An ontological context model for representing a situation and the design of an intelligent context-aware middleware

Preeti Bhargava*
Department of Computer Science
University of Maryland, College Park
prbharga@cs.umd.edu

Shivsubramani Krishnamoorthy*
Department of Computer Science
University of Maryland, College Park
shiv@cs.umd.edu

Ashok Agrawala
Department of Computer Science
University of Maryland, College Park
agrawala@cs.umd.edu

ABSTRACT
A major challenge of context models is to balance simplicity, generality, usability and extensibility. It is also important that the model be practical and implementable. In pursuit of this goal, this paper proposes a context model, Rover Context Model (RoCoM), structured around four primitives that can be used to represent and model any situation and activity: entities, events, relationships, and activities. It introduces the notion of templates of context for each primitive and describes, albeit briefly, the RoCoM Ontology (RoCoMO). It also describes the design and architecture of an abstract, generic and intelligent context-aware middleware called Rover II. We propose this framework as a solution to address the context problem as a whole, and be usable in many domains. We also illustrate its application with the aid of a context-aware public safety application that is deployed in the UMD campus.

Author Keywords
Context-Aware Computing, Context Modeling and Representation, Context-Aware Middleware, Situation Modeling

ACM Classification Keywords
H.5 Information interfaces and presentation; I.2.4 Knowledge Representation Formalisms and Methods

General Terms
Design

INTRODUCTION
Many computer query outcomes are significantly improved by the use of context information. Many effective decisions and actions require that the context be explicitly considered. 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 between applications. Such a system can store, intelligently process, reason over, merge and learn from contextual information available from heterogeneous sources without coordinating each pair of applications. It will also shift the burden of acquiring, storing, reasoning and other processing of context away from mobile devices with limited resources and computing power and on to a central system which is resource rich and has ample computing power. Moreover, 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.

To be effective and efficient aids to decision making, context-aware systems should be universally 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 simple, general, flexible and expressible. We present such a context model in this paper, which focuses on simple, extensible and general representations of both context and its relationship to different elements of the system. It is abstract enough to manage various dimensions of context such as location, time, and user profile. It is general enough to allow additions of context categories without redesign, while remaining usable across many applications.

We also present the design of a context-aware middleware, Rover II, based on this context model which will serve 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

*These authors contributed equally to the work.

UbiComp '12, Sep 5-Sep 8, 2012, Pittsburgh, USA.
ACM 978-1-4503-1224-0/12/09.

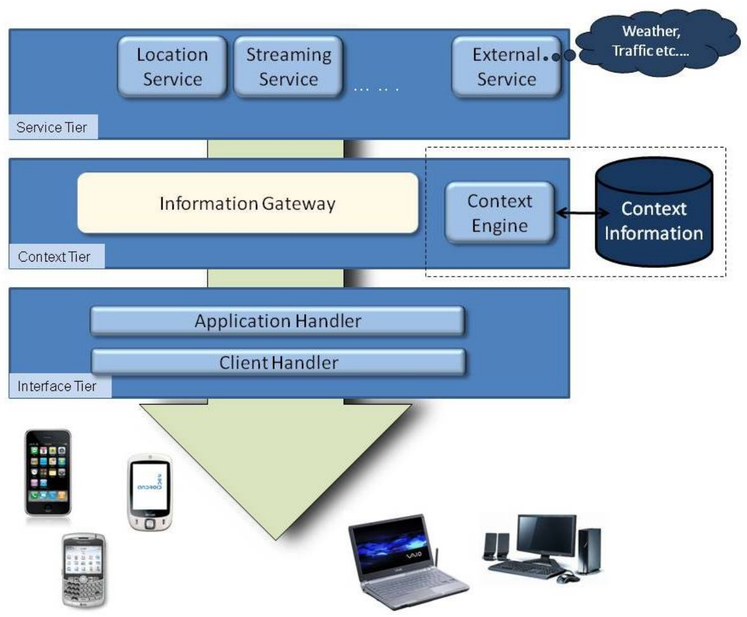


Figure 1. Rover II ecosystem architecture

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.

Thus, our objectives in this paper are to:

- Explain context, context models, context-aware systems and middleware,
- Describe our context model - Rover Context Model (RoCoM) and compare it with existing models,
- Briefly describe the RoCoM Ontology (RoCoMO),
- Present the design of Rover II [1], the context-aware middleware and integration platform that we are developing. This platform is being built on top of Rover 1.0 [2] which relied on 8-tuple context entries store and represent context, and
- Illustrate the practical aspects of RoCoM and Rover II through M-Urgency [14], a context-aware public safety application

Figure 1 shows a high-level design of the Rover II ecosystem architecture. An ecosystem here is a logical view of how different entities interact with the context-aware system. The 3 tiers in the architecture are:

[1]Note that the development of Rover II is an ongoing effort and we present the current state of the system in this paper.

