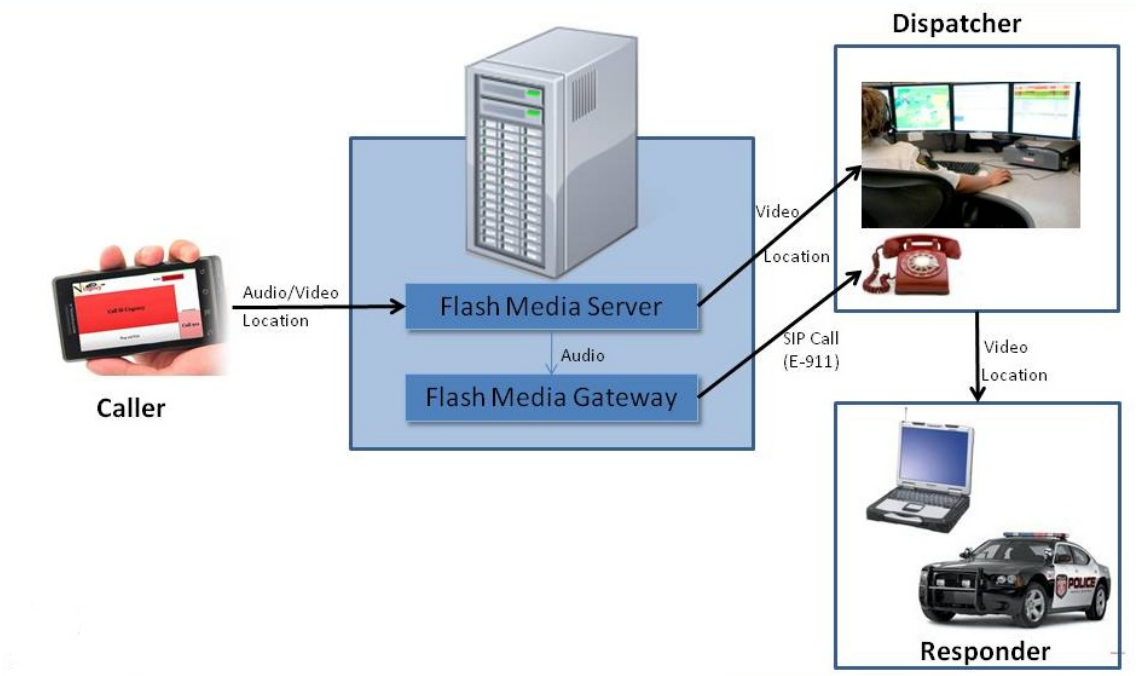

Figure 8.1: High level architecture of M-Urgency system

8.1.1 Caller Application

It runs on a smart phone, tablets, PDA, laptop, etc., allowing the user to initiate an emergency call by the press of a button. The application communicates with Rover-II and facilitates the location and the streaming service. Thus, an audio/video stream along with the location information is sent to the PSAP. Fig 8.2 shows the screen shot of the caller application.

8.1.2 Dispatcher Console Application

This is the critical part of the M-Urgency system. Used by the PSAP, it runs on a desktop and receives emergency calls with the audio/video feeds. Fig 8.3 shows the screenshot of the dispatcher console application. It enables the dispatcher to:

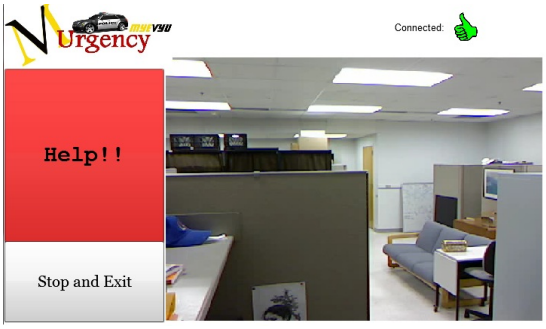

Figure 8.2: Caller Application screenshot

- View the incoming calls (on the left) and interact with them using audio selectively as needed,
- View the real time location of the caller on a map interface along with the context information about the caller,
- View the list of available emergency responders (on the right),
- Group more than one emergency calls together, if so desired,
- Assign responder(s) to the caller/incident through drag and drop operations, etc.

We present the actual sequence of events in the subsequent sections.

8.1.3 Emergency Responder Application

It runs on laptops, typically in the emergency vehicle. Once the dispatcher assigns the responder to an incident/caller, this application receives the forwarded audio/video streams of the assigned caller(s) along with the real time location information of the same. Thus the responder gets the precise information about the emergency situation and can get there prepared accordingly. Fig 8.4 shows the screen shot of a responder application.

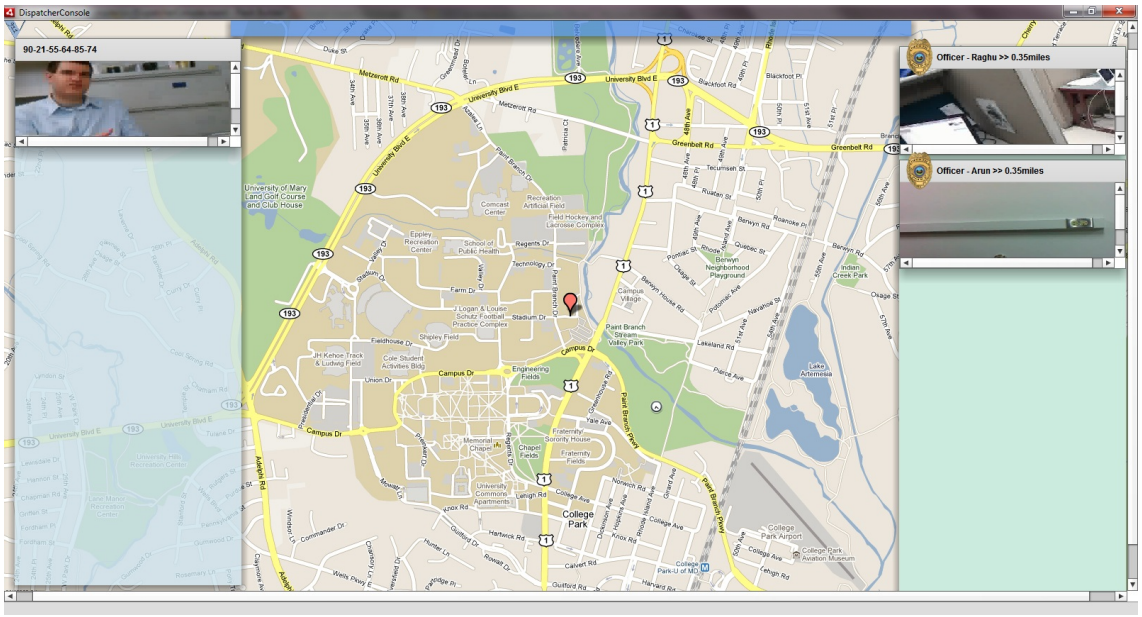

Figure 8.3: Dispatcher console application screenshot

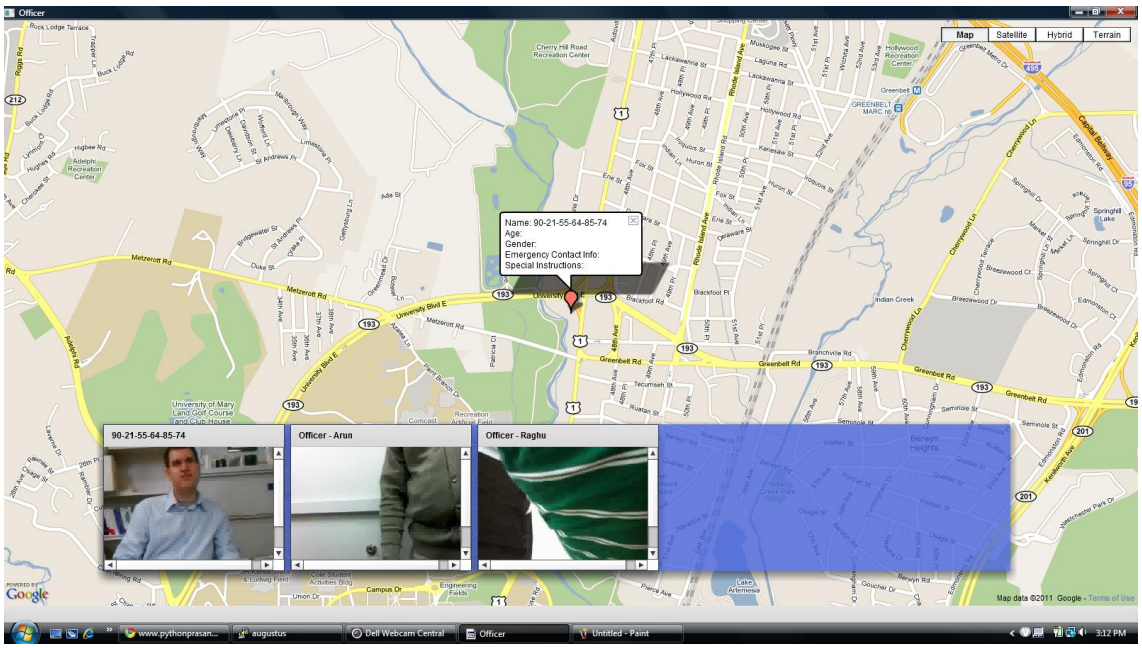

Figure 8.4: Emergency Responder application screenshot

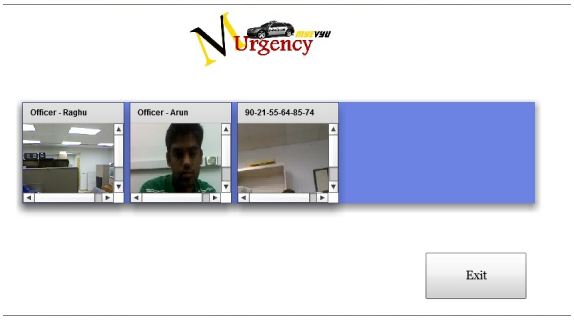

Figure 8.5: Emergency Responder application screenshot on a mobile device

In practical situations, the emergency responder cannot always be present in the emergency vehicle to receive the calls. The application can also run on a mobile phone which can receive the forwarded streams of audio and video. Fig 8.5 shows the screen shot of the mobile version of the emergency responder application.

8.2 Social Contribution in M-Urgency

We believe that the M-Urgency system could be further effective if more information could be gathered from the point of incident or from the vicinity. Installing additional sensors, security cameras or deploying additional security personnel would be an expensive option. An effective approach would be to utilize the people around, in the vicinity of the emergency incident. Social contribution from other users in the near vicinity of the M-Urgency caller can greatly help the emergency personnel understand the situation better and can even provide assistance to the caller, if required. We have extended the M-Urgency application enabling the emergency dispatcher to define a geo-fence around the caller and look for other M-Urgency users in the region. Other than just gathering information from them, the dispatcher could also group the user(s) with the caller establishing

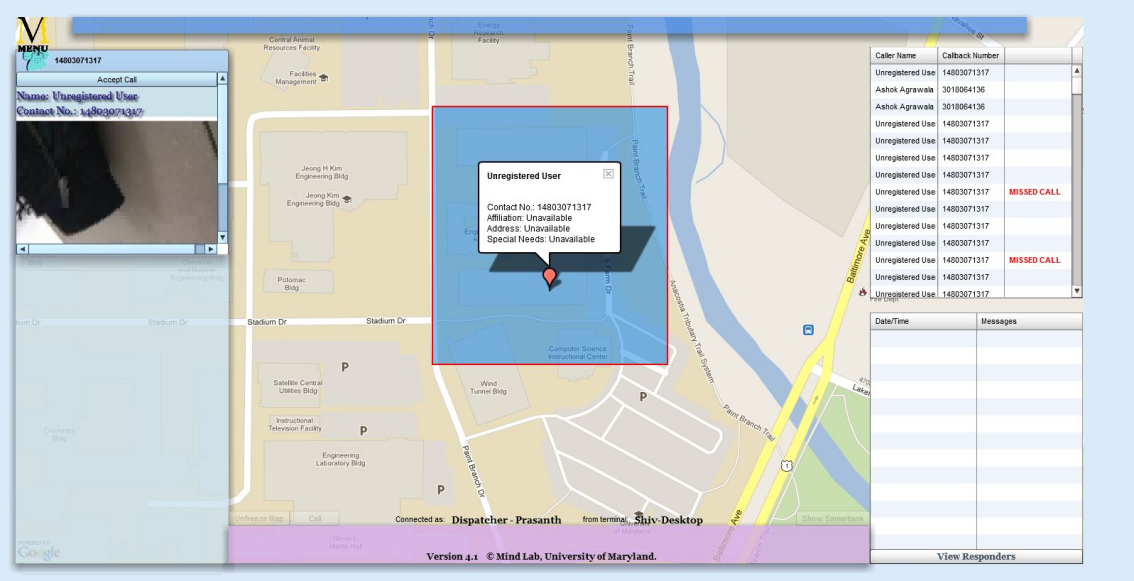

Figure 8.6: Dispatcher creates a geo-fence around the caller's location and initiates a call to the users within the region

a audio/video interaction between them.

As the emergency dispatcher, at the PSAP, receives an M-Urgency call, the caller's window appears on the left side of the screen and the real time location displayed on the map interface (figure 8.3). The map is zoomed to the resolution such that all the structures and landmarks of the region can be recognized by the dispatcher. The dispatcher has the provision to mark a rectangular region around the callers' location, defining a geo-fence to look for other users in the region. Figure 8.6 shows the callers' location from a particular building and the dispatcher geo-fences the whole building looking for other users in the building.

M-Urgency users can opt in to be a good samaritan, ready to help in case of an emergency. When such a user activates the caller application on mobile device, it establishes a connection with Rover-II and runs in the background. The location of the users are

Figure 8.7: The caller application interface, when a call from the dispatcher is received.

provided to the server once every few minutes. Thus, the Rover-II server keeps track of active users and their locations. When the dispatcher defines a region around the caller's location and initiates a call, all the M-Urgency users within the rectangular region are intimated. The user receives an audio alert and the caller application is automatically invoked to the foreground. The interface of the application changes, as shown in figure 8.7, wherein the user can accept or reject the call. Once the user accepts the call from the dispatcher, an audio/video stream is established with the dispatcher. The windows of such users appear at the bottom of the dispatcher console, as shown in figure 8.8.

The dispatcher can interact with each of the users and gather as much information about the situation from them. He could also hook the users up together by grouping them. The users can then interact with each other through a audio/video stream. The caller application interface is shown in figure 8.9. The caller could tap on each of the other user's window to interact with them and to enlarge their video pane. The dispatcher will be involved in every group and has the ability to add or remove any user to/from the group.

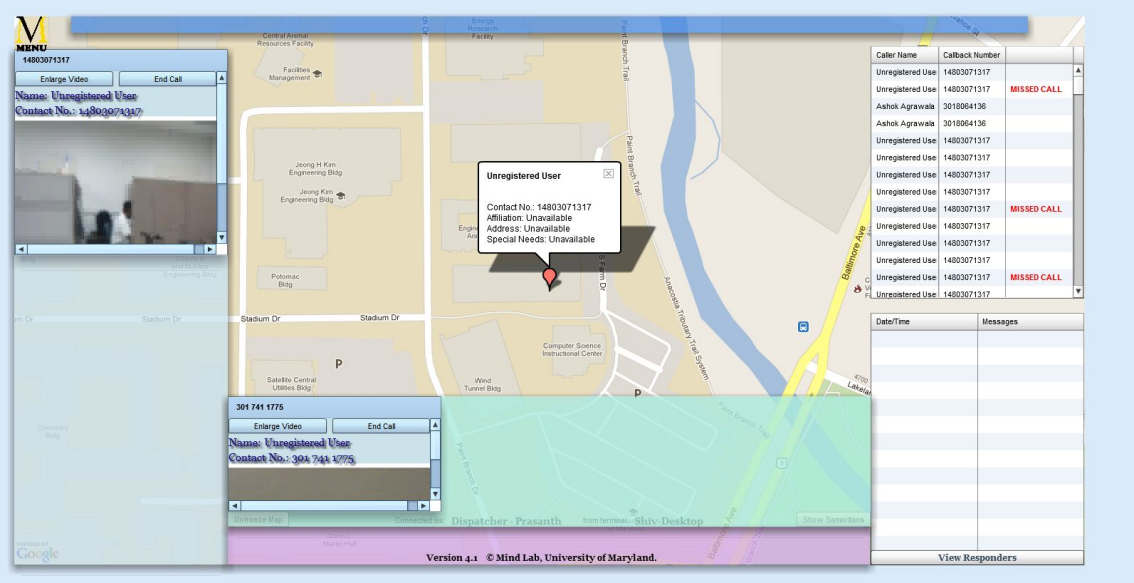

Figure 8.8: Users who have accepted the call from dispatcher are displayed at the bottom of the dispatcher console.

Figure 8.9: Caller application interface when the users are grouped together by the dispatcher.

8.2.1 Prospective Situations

We believe the new feature in M-Urgency will be effective in many situations, few of which are listed below:

- *Situations demanding immediate additional information*: For instance, there has been an M-Urgency call made from indoors the dispatcher can well gather information from users around the building about the situation around the building. For example, a fallen tree blocking the front entrance or huge snow accumulation on the west side of the building that can hinder the rescue process etc.

- *Situation demanding different angles of view of a location*: In the event of a fire or road accident users around the location can stream videos, providing different angles of views or the users can show the dispatcher the situations on the nearby roads leading to the building.

- *Situation demanding immediate assistance*: In case of a medical emergency, a doctor (and user of M-Urgency) spotted in the same building can be requested for immediate assistance.

- *Situations demanding coordinated response*: In situations of hostage or campus shootout, various users within a building can be grouped together for a coordinated effort.

8.3 M-Urgency and RoCoM

In this section we explain M-Urgency with respect to our proposed context model. Figure 3 illustrates the involved entities, their associated context, and their relationships. M-Urgency can be represented in terms of the primitives of our context model as in the following:

1. **Event**: As discussed in section 3, an event triggers the use of the context-aware system. For M-Urgency, a road accident is a good example of a trigger event. An M-Urgency caller contacts the police dispatcher and starts the flow of video, audio, and location context into the system. This triggers the use of the system to inform and seek help of the emergency personnel, by the M-Urgency caller making a call to the police dispatcher.

2. **Entities**: Entities are the different players involved in the event. Each of the entities may have contextual information stored about them in the system. We list a few of the entities involved in the above event, along with some of their relevant contextual information:

    - *Caller* - age, location, medical history(of the victim) etc.
    - *Dispatcher* - availability, serving jurisdiction etc.
    - *Police or Medical responder* - availability, experience/expertise, location etc.
    - *Vehicle* involved in the accident - make and model, color etc.
    - *Place* of accident - location, whether a highway or a local road etc.

3. **Activities**: An event template may include numerous activities aiming at achieving the goal, created by the event. For a traffic accident, the goal is to ensure safety and well being for the involved parties. The event template has a series of activities to fulfill the goal. Each activity brings in some change in the context of the entities involved in it. A few of the activities and the subsequent contextual changes are listed below:

    - *Call the police dispatcher*: the status of the dispatcher becomes busy or unavailable for another call,the context or role of the caller changes from victim or witness to an ”informer”
    - *Dispatcher assigns police responder to the accident*: the status of the police responder changes to busy or assigned, dispatcher becomes available for the next call
    - *Vehicle is towed away*: the context of the place changes from traffic block to slowly moving traffic, the context of the medical personnel changes to as ”attending a patient”

4. **Relationship**: The primitives have some relationship between them based on the situation and the role that they play in the situation. The relationships between the primitives involved in the accident are:

    - *caller-event*: caller is a witness of the event. (It could be the victim also.)
    - *vehicle-passenger*: the passenger is the owner of the car (thus, some information about the passenger could be obtained understanding this relationship).

- *dispatcher-officer/medical responder*: the responders accept the dispatcher's requests and are available for service
- *car-place*: car will have a relationship with the place as a hindrance to the traffic flow
- *medical responder-passenger*: has the ability to access or even update his/her medical records

5. **Templates**: The primitive template module and activity manager in Rover-II core, function based on the templates. As an M-Urgency connection is established with Rover-II, the activity template for the same is instantiated by the activity manager. A list of activities for the *MUrgencyEvent* are instantiated from the ontology. These set of activities are executed by Rover-II. The primitive template module instantiates all the involved entities from the ontology, based on the templates designed for the entities with respect to M-Urgency.

6. **Rules**: The reasoning engine and the relevant context module function based on the rules specified. A simple rule could be with regards to the status of the dispatcher, who is permitted to accept only one call at a time. A *busy* status compels an incoming M-Urgency call to be put on hold. Another example could be the activity specified to handle call failures due to connection failures to the media server. In that case a default activity to instruct the M-Urgency caller application to force a regular 911 phone call is initiated. Many such rules have been and can be specified in Rover-II ontology with respect to each application.

8.4 M-Urgency and Rover-II

In this section we explain how the M-Urgency applications make use of Rover-II as an emergency call is made. Here, we provide only an instance of what happens in Rover when a connection is made from an M-Urgency caller. The overall functioning of the system is not discussed here due to space constraints. Consider the event of a road accident, the caller application establishes a connection with the Rover server. The Rover Core maintains an Event Template for the different applications designed to work with Rover. Once it recognizes a connection coming in from an M-Urgency caller application, it conveys to the Controller, the set of activities that are to be executed immediately. The activities could be: inform the dispatcher application about the caller, establish a connection between the caller application and the streaming server (service tier) to begin the audio/video transmission, fetch the user's information from the context storage etc. The Controller instantiates these activities. The Activity Manager manages each activity by informing the change of context of each primitive involved, to the Controller. For example, once a call is forwarded to a dispatcher application, the context of the dispatcher changes to 'busy'. Any further immediate call is to be forwarded to another connected dispatcher. The Activity Manager invokes the Relevant Context module to filter the relevant contextual information about the user. When an M-Urgency call is made, the user's medical history would be a piece of relevant information but not his choice of food or hobbies etc. Also, contextual information like the location of the caller is obtained from the application. The relevant information is provided to the Dispatcher application along with the audio/video stream through the streaming server, thus establishing a full connection