1. The interface tier enables client applications to communicate with Rover. Client applications can run on desktops, tablets, or mobile phones.
2. The core tier or context tier where storage, processing and propagation of contextual information takes place.
3. The service tier which provides additional services such as streaming services, location based services, third party web services or external databases to supplement the contextual information.

CONTEXT, CONTEXT MODELS AND CONTEXT-AWARE SYSTEMS : AN OVERVIEW

Context

The most quoted definition of context, amongst all the definitions available, is by Dey [7],

> 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 general notion of context is vague, so it is useful to consider concrete examples. Location and identity can both be considered as properties of an entity and hence its context. A motor disability is also part of the person's context. This disability can play an important role in situations with tasks requiring movement. In the real world, context is essentially a property or an attribute of any of the elements of the context-aware system. This includes an entity's ability to carry out an activity, or the role that the entity plays in an activity.

Context can have a type, a description and a value that can be discrete or nominal. It can also have a hierarchical structure.

Context Models

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*. Thus, 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 since, at design time, we do not know what information will be relevant to all applications. New applications may require new types of contextual information. Bolchini et al, in their survey of context models [3], define the essence of the context design problem as one in which "the modeling of elements, that affect the knowledge/services/actions that have to be made available to the user at run-time, when a context becomes active, is carried out according to the application domain."

To this end, they analyzed the existing context models and developed a framework to evaluate and assess them. They specified certain common features that characterize the existing context models. These features are categorized as:

- Context dimensions - This category includes the set of context dimensions managed by the model. Examples are location, time, context history, subject, and user profile
- Representation features - This category includes the general characteristics of the model. Examples are flexibility, context granularity, and context constraints.
- Context management and usage - This category focuses on the context itself and its management, use and exploitation. Examples are context construction, reasoning, context information and quality monitoring, automatic learning features, multi context modeling, contextual ambiguity and incompleteness management.

An *ideal* context model is one which incorporates the most useful of these features and characteristics. As evident by their survey, it will 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 will be versatile enough to have a rich set of representation features such as flexibility, context granularity and constraints. It will 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 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 and its underlying ontology - RoCoMO.

Context-aware Systems and Middleware

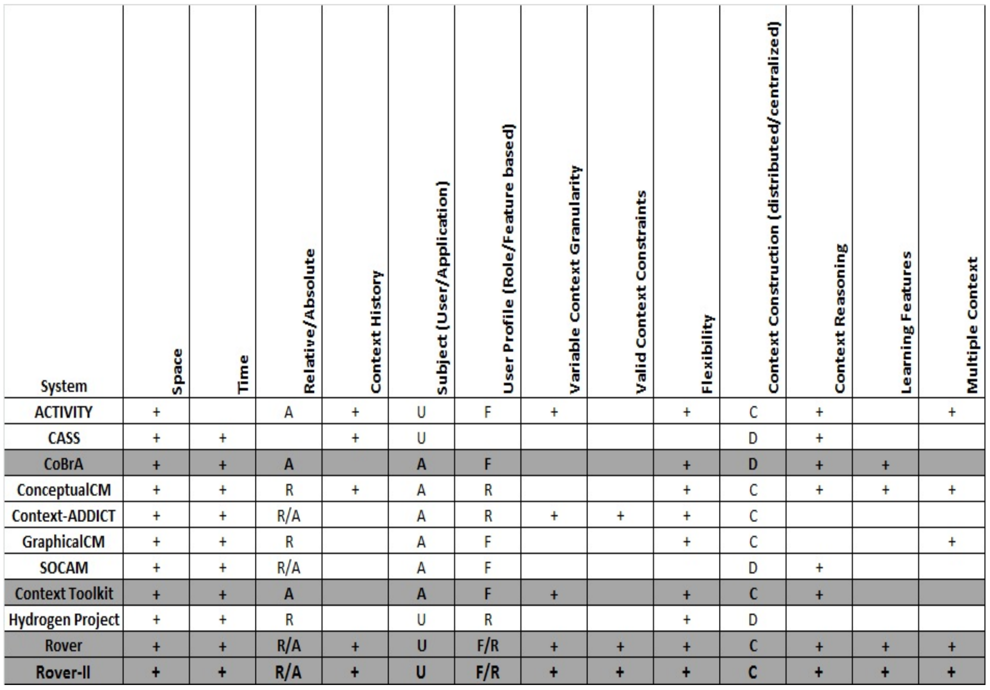
Dey et al [8] state, "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." They focus on adaptiveness of the application rather than change in its behavior. They also specify certain features that a context-aware 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.