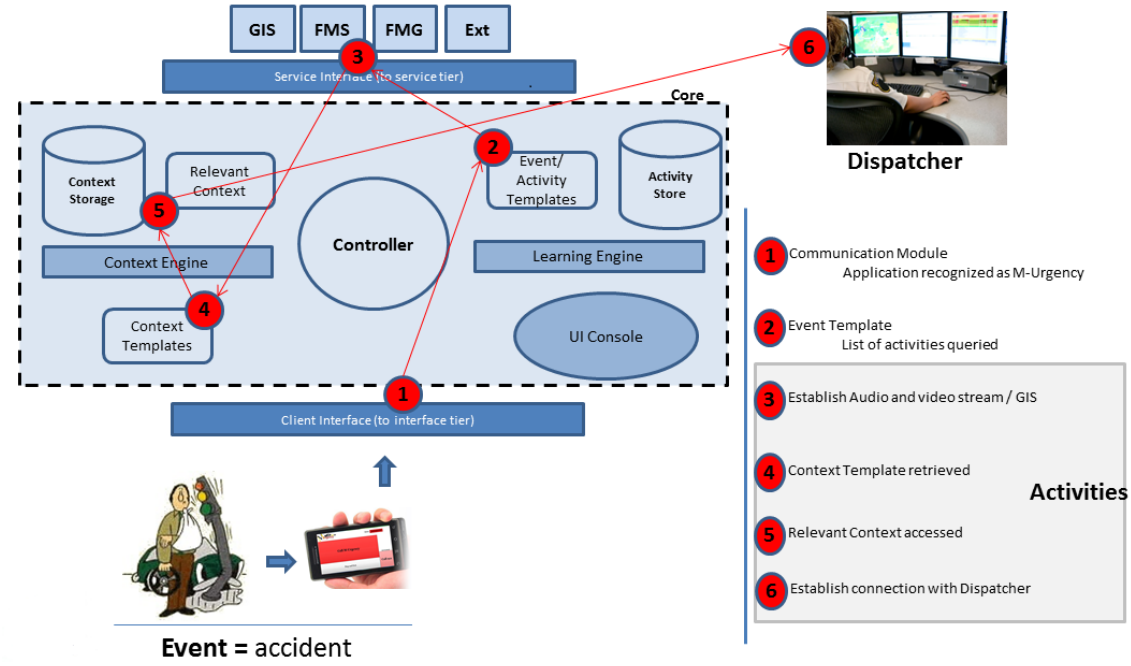

Figure 8.10: Control flow on Rover-II as M-Urgency caller application initiates a call and a Dispatcher application is already connected.

from the caller to the police department.

8.5 M-Urgency and our Information Model

The Information Model, discussed in 7 can be well explained with the help of the M-Urgency system. Rover takes up the role of the ”I” from 7.1, acting as the information center for the whole system. Rover maintains static contextual information in its context storage and also gathers the dynamic context information from the clients connected to it.

The role of Information Integration Point (IIP) is played by the Dispatcher Console. Emergency dispatchers try to make use of as much contextual information provided to them to take quick critical decisions for initiating relief activities. Say, when a call comes

in from the caller application regarding a road accident, the dispatcher console is provided with contextual information, few of which are listed below:

- The caller's information like age, gender etc.
- Special needs/ disabilities that the caller may have. For example, the caller is allergic to certain medications or has poor eye sight without glasses etc.
- Callers' medical history is made available
- The video from the callers side provides a lot of visual information based on which decisions can be made
- Caller's real time location
- Traffic information around the place of accident, so that personnel dispatched to attend the accident victims can take a route accordingly
- List of available responders organized such that the nearest one is pushed to the top
- The weather condition at the location. Say, if it is snowing, certain precautions need to be taken or if a fire incident, the wind gust and the direction are critical pieces of information.
- Twitter tweets can be pulled in, gathering additional information about an event.

Based on the contextual information, the dispatcher is enabled to take very quick decisions. At the same time the dispatcher is also provided with an interface wherein he could very easily access the information and initiate the activities with mere gestures like a mouse click operation or drag-and-drop operation etc., which is important.

Below are some screen captures of the M-Urgency dispatcher's console showing how the dispatcher gathers additional information about the situation with simple textual descriptions/commands. This provision in M-Urgency was implemented as a proof of the concept. We believe this provision would be more effective if augmented with voice recognition, automation with sensors, enhanced with Natural Language Processing techniques etc.

Figure 8.11 shows a screenshot of the M-Urgency dispatcher with the contextual information panes, wherein he/she could opt to gather additional information from the menu on the top left corner. A text box is provided at the bottom of the console along with the contextual information pane on the right side of the screen. The dispatcher could either describe the incident or issue simple commands to gather additional contextual information. As an M-Urgency call is made, Rover-II considers the context of the event, being an MUrgencyEvent, and aggregates the basic information that the police department usually requires when an emergency call is made. The caller's full name and the call back number is automatically made available to the dispatcher as shown in figure 8.12. Figure 8.13 is the screenshot of the contextual information pane that aggregates some additional information about the caller when required by the dispatcher. The dispatcher could similarly describe the situation specifying queries/comments like "health issues", as shown in figure 8.14 wherein Rover-II aggregates medical information about the user. The dispatcher could, otherwise, seek twitter tweets giving basic description about the incident as in figure 8.15. Figure 8.16 shows the aggregated contextual information based on the various queries/descriptions given by the dispatcher.

The primitive module in the context tier of Rover-II instantiates all the entities directly

related to the concerned entity, to make contextual information access efficient. Say that the M-Urgency call was made related to the fire event and the caller is in danger. The dispatcher may want to pull in the information about some people related to the caller who could be informed about the situation. As in figure 8.17, the dispatcher describes the situation and Rover-II provides the dispatcher with contact information of the callers' parents/friends (figure 8.18).

Since a fire incident is reported, Rover-II can pull out building information in the close vicinity of the caller's location. As shown in figure 8.19, the dispatcher is shown on the map the nearby buildings that have hazardous materials stored. The dispatcher may have to make important decisions of cordoning off areas or evacuation based on that. The weather information provides the dispatcher with the wind gust speed and directions, so that the dispatcher could make sure whether or not the fire has the potentiality to spread towards very intimate buildings with hazardous storage. Likewise, Rover-II provides contextual support to the Information Integration Point (IIP), so that the situation is understood as much as possible and educated critical decisions are made based on that.

Please note that these are sample pieces of information for the purpose of demonstration and only a minimum amount is made use of to keep the example simple. As much information can be stored and made use of in real scenario.

The fire incident is the event in the information model that initiates the iteration. The event itself acts as a source of lot of information regarding the accident. As the event takes place, information is gathered at Rover and provided to the IIP (Dispatcher). As mentioned before, the static contextual information will be available at the Rover context storage, and the dynamic context information is gathered from various sources like the

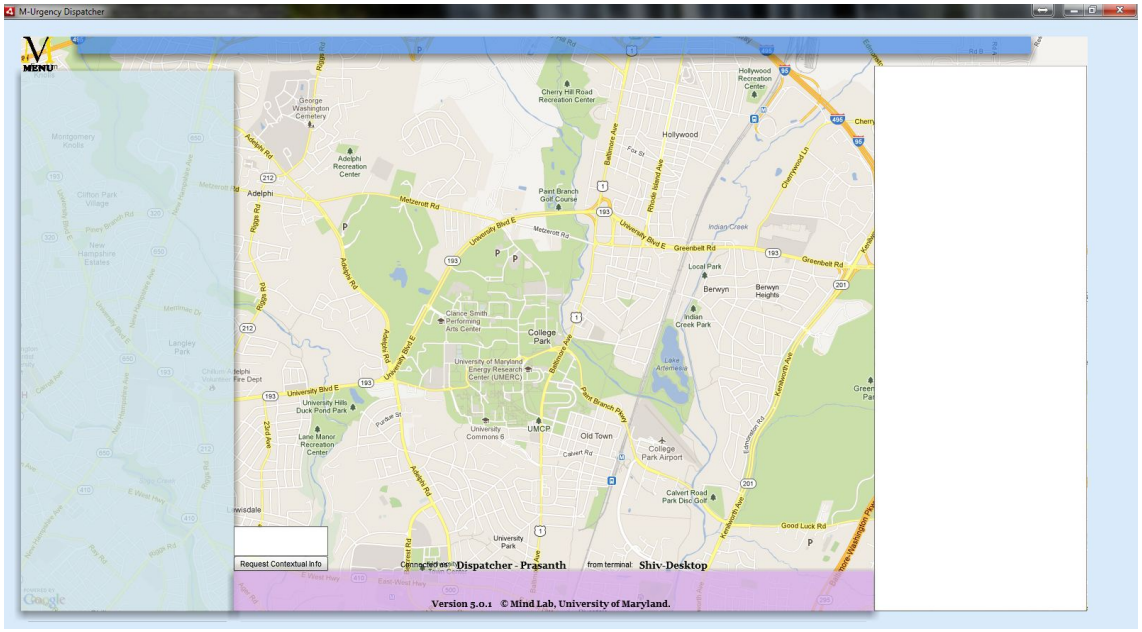

Figure 8.11: A screenshot of the M-Urgency dispatcher with the contextual information panes

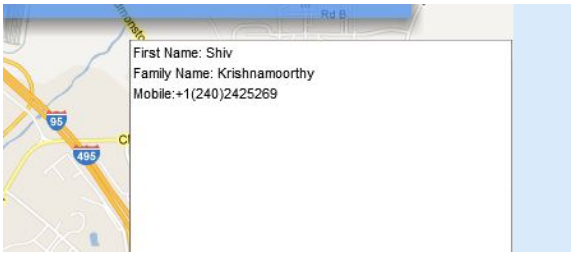

Figure 8.12: Considering the context of the event, initial contextual info is provided to the dispatcher

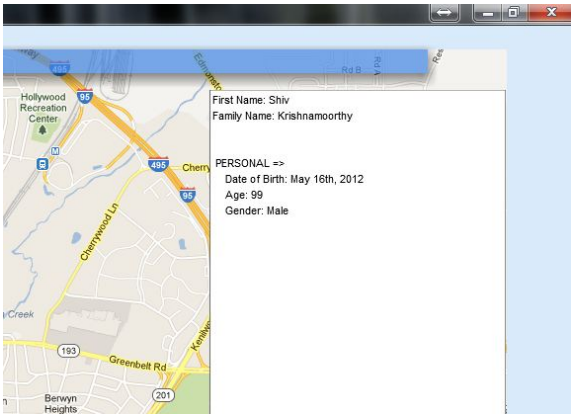

Figure 8.13: Additional information about the caller, when sought by the dispatcher

Figure 8.14: Dispatcher seeks any health issues the caller may have

Figure 8.15: Dispatcher seeks tweets related to the incident to get a better picture

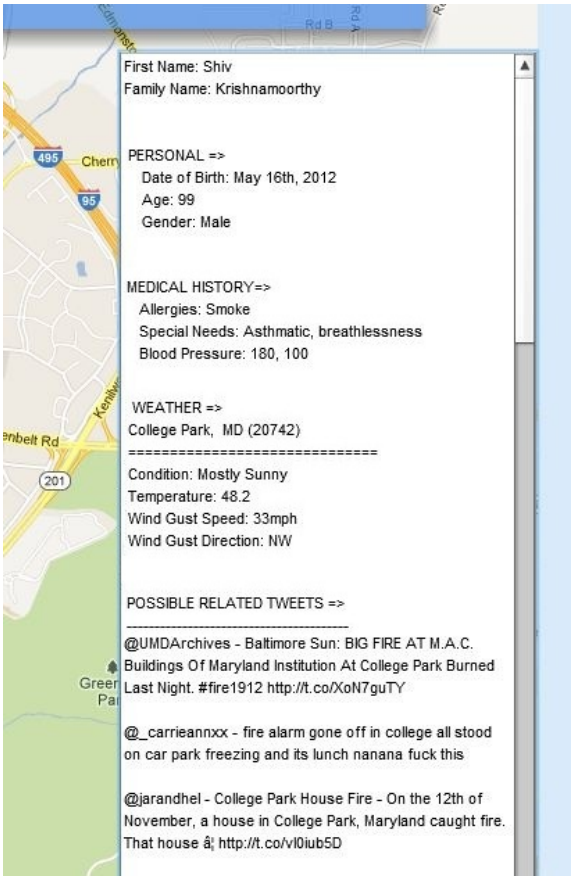

Figure 8.16: Contextual information about the caller and the scene is provided to the dispatcher based on descriptions/queries put forth

Figure 8.17: Dispatcher reckons that the caller is in danger and informs Rover-II about it

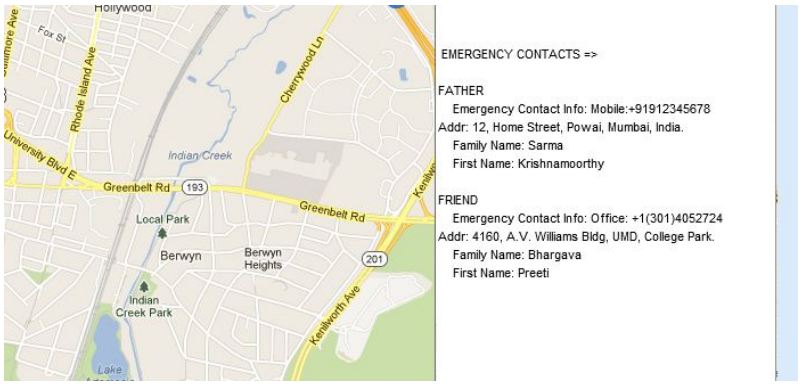

Figure 8.18: Rover-II provides emergency contacts list to the dispatcher

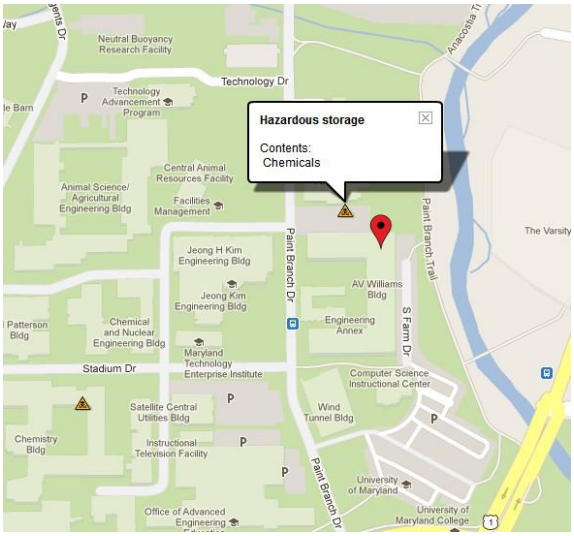

Figure 8.19: Buildings with Hazardous Storage pointed out to the dispatcher in the event of a fire

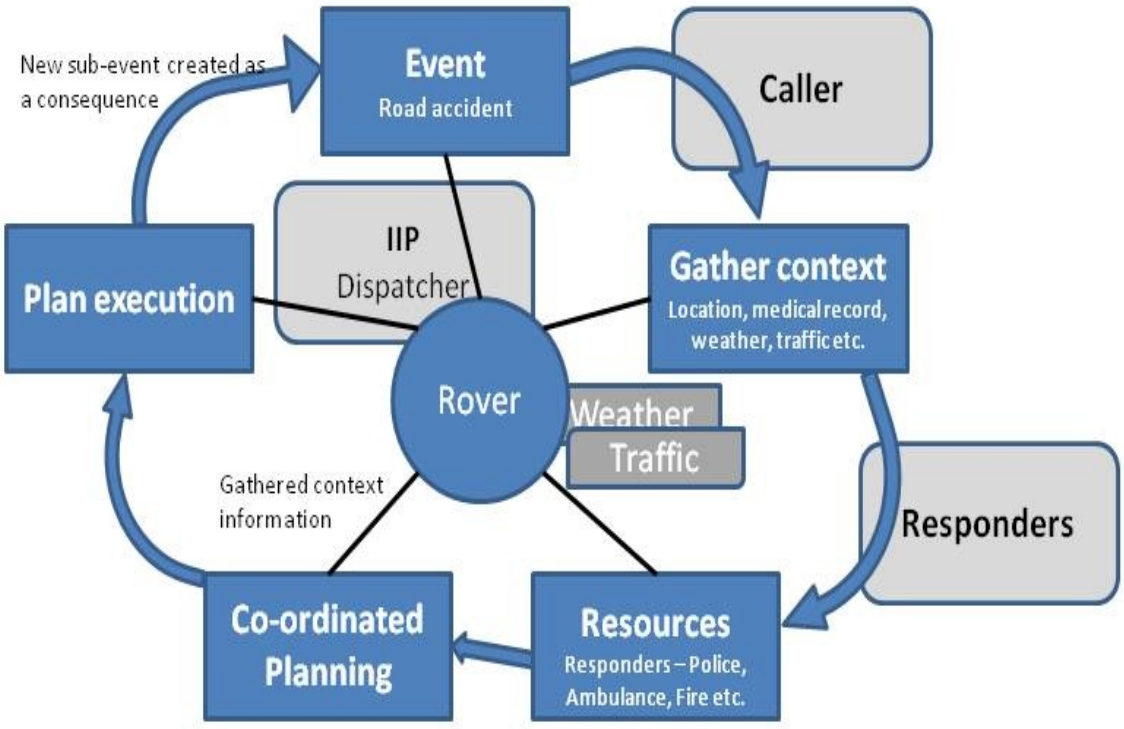

Figure 8.20: An instance of M-Urgency with respect to our Information Model

caller application itself, responder personnel applications, location server, weather server, traffic server etc.

The Dispatcher has a list of resources available, in terms of police officers, medical responders, fire responders etc. The Dispatcher initiates a coordinated planning between the police, ambulance and fire personnel. Finally the plan is executed. Each and everyone of the five nodes have to interact with Rover, either providing information or accessing information which explains why in 7.1 we depict the inter-relationship between every node with each other and the ”I” in the center.

8.6 Status and Development Details

All the functionality of the application, the important ones being presented here, has been implemented and tested. The pilot of the system has been deployed at the University of Maryland Police Department, College Park, MD, USA. The caller applications are available to the faculty, staff and students of the University community. Only the very basic functionalities of the system are discussed in this chapter. Additional functionalities and customizations have been incorporated in the actual deployed version. We are also in the process of customizing the system to many other interested City/University police departments. Many such agencies have expressed their interest in deploying M-Urgency system in their respective jurisdictions.

M-Urgency has been developed using Adobe Flex open source framework supporting desktop/laptops, android phones/tablets and iOS phones/tablets. We use Adobe Flash Media Server for the streaming of videos and establish a SIP connection with the E-911 system of UMPD through Adobe Flash Media Gateway. Other major components of the system include MySQL, PHP and Apache based web server.

Chapter 9

Distributed Implementation of Rover-II

Another direction where efforts have been put forth by us is in designing and developing a distributed architecture for Rover-II. We have attempted to do so, especially addressing concerns regarding M-Urgency applications connecting to Rover-II. We believe that designing the distributed architecture keeping a complicated system like M-Urgency in mind would make it robust enough to handle other simple applications. Considering M-Urgency being an application that would be made use of during critical emergency situations, we discuss below some of the challenges that we face now or expect to face in the near future as the system is rolled out to the University community:

- *Bulky Data*: For every call made, we deal with real time audio/video stream which consumes high bandwidth. The police department requires good quality video which would facilitate eventual investigations.

- *Occasional, but critical heavy call rate*: Though, practically, not many emergency calls could be made simultaneously, we have learnt from the police department that on particular days (usually Thursday to Saturday) they experience a spike in the call rate. Also, an unfortunate event during a public gathering could produce heavy call rate.

- *Centralized*: The whole system is dependent on a Rover and a Streaming server. Failure of either of them would bring the whole system down; an uncalled for de-

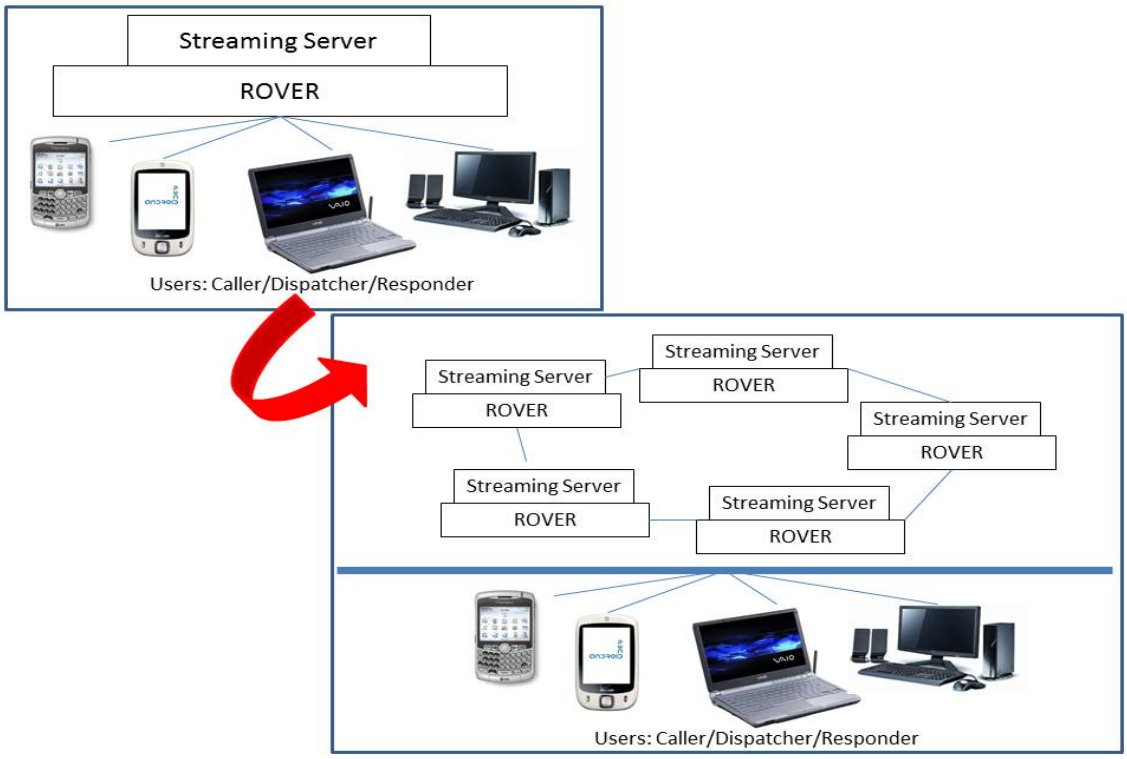

Figure 9.1: Moving from the centralized architecture to a distributed architecture

velopment in an emergency situation.

One of the most significant issues that we have noticed in existing system is that when the number of calls on an average, exceeds 20, we experience issues such as significant delays in the stream, frequent freezing of audio/video and the overall performance degradation in the Dispatcher and Responder applications.

Taking this into account, we propose a distributed architecture for M-Urgency wherein a single Rover server is replicated and organized as a peer to peer network as depicted in Figure 9.1

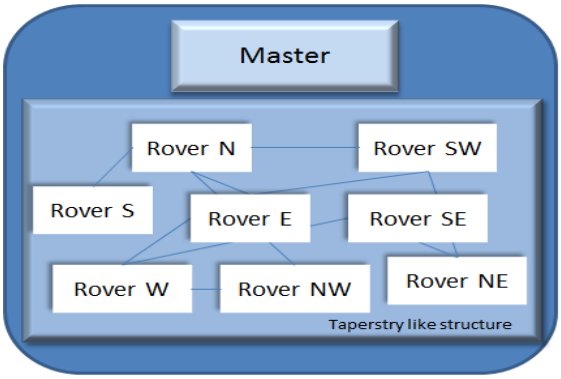

Figure 9.2: A partially centralized approach in distributing Rover

9.1 Distributed Architecture

9.1.1 System Design

Our approach derives inspiration from Tapestry [149] [150] to design the distributed system. Each Rover is assigned a unique nodeid that is generated using Secure Hash Algorithm - SHA-1 [108]. Though the Rover servers are organized in a decentralized manner, as in Tapestry, our approach is partially centralized as shown in Figure 9.2. A Master node maintains certain global information, which the Rover servers share within themselves. Following a partially centralized approach helped us reduce the routing cost within the network as it substantially minimizes the number of messages sent between the Rover servers. We discuss the role of the Master node eventually.