Middleware is considered an indispensable component of context-aware environments. Ranganathan et al [17] 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 applications.
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)

Related Work in Context Modeling, Context-aware Systems and Middleware

As part of their survey, Bolchini et al [3] also identify five categories of models and context-aware systems based on the main focus of the model, the representation of context



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. Context Models, Context-aware Systems and Middleware - A Comparison

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. 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 models 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. Context definition is distributed and context quality monitoring and ambiguity are the key issues.
- 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.

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, context-aware systems and middleware, which cover all the categories identified by Bolchini et al in [3], include ACTIVITY [12], an Activity Theory based model which encapsulates context as a set of elements that influence users' intentions while doing some activities. The model appears to be in its nascent stage and the implementation details are not very lucid. In [20], Yang et al, describe an ontology based context model called U-Learn which is specific to learning environments.

Figure 2 presents a more detailed comparison of Rover and Rover II with a few other context models, context-aware systems and middleware. It is an extension of the analysis done by Bolchini et al [3]. Here, we have highlighted the systems for which we found concrete implementation details in the literature. CASS [9] is a centralized server-based context management framework with distributed sensors. It consists of a sensor listener and a rule engine. CoBrA [6] 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 [4], 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 [10], a theoretical context model, focuses on context quality and its temporal aspects. Context Toolkit [18] 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 [11] follows a completely decentralized architecture with two devices exchanging contextual information as soon as they discover that they are in close proximity.

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 and Rover II. 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 [17] - It supports integration of different mobile applications and collection and storage of information 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.

Moreover, Ye et al [21] propose the following coarse-grained criteria to assess ontologies in pervasive computing 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 [6] and CoBrA-Ont [5] are the only ontologies that come close to satisfying these requirements. However, for our purposes of modeling every situation in terms of the four primitives of RoCoM and setting a goal for the context-aware system to address the shortcomings of other context models, neither of these ontologies is adequate. In addition, the problem with some of the general and exhaustive ontologies like OpenCyc [15] 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 for a finite number of people to enumerate all the possible concepts, and the relationships between them, that could be used in a practical context-aware system. Hence, in our opinion, it is best to develop a generic base ontology and make it extensible. RoCoMO is currently being developed in OWL2 DL [1] that is expressive, versatile and supports reasoning. It also satisfies the assessment criteria mentioned above 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.

ROCOM AND ROVER II ARCHITECTURE

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.
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 necessary 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.
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.
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.