Each Rover server (at the Interface tier) maintains the following:

- *Routing table* - identical to that in Tapestry, the table has the entries of the neighboring nodes. As SHA-1 generates 40 digit hexa-decimal nodeid for each Rover server, the routing table has rows ranging from 0 to 39 and columns ranging from

0 to F. An entry in the m^{th} row and n^{th} column signifies that the nodeid of that particular node shares the same first n digits with the node owning the routing table and the n+1th digit is m. For example, a node with nodeid as 102ABD will have 102CD2 entered in the routing table with row=3 and column=C, because they share the same first three digits (102) and the 4th digit is C.

- *Threshold value* - a limit on how many calls can the particular Rover server handles at a point of time. All the Rover servers maintain the same threshold at any given time and the master intimates each of them to increase the same as and when required.

- *Reject list* - once the number of incoming calls goes beyond the threshold, the Rover Server forwards the excessive call loads to other Rover servers. If the recipient Rover server rejects a forwarded call, the same is maintained in the Reject list, until the threshold value is updated globally.

Addition of a Rover server node into the network is the same as in Tapestry as explained in [149] and we handle the case of voluntary deletion of the node from the network in a similar way in the paper.

9.1.2 Master Node

Though the Rover servers are organized in a decentralized, distributed fashion, we found it important to have a partially centralized approach to maintaining certain global information. This substantially brought down the number of communication messages

between the Rover servers, thus reducing the routing cost. The Master node is assigned the following responsibilities:

- Direct incoming calls to appropriate Rover server based on the location of the caller
- Re-route calls to other Rover servers when either the appropriate rover server fails or has reached the threshold value.
- Receive regular pings from the Rover servers and keep track of number of calls handled by them
- Maintain the global threshold value wherein when every Rover server reaches the threshold, they are all instructed to increase their respective thresholds

Fig 9.3 shows the pseudo code for the Master node. The Master can receive a call request from a caller wherein it calculates the nodeid of the Rover server from the location of the caller and return the information like the ip address and the port number. The caller device then initiates a direct connection with the respective Rover server. In case the concerned Rover has failed, the details of the least busy Rover server are returned.

The Master could also receive a call forward request from a Rover server. The least busy Rover server is assigned the particular call.

The Master receives the heartbeat ping messages from every Rover server. The Rover servers intimate the Master about the number of calls they are handling at that point of time. If all the Rover servers have reached their threshold, the Master informs them to increase the threshold by the ”initial threshold” value. This ensures that the incoming calls are well distributed before each Rover server can begin taking more calls. In case

any particular Rover server does not ping the Master for more than 1 second, the Master treats it as a failure and informs all the other Rover Servers about the failure.

9.2 Algorithms

In this section we explain our approach to maintaining the load balance of incoming calls in the distributed system and also handling cases of failures of either of the Rover server nodes or the Master node.

9.2.1 Load Balancing

Figure 9.4 presents the algorithm for each of the Rover server, on how do they handle the incoming calls directed to them and share the overload with the other Rover servers.

When a Rover server receives an incoming video call from a client, it simply accepts it if the number of calls it is currently handling is below the threshold. Otherwise, it forwards the call to the node having the longest matching prefix nodeID in the routing table. If this neighboring node is incapable of accepting calls, it sends a reject message after which the current node adds the call to the Reject List so that no forwarding attempt is made to the particular node (until the threshold value is increased globally). If the host node is unable to find a free neighboring node to forward the call to, it forwards the call to the Master. The Master then allocates a Rover server to handle the call.

Besides handling calls, the Rover servers also periodically ping the Master by sending heartbeat messages every 100ms, intimating the Master about the number of calls it is currently handling. Figure 9.5 provides the pseudo code for this function. After every

```
initialThreshold=n
currentThreshold=initialThreshold
While(true)
{
Wait for request
If(request from caller)
{
      if(appropriate Rover is alive)
              return appropriate Rover Info
      else
        return info about first Rover from the list with least
        clientCount
}
If(ping from a Rover)
{
      update the client count value
      reset corresponding timer
      acknowledge ping
      if(client count of all Rover = currentThreshold)
      {
               currentThreshold+=initialThreshold
               update all Rovers about new threshold
      }
}
if(forward request from a rover)
{
       select first Rover from the list with least clientCount
}
if(any of the rover timer has exceeded 1 second)
{
       Inform all Rovers about the failure
       Remove the entry of the rover information
}

}
```

Figure 9.3: Pseudo code for Master node

```
while(true)
{
If (connection request  arrives & current connections <
threshold)
{
     Accept call request.
}
else if (connection arrives from client)
{
     flag=false;
     Repeat until request succeeds
     {
          Select longest matching prefix not in reject list.
          Request call forwarding.
          If rover is ready,
                  forward call.
                  flag=true;
          Else if declined,
                  Add nodeid to reject list.
     }
     if(flag==false)
          forward to Master
}
else if( connection request form rover)
{
     reply "reject"
}
}
```

Figure 9.4: Routing algorithm at each Rover node

third ping, the Rover server waits for a response from the Master. If no response is received, the Master is considered failed and all future communication is directed to the Master2 (backup Master). The Rover server also increases its threshold as and when the Master sends the threshold update message.

9.2.1.1 Initial Load Balancing

As an attempt for better load balancing, the system has been designed in such a way that the incoming call is directed to a particular Rover server based on the location from where it originates. The Rover servers obtain their nodeid by secured hashing a string describing the geographical region. For our experiments we considered 8 regions described

```
masterPingCount=0
while(true)
{
      ping Master updating with client count
      masterPingCounter++

      if(masterPingCounter==3)
      {
            set timer
            Wait for response from Master
            stop timer
            if(changed threshold)
                  update threshold.
            masterPingCounter=0
      }
      TimerTimeOut()
      {
            change ip to secondary Master.
      }
}
```

Figure 9.5: Heartbeat ping message to the Master from Rover node

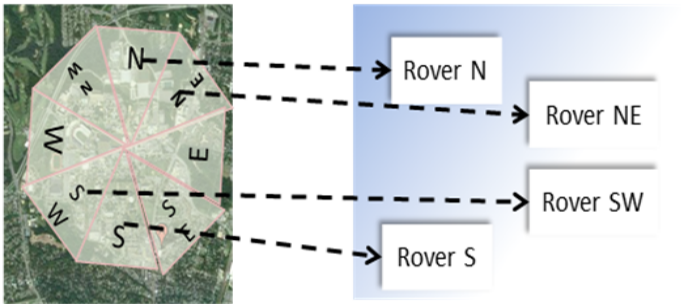

Figure 9.6: Incoming calls directed to particular Rover servers based on the callers' location

as S (south), N (North), NW (Northwest) etc. Thus the load balancing of the incoming calls begins right at the incoming stage itself reducing the effort put forth by the individual Rover servers to do it, as shown in Figure 9.6

9.2.2 Fault Tolerance

We can expect failures at two levels (a) any one of the Rover Servers fails or (b) the Master node fails. We have devised mechanisms for handling both the cases which we

discuss in this section.

9.2.2.1 Rover Server Failure

Master Side Recovery

Whenever a Rover server goes down, the master detects it if it doesn't receive a ping from the it for 1 second. It updates the local information regarding the particular Rover server and informs all other Rover servers to update their routing tables.

Rover Side Recovery

The other Rover servers remove the entry of the failed node from their routing tables when they are informed by the master.

Client Side Recovery

When initiating a call, the Master automatically redirects the client to another appropriate Rover server. In case of a call that was already being handled by the failed Rover, the client reconnects to the Master. The Master then provides the information of the new Rover that is responsible for handling the client and the connection is established.

9.2.2.2 Master Failure

To ensure reliability in the event of Master's failure, the distributed system comprises of a secondary Master. The primary Master periodically sends its bookkeeping statistics (like threshold and client-counts) to the secondary Master. This ensures that at almost all times the secondary Master has the necessary information to take over from the primary.

Rover Side Recovery

When a master node exits or goes down abruptly, the Rovers detect Master's failure by a timeout in not receiving an acknowledgement for the pings and update their Master IP to the secondary Master.

Master Side Recovery

When the secondary Master receives a ping from any of the Rovers, it realizes its role as the master and becomes the new Master for the system.

Client Side Recovery

When a client detects that the primary Master has gone down, it automatically tries to connect to the secondary Master for any further communications.

9.3 Experiments and Results

Our objective at this point was to simulate the Interface Tier of the Rover server and see how they organized themselves in the distributed setup. Our main focus, in this work, is on how the incoming calls traffic is handled and Rover servers balance the load; and how quickly does the system recovers to a consistent state in case of a failure.

9.3.1 Load Balancing

We considered two scenarios - (1) when the calls come in a random order from the different regions and (2) a less probable but more critical case, where all the calls originate from the same region (for example, an unfortunate incident occurs in a public place and many callers call in to report the same).

9.3.1.1 Calls from same region

We simulated 119 calls to originate from the same region, so that they are all directed to Rover S (south). We monitored the load balance at intervals after 40 calls, 97 calls and finally 119 calls. We performed the same experiment setting the initial threshold value as 5 and 3.

Tables 9.1 and 9.2 show the distribution of the calls within the eight Rover servers. It is clear from the tables that our algorithm produced a good distribution of the incoming calls. Instead of Rover S handling all the 119 calls, the load was well balanced with each Rover server handling a maximum of only 15 calls. The difference in the threshold did not show a significant difference in terms of distribution of the calls, but the number of routing messages between the Rover servers and the Master was observed to be less in case with threshold=3.

9.3.1.2 Calls from different regions

We, then, simulated 43 calls to originate from the eight different regions as shown in Figure 9.7. The calls were made to originate one by one, in a round robin fashion, beginning from the North region. Table 9.3 and 9.4 show the distribution of the calls within the 8 rover servers.

One interesting thing to notice here is the number of routing messages. The number of routing messages is significantly less than when the calls originated from the same region. This shows that our initial load balancing approach was effective. Since the calls are directed to their respective Rover servers, based on the location of the caller, the

After 40 calls	**S**	**N**	**NW**	**E**	Routing Messages
	5	5	5	5	
	W	**SE**	**NE**	**SW**	65
	5	5	5	5	
After 97 calls	**S**	**N**	**NW**	**E**	Routing Messages
	15	15	15	12	
	W	**SE**	**NE**	**SW**	132
	10	10	10	10	
After 119 calls	**S**	**N**	**NW**	**E**	Routing Messages
	15	15	15	15	
	W	**SE**	**NE**	**SW**	172
	15	15	15	14	

Table 9.1: Distribution of incoming calls within the Rover servers (initial threshold=5)

After 40 calls	**S**	**N**	**NW**	**E**	Routing Messages
	6	6	6	6	
	W	**SE**	**NE**	**SW**	50
	4	4	4	4	
After 97 calls	**S**	**N**	**NW**	**E**	Routing Messages
	13	12	12	12	
	W	**SE**	**NE**	**SW**	132
	12	12	12	12	
After 119 calls	**S**	**N**	**NW**	**E**	Routing Messages
	15	15	15	15	
	W	**SE**	**NE**	**SW**	163
	15	15	15	14	

Table 9.2: Distribution of incoming calls within the Rover servers (initial threshold=3)

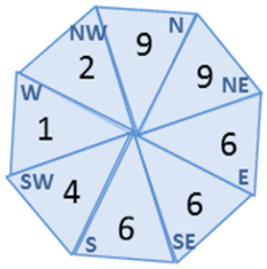

Figure 9.7: Number of calls originating from each region

S	**N**	**NW**	**E**	Routing Messages
5	6	5	5	25
W	**SE**	**NE**	**SW**	
5	5	7	5	

Table 9.3: Distribution of incoming calls originating from different regions (initial threshold=5)

S	**N**	**NW**	**E**	Routing Messages
6	6	3	6	17
W	**SE**	**NE**	**SW**	
6	6	6	4	

Table 9.4: Distribution of incoming calls originating from different regions (initial threshold=3)

number of routing messages sent between the Rover servers is significantly low as the Rovers automatically receive the incoming calls in a distributed manner. We can compare the number of messages after 40 calls in Table 1 with the number of messages in table 3 (after 43 calls). With initial load balancing, the number of messages is about 62

9.3.2 Fault Tolerance

We conducted experiments with 3 scenarios. In scenario 1, we brought down a Rover and observed the behavior of the system. We noticed that the system converged to a consistent state within 2 seconds (i.e. the other Rovers updated the routing information accordingly). In scenario 2, we brought down a Master and observed that the Rovers detected the failure and contacted the secondary Master within 2.5 seconds. In scenario 3, we brought down a master and a rover together and observed that the system converged to a consistent state within 4.5 seconds.

We noticed that calls originating from client, within the above mentioned period of recovery time, were handled successfully by the Master by forwarding it to another active Rover.

9.3.3 Inferences

We list a few inferences that we have made from our experimental results:

- Our approach does produce a good level of distribution of incoming calls within the Rover servers, thus enabling good load balancing.

- The initial load balancing helped us significantly bring down the number of routing

messages, thus reducing cost.

- We also observed in our experiments that the performance of the system depends on:
  - Number of Rover server nodes in the network
  - The way the network is formed (with links between the Rovers) based on the nodeid assigned to them
  - Number of neighbors a node, that receives significantly high number of calls, has.
  - The initial threshold value

9.4 Related Work

A lot of emphasis has already been made, in the literature, regarding the significance of a distributed architecture over a centralized approach. Many have exhibited the advantages of a distributed approach like in [109][40].

The first generation of peer-to-peer applications, like Napster and Gnutella had restricting limitations such as a central directory for Napster and scoped broadcast queries for Gnutella limiting scalability [112]. To address these problems a second generation of P2P mechanisms were developed like Tapestry [149][150], Chord [126] , Pastry [110], and CAN [106]. Our approach derives its approach from that of Tapestry. Tapestry is an extensible infrastructure that provides decentralized object location and routing focusing on efficiency and minimizing message latency. This is achieved since Tapestry constructs

locally optimal routing tables from initialization and maintains them in order to reduce routing stretch. Chord is a protocol and algorithm for a peer-to-peer distributed hash table. A distributed hash table stores key-value pairs by assigning keys to different computers (known as ”nodes”); a node will store the values for all the keys for which it is responsible. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key. Pastry is an overlay and routing network for the implementation of a distributed hash table similar to Chord. The protocol is bootstrapped by supplying it with the IP address of a peer already in the network and from then on via the routing table which is dynamically built and repaired. Because of its redundant and decentralized nature there is no single point of failure and any single node can leave the network at any time without warning and with little or no chance of data loss.

A number of approaches for distributed video streaming have been discussed in the literature [92][91][[26], addressing issues of source and channel coding, implementation of transport protocols, or modifying system architectures in order to deal with delay, loss, and time-varying nature of Internet. Our focus is mainly on how to bring in a p2p approach in handling and routing of video streams.

Coolstreaming [112] is a P2P data driven Overlay distributed users. Notable features of the protocol network for live media streaming that achieves good streaming quality for globally include its intelligent scheduling algorithm that copes well with the bandwidth differences of uploading clients and thus minimizes skipping during playback, and its swarm-style architecture that uses a directed graph based on gossip algorithms to broadcast content availability.

Though the critical parts of our system are totally decentralized, we drew inspiration of having a partially centralized approach from [80][51]. Though they adopted this approach in a different context, it proved beneficial for us.

Chapter 10

Conclusion and Future Work

We believe we have laid a strong foundation by introducing our context model - Rover Context Model (RoCoM), based on our ontology - Rover Context Model Ontology (RoCoMO) and proposed a middleware approach by designing and implementing a general framework - Rover-II that can (i) efficiently maintain, represent and integrate contextual information, (ii) act as an integration platform where different applications can share contexts and (iii) provide relevant services to make efficient use of the contextual information. We presented *Rover Context Model (RoCoM)* that is generic, extensible and practically usable; structured around four primitives: *entities, events, relationships, and activities*. RoCoM has been designed based on our ontology *Rover Context Model Ontology (RoCoMO)* - an efficient mechanism to represent, store and manage contextual information. We introduced the concept of *Relevant Context* - context of context, to efficiently and effectively model and understand the context of a given situation. Our concepts has taken shape in form of *Rover II*, a middleware for contextual information integration, that could be used not just to store and represent contextual information, but also integrate relevant services to efficiently use the same. We discussed our initial step towards the design and implementation of a distributed architecture for Rover-II, following a P2P arrangement inspired by Tapestry. We finally presented *M-Urgency* - a next-generation public safety system, a practical application as proof of our concepts.

10.1 Contribution

The main contribution of our work has been towards the following:

- Introducing and conceptualizing a generic, extensible and practically usable context model - *Rover Context Model (RoCoM)* that is structured arou6nd four primitives: entities, events, relationships, and activities; built using *Rover Context Model Ontology (RoCoMO)*.

- Introducing the concept of *relevant context* and implementing the same using relevance filtering, context templates and context engine for efficiently handling and making best use of context information.

- Designing, implementing and deploying a context middleware - Rover-II, an advance to Rover [6][7], based on our context model which serves as a framework for contextual information integration, that could not just be used to store and represent ontological contextual information, but also integrate relevant services to efficiently use the same for applications.

- Developing and implementing M-Urgency, a full fledged public safety system built on Rover-II, as a practical proof of our concepts and ideas. Though this example we have exhibited how an application built on Rover-II can avail the context and services provided by Rover-II. These provisions could be made use of, by other applications, as deemed proper and appropriate for them.

- Initial designing and implementation of a distributed architecture for Rover-II, following a P2P arrangement inspired from Tapestry [149].

10.2 Future work

We have taken firm initial steps, paving a new direction, in the field of context-aware computing. It has been a challenging and audacious endeavor, comprising of multiple research problems that cannot be all addressed in a single PhD dissertation. We are convinced that we have laid a strong foundation and a lot can be built upon it. We have set a good starting point for many more dissertations to come in the future. There are many areas that need to be specifically researched and addressed, some of which we discuss in appropriate sections below:

1. *Context Model Ontology - RoCoMO*

   (a) Lot of information is currently stored along with the ontology itself, which results in data access delays. There has been some work done in storing ontologies in databases [49][151][66][137][10]. Research and implementation of an efficient storage mechanism for Rover-II needs to be done.

   (b) The ontology needs to be expanded as much as possible, adding more of the primitives (entities, activities and events) to it. We see this taking place as more context-aware applications are developed on Rover-II as the entities and activities specific to those applications would get added to RoCoMO.

2. *Middleware - Rover-II*

   (a) Any context-aware system is incomplete until complimented by learning. The *learning engine* module is yet to be designed and implemented. Rover-II will be at its best if it could react to a situation based on knowledge gathered from

previous similar instances and context history. It needs to graduate from being just a reactive system to also a proactive system. It remains an open problem.

(b) Gathering static context needs to designed to make it interactive. An interactive process helps gather better and more information about individuals or other entities. Gathering information from Facebook and Twitter profiles and posts will be very useful.

(c) The User Interface console in the Rover-II core is presently designed to serve the basic purpose. So as to make it effective, a significant amount of work can be done in researching, designing and implementing various visualizations for keeping track of the state of the system, connections, load, networks etc.

(d) It would also be desirable to design and implement a visualization for the situation graph to enable the users to grasp a given situation most effectively.

(e) As part of the Data Service in the Service Tier of Rover-II, there is a large scope for data services that could tap useful information from the many available data sources to understand a situation better. Many more such services can be added to the service tier.

(f) As part of the dissertation, we have discussed only the initial study and feasibility of a distributed architecture for Rover-II. The work can be extended to a full fledged implementation as failures and fault are common in real world scenario and one point failure is not desirable.

(g) Though Rover-II is being developed keeping in mind that it needs to have a full fledged wrapper that enables a third party application developer to make use

of the platform; having the ability to make changes in how Rover-II handles their application. We have addressed this concern to a large extent with the help of the ontologies, wherein all specific instructions to the system could be provided. But we feel, this concern has not been completely addressed yet, considering all angles from a developers point of view.

(h) In twitter service, we have employed very basic filtering techniques, as of now. The same can be enhanced and made more effective making use of filtering based on natural language processing techniques or other third party libraries.

3. *Information Model*

(a) The interface provided at the Information Integration Point (IIP) to gather information can be well enhanced with voice recognition and Natural Language Processing techniques. Steps needs to be taken to enable Rover-II to support such provisions.

Bibliography

[1] Owl 2 web ontology language document overview http://www.w3.org/tr/owl2-overview/.

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

[3] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing*, pages 304–307. Springer, 1999.

[4] Adove Flex ActionScript.

[5] Alessandra Agostini, Claudio Bettini, and Daniele Riboni. Loosely coupling ontological reasoning with an efficient middleware for context-awareness. In *In Proc. of the 2nd Annual Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2005*, pages 175–182. IEEE Computer Society, 2005.

[6] Christian B. Almazan. *ROVER: Architectural Support for Exposing and Using Context*. PhD thesis, University of Maryland, College Park, 2010.

[7] Christian B. Almazan, Moustafa Youssef, and Ashok K Agrawala. Rover: An integration and fusion platform to enhance situational awareness. In *In proc. IEEE International Conference on Performance, Computing and Communications 2007,*, pages 582–587. IEEE, 2007.

[8] Joshua Anhalt, Asim Smailagic, Daniel P. Siewiorek, Francine Gemperle, Daniel Salber, Sam Weber, Jim Beck, and Jim Jennings. Toward context-aware computing: Experiences and lessons. *IEEE Intelligent Systems*, 16(3):38–46, May 2001.

[9] WeatherBug API. http://weather.weatherbug.com/desktop-weather/api.html.

[10] I. Astrova, N. Korda, and A. Kalja. Storing owl ontologies in sql relational databases. *International Journal of Electrical, Computer and Systems Engineering*, 1(4):242–247, 2007.

[11] P. Bahl and V.N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784. Ieee, 2000.

[12] P. Bahl, V.N. Padmanabhan, and A. Balachandran. Enhancements to the radar user location and tracking system. Technical report, Microsoft Research, 2000.

[13] A Bahrani, J Yuan, C.D. Emele, D Masato, T.J. Norman, and D Mott. Collaborative and context-aware planning. In *IEEE Military Communications Conference(MILCOM 2008)*, pages 1–7, Nov 2008.