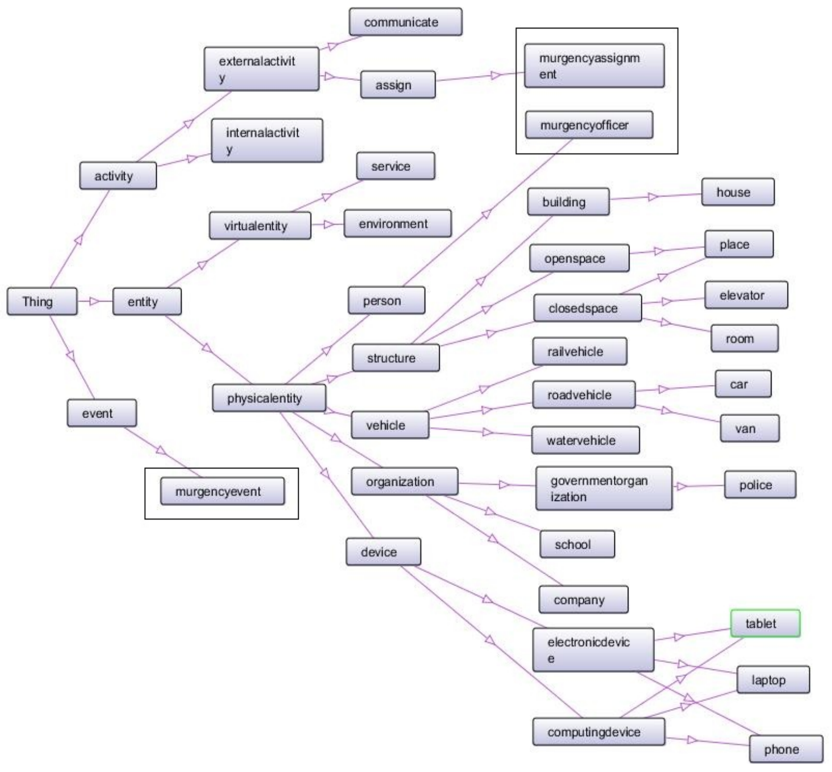


Figure 3. Partial description of the RoCoM Ontology

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 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 [16]. 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 3 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. The complete details of RoCoMO are beyond the scope of this paper. For a detailed explanation of how a situation can be visualized, please refer to [13].

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 [19]. 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 etc. 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.

Rover II architecture

Rover [2] is a context-aware middleware and integration platform that caters to the development of context-aware mobile applications. Our framework uses a paradigm for handling context information that includes user specific context combined with common context. The next version – Rover II [13] is in its early stages of development. The context model mentioned in the previous section forms the crux of Rover II.

We have designed the architecture of Rover II keeping three essential features of context-aware systems in mind, namely customization, adaptability and interactivity[2] as well as the requirements of middleware in ubiquitous environments [17]. The system is meant to be used as an auxiliary decision making support aid to a human entity. Thus, the final decision of taking any action, based on the contextual information provided by Rover II, rests with the human decision maker.

Figure 4 depicts the detailed design of the Rover II Context Tier. 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 Rover. 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 from one module to another.
   (b) The Relevant Context module determines the relevant context for each primitive, based on predefined primitive templates stored in the Template Store.
   (c) Primitives Templates module is used to fetch the templates for any of the primitives from the Template Store.
   (d) The Reasoning Engine module will perform inference and reasoning about complex situations using rule based reasoning, probabilistic reasoning and first order/predicate/temporal/spatial logic.
   (e) The Learning Engine will be capable of learning about application and user behavior from context history using learning and data mining techniques like supervised or unsupervised learning, reinforcement learning, Naive Bayes Classifiers, Hidden Markov Models to model causal relationships etc.
   (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 Controller.
   (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 predefined template for each primitive.

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 the template for the primitive (activity or event) from the Template store through the Primitives Templates module, 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.

ILLUSTRATIVE APPLICATION - M-URGENCY

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, that can be forwarded to police squad car(s) or other first responders nearest to the location of emergency. The M-Urgency system comprises three applications: the *caller* application from the emergency

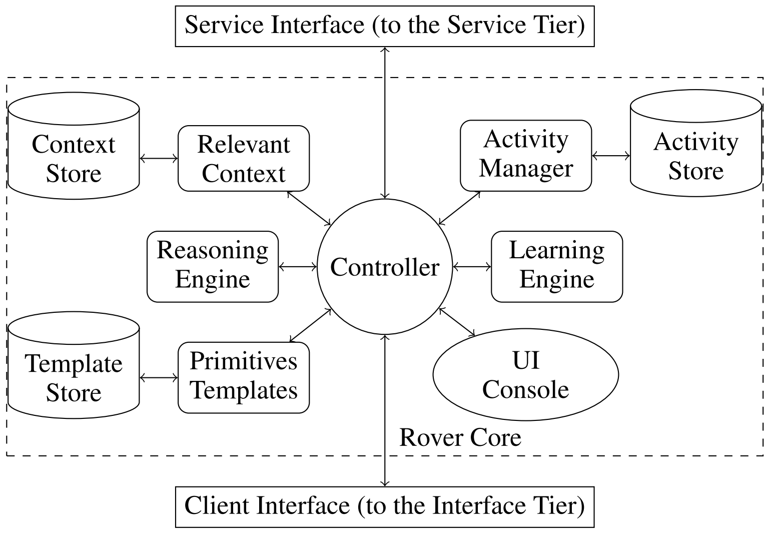


Figure 4. Rover II Context Tier

site, the *emergency dispatcher* application and the *emergency responder* application.

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.

The basic functionalities of the application have been developed. We are in the process of deploying a pilot system at the University of Maryland Police Department.

Figure 5 illustrates the involved entities, their associated context, and their relationships in a car accident situation. This situation can be represented in terms of the primitives of RoCoM as the following:

Event
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 catalyzes the context-aware system and sets the goal as informing and seeking help of the emergency personnel.

Entities
A few of the entities involved in the aforementioned event, along with some of their relevant contextual information are:

- *Caller* - age, location, medical history(of the victim) etc.
- *Dispatcher* - availability, serving jurisdiction etc.
- *Police or Medical responder* - availability, experience or 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.