[14] Jakob E. Bardram. The java context awareness framework (jcaf) – a service infrastructure and programming framework for context-aware applications. In *Proceedings of the Third international conference on Pervasive Computing*, PERVASIVE'05, pages 98–115, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] Russell Beale and Peter Lonsdale. Mobile context aware systems: The intelligence to support tasks and effectively utilise resources. In *IN PROCEEDINGS OF MOBILEHCI 2004*, 2004.

[16] Michael Beigl, Albert Krohn, Tobias Zimmer, Christian Decker, and Philip Robinson. Awarecon: Situation aware context communication. In Anind Dey, Albrecht Schmidt, and Joseph McCarthy, editors, *UbiComp 2003: Ubiquitous Computing*, volume 2864 of *Lecture Notes in Computer Science*, pages 132–139. Springer Berlin / Heidelberg, 2003.

[17] Victoria Bellotti and Keith Edwards. Intelligibility and accountability: human considerations in context-aware systems. *Hum.-Comput. Interact.*, 16(2):193–212, December 2001.

[18] Fredrik Bergstrand and Jonas Landgren. Using live video for information sharing in emergency response work. *International Journal of Emergency Management*, 6:295–301, 2010.

[19] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[20] Preeti Bhargava, Shivsubramani Krishnamoorthy, and Ashok Agrawala. An ontological context model for representing a situation and the design of an intelligent context-aware middleware. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 1016–1025, New York, NY, USA, 2012. ACM.

[21] Preeti Bhargava, Shivsubramani Krishnamoorthy, and Ashok Agrawala. Rocomo: a generic ontology for context modeling, representation and reasoning in a context-aware middleware. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 584–585, New York, NY, USA, 2012. ACM.

[22] Preeti Bhargava, Shivsubramani Krishnamoorthy, Aditya Karkada Nakshathri, Matthew Mah, and Ashok Agrawala. Locus: An indoor localization, tracking and navigation system for multi-story buildings using heuristics derived from wi-fi signal strength. In *Proceedings of 9th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2012)*. Springer, 2012.

[23] C. Bolchini, C. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. Context-addict. Technical report, Dip. Elettronica e Informazione, Politecnico di Milano, 2006.

[24] C. Bolchini, C.A. Curino, E. Quintarelli, F.A. Schreiber, and L. Tanca. A data-oriented survey of context models. *ACM SIGMOD Record*, 36(4):19–26, 2007.

[25] O. Brdiczka, J. L. Crowley, and P. Reignier. Learning situation models for providing context-aware services. In *Proceedings of the 4th international conference on Universal access in human-computer interaction: ambient interaction*, UAHCI'07, pages 23–32, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] S.H.G. Chan. Distributed servers architecture for networked video services. *IEEE/ACM Transactions on Networking (TON)*, 9(2):125–136, 2001.

[27] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003.

[28] H Chen, F Perich, T Finin, and A Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *in Proodeecings of Mobile and Ubiquitous Systems: Networking and Services (MOBIQUITOUS)*, pages 258–267, 2004.

[29] Harry Chen. *An Intelligent Broker Architecture for Context-Aware Systems*. PhD thesis, University of Maryland Baltimore County, 2003.

[30] Harry Chen, Tim Finin, and Anupam Joshi. The soupa ontology for pervasive computing. In Valentina Tamma, Stephen Cranefield, Timothy Finin, Steven Willmott, Marius Walliser, Stefan Brantschen, Monique Calisti, and Thomas Hempfling, editors, *Ontologies for Agents: Theory and Experiences*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 233–258. 2005.

[31] Michael D. Goldsmith Chris P. Rainsford and Paul Prekop. Tiki: A trigger-based infrastructure for knowledge and information sharing. In *In Proceedings of the IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises: Collaborative Business Ecosystems and Virtual Enterprises (PRO-VE '02)*, pages 329–336, 2002.

[32] Anne E. Cook, John E. Limber, and Edward J. O'Brien. Situation-based context and the availability of predictive inferences. *Journal of Memory and Language*, 44(2):220 – 234, 2001.

[33] P. Dockhorn Costa, L. Ferreira Pires, and M. Van Sinderen. Architectural patterns for context-aware services platforms. In *in Proceedings of the Second International Workshop on Ubiquitous Computing (IWUC 2005*, pages 3–19, 2005.

[34] J. Coutaz, J.L. Crowley, S. Dobson, and D. Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[35] J. Coutaz, A. Dearle, S. Dupuy-Chessa, G. Kirby, C. Lachenal, R. Morrison, G. Rey, and E. Zirintsis. Gloss: Global smart space - working document on gloss ontology. Technical report, GLOSS Consortium, 2001.

[36] A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[37] A.K. Dey and G.D. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 workshop on the what, who, where, when, and how of context-awareness*, volume 4, pages 1–6. Citeseer, 2000.

[38] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, 2000.

[39] Richard Dickinson, Roger Marshall, and Steven Helme. Enhance e911 locaiton information using voice over internet protocol (voip), 2005.

[40] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *Network, IEEE*, 13(4):6–15, 1999.

[41] Dejene Ejigu, Marian Scuturici, and Lionel Brunie. An ontology-based approach to context modeling and reasoning in pervasive computing. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, PERCOMW '07, pages 14–19, Washington, DC, USA, 2007. IEEE Computer Society.

[42] Ekahau. http://www.ekahau.com/.

[43] M Endsley. Toward a theory of situational awareness in dynamic systems. *Human Factors*, 1995.

[44] M.R. Endsley. Design and evaluation of situation awareness enhancement. In *Proceedings of Human Factors Society 32nd Annual Meeting*, volume 1, pages 97–101, 1988.

[45] R. Endsley. Designing for situation awareness in complex systems. In *Proceedings of the Second intenational workshop on symbiosis of humans, artifacts and environment, Kyoto, Japan.*, 2001.

[46] ESRI. http://www.esri.com/software/arcgis.

[47] P. Fahy and S. Clarke. Cass–a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*. Citeseer, 2004.

[48] Yu-Hong Feng, Teck-Hou Teng, and Ah-Hwee Tan. Modelling situation awareness for context-aware decision support. *Expert Syst. Appl.*, 36(1):455–463, January 2009.

[49] A. Gali, C. Chen, K. Claypool, and R. Uceda-Sosa. From ontology to relational databases. *Conceptual Modeling for Advanced Application Domains*, pages 278–289, 2004.

[50] Adobe Flash Media Gateway. http://www.adobe.com/products/adobe-media-server-family.html.

[51] S. Ghemawat, H. Gobioff, and S.T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[52] M. Golemati, A. Katifori, C. Vassilakis, G. Lepouras, and C. Halatsis. Creating an ontology for the user profile: Method and applications. In *Proceedings of the First RCIS Conference*, pages 407–412, 2007.

[53] Philip D. Gray and Daniel Salber. Modelling and using sensed context information in the design of interactive applications. In *Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, EHCI '01, pages 317–336, London, UK, UK, 2001. Springer-Verlag.

[54] W.G. Griswold, R. Boyer, S.W. Brown, T.M. Truong, E. Bhasker, G.R. Jay, and R.B. Shapiro. Activecampus-sustaining educational communities through mobile technology. *University of California, San Diego, Department of Computer Science and Engineering, Technical Report*, 2002.

[55] T. Gu, X.H. Wang, H.K. Pung, and D.Q. Zhang. An ontology-based context model in intelligent environments. In *Proceedings of communication networks and distributed systems modeling and simulation conference*, volume 2004, pages 270–275. Citeseer, 2004.

[56] K. Henricksen, S. Livingstone, and J. Indulska. Towards a hybrid approach to context modeling, reasoning and interoperation. In *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management - The First International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 54–61, 2004.

[57] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, PERCOM '04, pages 77–, Washington, DC, USA, 2004. IEEE Computer Society.

[58] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In Friedemann Mattern and Mahmoud Naghshineh, editors, *Pervasive Computing*, volume 2414 of *Lecture Notes in Computer Science*, pages 79–117. Springer Berlin / Heidelberg, 2002.

[59] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9 - Volume 9*, HICSS '03, pages 292.1–, Washington, DC, USA, 2003. IEEE Computer Society.

[60] James Holloway, Elaine Seeman, and Margeret oHara. State agency and local next generation 911 planning and coordination to implement state ng 911 and ip enabled network policies. *Pittsburgh Journal of Technology Law and Policies*, 11, 2010.

[61] P. T. Jaeger, K. R. Fleischmann, J. Preece, B. Shneiderman, F. P. Wu, and Y. Qu. Community response grids: Facilitating community response to biosecurity and bioterror emergencies through information and communication technologies. *Biosecurity and Bioterrorism*, 5:1–12, 2007.

[62] P. T. Jaeger, B. Shneiderman, K. R. Fleischmann, J. Preece, Y. Qu, and F. P. Wu. Community response grids: E-government, social networks, and effective emergency response. *Telecommunications Policy*, 31:592–604, 2007.

[63] Anthony Jameson. Modelling both the context and the user. *Personal Ubiquitous Comput.*, 5(1):29–33, January 2001.

[64] Java. http://www.java.com/en/.

[65] Jena. jena.apache.org/.

[66] D. Jeong, M. Choi, Y.S. Jeon, Y.H. Han, L. Yang, Y.S. Jeong, and S.K. Han. Persistent storage system for efficient management of owl web ontology. *Ubiquitous Intelligence and Computing*, pages 1089–1097, 2007.

[67] Jess. http://www.jessrules.com/.

[68] Xiaodong Jiang, Nicholas Y. Chen, Jason I. Hong, Kevin Wang, Leila Takayama, and James A. Landay. Siren: Context-aware computing for firefighting. *Pervasive Computing*, 3001/2004:87–105, 2004.

[69] M. Kaenampornpan and E. ONeill. An integrated context model: Bringing activity to context. In *Proc. Workshop on Advanced Context Modelling, Reasoning and Management*. Citeseer, 2004.

[70] Panayotis K. Kikiras. An integrated platform for autonomic computing for disaster relief operations. *Journal of Battlefield Technology*, 12(3):29–34, Nov 2009.

[71] Anders Kofod-Petersen and Agnar Aamodt. Case-based situation assessment in a mobile context-aware system. In *Proceedings of Artificial Intelligence in Mobile Systems 2003 AIMS (2003)*, pages 41–49. University des Saarlandes, 2003.

[72] P. Korpipää and J. Mäntyjärvi. An ontology for mobile device sensor-based context awareness. *Modeling and Using Context*, 2680/2003:451–458, 2003.

[73] Robert E. Kraut, Susan R. Fussell, , and Jane Siegel. Visual information as a conversational resource in collaborative physical tasks. *Human Computer Interaction*, 18:13–49, 2003.

[74] Shivsubramani Krishnamoorthy and Ashok Agrawala. A context-aware framework for mobile applications. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 403–404, New York, NY, USA, 2011. ACM.

[75] Shivsubramani Krishnamoorthy and Ashok Agrawala. Contextual information integration platform for humanitarian relief. In *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*, ACWR '11, pages 55–61, New York, NY, USA, 2011. ACM.

[76] Shivsubramani Krishnamoorthy and Ashok Agrawala. M-urgency: a next generation, context-aware public safety application. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 647–652, New York, NY, USA, 2011. ACM.

[77] Shivsubramani Krishnamoorthy and Ashok Agrawala. Context-aware, technology enabled social contribution for public safety using m-urgency. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*, MobileHCI '12, pages 123–132, New York, NY, USA, 2012. ACM.

[78] Shivsubramani Krishnamoorthy, Preeti Bhargava, Matthew Mah, and Ashok Agrawala. Representing and managing the context of a situation. *The Computer Journal*, 55:1005–1019, 2012.

[79] Reto Krummenacher and Thomas Strang. Ontology-based context modeling. In *In Workshop on Context-Aware Proactive Systems*, 2007.

[80] C. Le Pape. A combination of centralized and distributed methods for multi-agent planning and scheduling. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 488–493. IEEE, 1990.

[81] X. Li. Ratio-based zero-profiling indoor localization. In *Mobile Adhoc and Sensor Systems, 2009. MASS'09. IEEE 6th International Conference on*, pages 40–49. IEEE, 2009.

[82] H. Lim, L.C. Kung, J.C. Hou, and H. Luo. Zero-configuration indoor localization over ieee 802.11 wireless infrastructure. *Wireless Networks*, 16(2):405–420, 2010.

[83] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1067–1080, 2007.

[84] M-Urgency. http://m-urgency.umd.edu/.

[85] Alan M. MacEachren, Guoray Cai, Michael McNeese, Rajeev Sharma, and Sven Fuhrmann. Geocollaborative crisis management: designing technologies to meet real-world needs. In *International conference on Digital government research 2006*, pages 71–72. ACM, New York, USA, 2006.

[86] D. Madigan, E. Einahrawy, R.P. Martin, W.H. Ju, P. Krishnan, and AS Krishnakumar. Bayesian indoor positioning systems. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1217–1227. IEEE, 2005.

[87] Natalia Marmasse. commotion: a context-aware communication system. In *CHI '99 extended abstracts on Human factors in computing systems*, CHI EA '99, pages 320–321, New York, NY, USA, 1999. ACM.

[88] Ulrich Meissen, Stefan Pfennigschmidt, Agns Voisard, and Tjark Wahnfried. Context- and situation-awareness in information logistics. In Wolfgang Lindner, Marco Mesiti, Can Trker, Yannis Tzitzikas, and Athena Vakali, editors, *Current Trends in Database Technology - EDBT 2004 Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 448–451. Springer Berlin / Heidelberg, 2005.

[89] Beigl Michael, Krohn Albert, Zimmer Tobias, Decker Christian, and Robinson Philip. Awarecon: Situation aware context communication. In Anind Dey, Albrecht Schmidt, and Joseph McCarthy, editors, *UbiComp 2003: Ubiquitous Computing*, volume 2864 of *Lecture Notes in Computer Science*, pages 132–139. Springer Berlin / Heidelberg, 2003.

[90] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda. Task ontology for reuse of problem solving knowledge. *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*, pages 46–59, 1995.

[91] S. Mungee, N. Surendran, and D.C. Schmidt. The design and performance of a corba audio/video streaming service. In *System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, pages 14–pp. IEEE, 1999.

[92] T. Nguyen and A. Zakhor. Multiple sender distributed video streaming. *Multimedia, IEEE Transactions on*, 6(2):315–326, 2004.

[93] OASIS. http://www.oasis-open.org/.

[94] The Nation of neighbors. http://www.nationofneighbors.com.

[95] Yoosoo Oh, Albrecht Schmidt, and Woontack Woo. Designing, developing, and evaluating context-aware systems. In *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*, MUE '07, pages 1158–1163, Washington, DC, USA, 2007. IEEE Computer Society.

[96] Ontologies. http://semanticweb.org/wiki/ontology.

[97] OpenCyc. http://opencyc.org/.

[98] PHP. http://www.php.net/.

[99] Davy Preuveneers, Jan Van Den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, E Berbers, Karin Coninx, and Koen De Bosschere. Towards an extensible context ontology for ambient intelligence. In *In: Proceedings of the Second European Symposium on Ambient Intelligence*, pages 148–159. Springer-Verlag, 2004.

[100] Protege-OWL. http://protege.stanford.edu/overview/protege-owl.html.

[101] Chris P. Rainsford, Michael D. Goldsmith, and Paul Prekop. Tiki: A trigger-based infrastructure for knowledge and information sharing. In *In Proceedings of the IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises: Collaborative Business Ecosystems and Virtual Enterprises (PRO-VE '02)*, pages 329–336, 2002.

[102] A. Ranganathan and R.H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 143–161. Springer-Verlag New York, Inc., 2003.

[103] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M. Dennis Mickunas. Middlewhere: a middleware for location awareness in ubiquitous computing applications. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, pages 397–416, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[104] Anand Ranganathan, Roy H. Campbell, Arathi Ravi, and Anupama Mahajan. Conchat: A context-aware chat program. *IEEE Pervasive Computing*, 1(3):51–57, July 2002.

[105] Anand Ranganathan, Robert E McGrath, Roy H. Campbell, and M. Dennis Mickunas. Use of ontologies in a pervasive computing environment. *Knowl. Eng. Rev.*, 18(3):209–220, September 2003.

[106] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[107] Roland Reichle, Michael Wagner, Mohammad Ullah Khan, Kurt Geihs, Jorge Lorenzo, Massimo Valla, Cristina Fra, Nearchos Paspallis, and George A. Papadopoulos. A comprehensive context modeling framework for pervasive computing systems. In *Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, DAIS'08, pages 281–295, Berlin, Heidelberg, 2008. Springer-Verlag.

[108] M.J.B. Robshaw. Md2, md4, md5, sha and other hash functions. Technical report, RSA Labs vol. 4.0, Tech . Rep. TR-101, 1995.

[109] J.K. Rosenblatt. Damn: A distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):339–360, 1997.

[110] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[111] D. Salber, A.K. Dey, and G.D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 434–441. ACM, 1999.

[112] S. Saroiu, K.P. Gummadi, and S.D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia systems*, 9(2):170–184, 2003.

[113] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, 2001.

[114] A. H. Sayed, A. Tarighat, and N. Khajehnouri. Network-based wireless location: Challenges faced in developing techniques for accurate wireless location information. *IEEE Signal Processing Magazine*, 22:24–40, 2004.

[115] A.H. Sayed, A. Tarighat, and N. Khajehnouri. Network-based wireless location: Challenges faced in developing techniques for accurate wireless location information. *IEEE Signal Processing Magazine*, 22:24 40, July 2005.

[116] Wendy A. Schafer, John M. Carroll, Steven R. Haynes, and Stephen Abrams. Emergency management planning as collaborative community work. *Journal of Homeland Security and Emergency Management*, 5(10).

[117] B. N. Schilit and M. M. Theimer. Disseminating active map information to mobile hosts. *Netwrk. Mag. of Global Internetwkg.*, 8(5):22–32, September 1994.

[118] A. Schmidt, M. Beigl, and H.W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.

[119] Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893 – 901, 1999.

[120] Hedda R. Schmidtke. Granularity as a parameter of context. In *Proceedings of the 5th international conference on Modeling and Using Context*, CONTEXT'05, pages 450–463, Berlin, Heidelberg, 2005. Springer-Verlag.

[121] T. Selkar and W. Burlesson. Context-aware design and interaction in computer systems. *IBM Syst. J.*, 39(3-4):880–891, July 2000.

[122] Adobe Flash Media Server. http://www.adobe.com/products/adobe-media-server-family.html.

[123] Apache HTTP Server. http://httpd.apache.org/.

[124] Ben Shneiderman and Jennifer Preece. 911.gov. *Science*, 315, 2007.

[125] John Soldatos, Ippokratis Pandis, Kostas Stamatis, Lazaros Polymenakos, and James L. Crowley. Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. *Comput. Commun.*, 30(3):577–591, February 2007.

[126] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[127] T. Strang. Towards autonomous context aware services for smart mobile devices. In *Lecture Notes in Computer Science (LNCS)*, pages 279–293. Springer-Verlag, 2003.

[128] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on advanced context modelling, reasoning and management as part of UbiComp*, pages 1–8. Citeseer, 2004.

[129] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France*, pages 236–247. Springer Verlag, 2003.

[130] Suggested Upper Merged Ontology (SUMO). http://www.ontologyportal.org/.

[131] MacEachren A.M. Tomaszewski B. Geo-historical context support for information foraging and sensemaking: Conceptual model, implementation, and assessment. In *IEEE Symposium on Visual Analytics Science and Technology (VAST 2010)*, pages 139–146, Oct 2010.

[132] Prefuse Information Visualization Toolkit. http://prefuse.org/.

[133] Twitter. www.twitter.com.

[134] twitter4j. - a java library for twitter api http://twitter4j.org/en/index.html.

[135] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–155, 1996.

[136] A. Varshavsky, A. LaMarca, J. Hightower, and E. de Lara. The skyloc floor localization system. In *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pages 125–134. IEEE, 2007.

[137] E. Vysniauskas and L. Nemuraite. Transforming ontology representation from owl to relational database. *Information Technology and Control*, 35(3A):333–343, 2006.

[138] X.H. Wang, D.Q. Zhang, T. Gu, and H.K. Pung. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. IEEE, 2004.

[139] Schafer Wendy, Carroll John, Haynes Steven, and Abrams Stephen. Emergency management planning as collaborative community work. *Journal of Homeland Security and Emergency Management*, 5:10, 2005.

[140] Connie White, Linda Plotnick, Jane Kushma, Starr Roxanne Hiltz, and Murray Turoff. An online social network for emergency management. *International Journal of Emergency Management*, 6(3-4):269–382, 2009.

[141] Philip Fei Wu, Yan Qu, Jenny Preece, Ken Fleischmann, Jennifer Golbeck, Paul Jaeger, and Ben Shneiderman. Community response grid (crg) for a university campus: Design requirements and implications. In *Proceedings of the 5th International Conference on Information Systems for Crisis Response and Management*, 2008.

[142] Jason I. Hong Kevin Wang Leila Takayama Xiaodong Jiang, Nicholas Y. Chen and James A. Landay. Siren: Context-aware computing for firefighting. *Pervasive Computing*, pages 87–105, 2004.

[143] S.J.H. Yang, APM Huang, R. Chen, S.S. Tseng, and Y.S. Shen. Context model and context acquisition for ubiquitous content access in ulearning environments. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, volume 2, pages 78–83. IEEE, 2006.

[144] Stephen S. Yau and Junwei Liu. Hierarchical situation modeling and reasoning for pervasive computing. In *Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, SEUS-WCCIA '06, pages 5–10, Washington, DC, USA, 2006. IEEE Computer Society.

[145] H. Ye, T. Gu, X. Zhu, J. Xu, X. Tao, J. Lu, and N. Jin. Ftrack: Infrastructure-free floor localization via mobile phone sensing. In *Pervasive Computing and Communications, 2012 Tenth Annual IEEE International Conference on*. IEEE, 2012.

[146] J Ye, Coyle L, Dobson S, and Nixon P. Ontology-based models in pervasive computing systems. *The Knowledge Engineering Review*, 22:315–347, 2007.

[147] M.A. Youssef, A. Agrawala, and A. Udaya Shankar. Wlan location determination via clustering and probability distributions. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 143–150. IEEE, 2003.

[148] J.M. Zagami, S.A. Parl, J.J. Bussgang, and K.D. Melillo. Providing universal location services using a wireless e911 location network. *IEEE Communications Magazine*, 36:66–71, 1998.

[149] B. Y. Zhao, Huang Ling, J. Stribling, S. C. Rhea, A.D. Joseph, and J. D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53,, Jan 2004.

[150] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, CA, USA, 2001.

[151] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable owl ontology storage and inference system. *The Semantic Web–ASWC 2006*, pages 429–443, 2006.

[152] Andreas. Zimmermann, Andreas. Lorenz, and Reinhard. Oppermann. An operational definition of context. *Lecture Notes on Artificial Intelligence, Springer-Verlag*, 4635:558–571, 2007.