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

- *Call the police dispatcher*: the status of the dispatcher becomes busy or unavailable for another call and the context 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 and dispatcher becomes available for the next call
- *Vehicle is towed away*: the context of the place changes from traffic block to slowly moving traffic and the context of the medical personnel changes to "attending a patient"

Relationships
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).

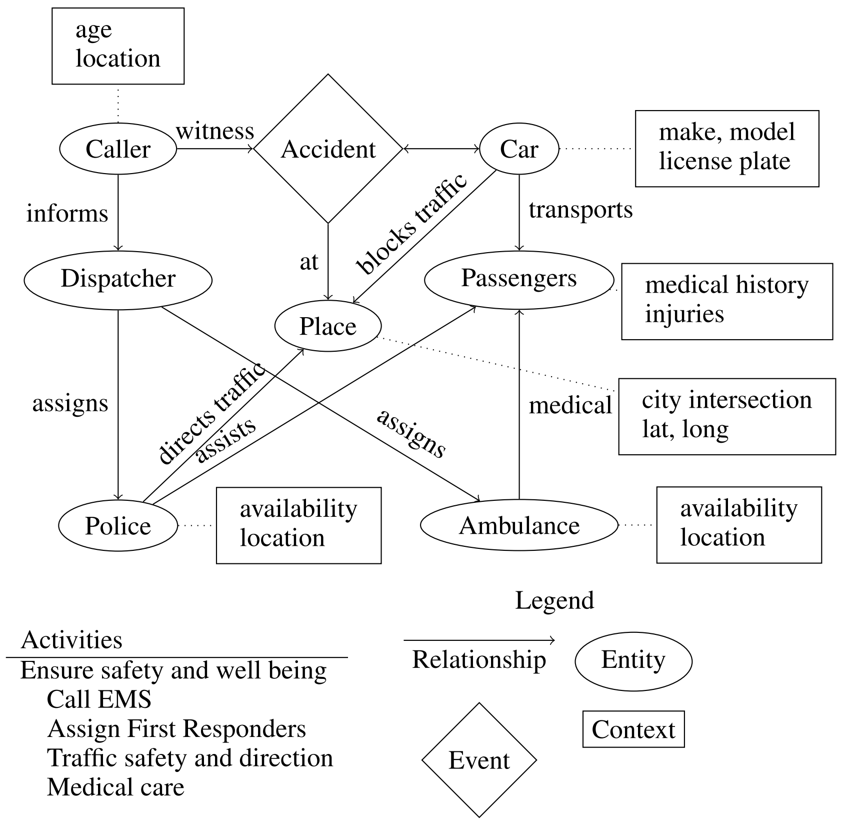


Figure 5. Representing a car accident situation using RoCoM

- *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

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. [2] Here, we provide only an instance of what happens in Rover when a connection is made from an M-Urgency caller.

Consider the event of a road accident, the caller application establishes a connection with the Rover system. 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 from the caller to the police department.

[2]M-Urgency is only one of the many applications that can interact with Rover II and exchange contextual information with other applications. Rover II can also coalesce information from other applications and third party services and reason and learn from it. The only criterion is that the applications provide the contextual information to Rover using RoCoMO templates

CONCLUSION AND FUTURE WORK

We presented in the paper, a context model RoCoM that we have adopted in the context-aware middleware and integration platform called Rover II, both being developed by us. RoCoM represents both context and its relationship to primitives in a simple, extensible and generic context ontology. It is general enough to allow additions of context categories without redesign, while remaining usable across many applications. The RoCoM Ontology that was described briefly in the paper is currently being designed and developed. We are also in the process of developing a GUI based extension or plugin to allow other users to use the ontology and extend it. Our context-aware middleware, Rover II, is also in its initial stages of development. It is abstract and generic, and can be adapted to any domain. It maintains a predefined structure for each of the primitives, termed as a template, which helps determine the contextual information for each primitive. It also incorporates a wide variety of functionalities and is usable in multiple ways. In addition, we illustrated the model and the system with the aid of a context-aware public safety application developed by us.

In the future, we plan to integrate several applications with Rover, allowing it to reason over information from different sources as well as learn from context history using several reasoning, inference and learning mechanisms. Also, contextual information from applications and sensors can always have a certain amount of uncertainty associated with it and thus Rover II should be able to handle that. Other aspects to consider in the design of Rover II are: security and authentication, logging and data distribution.

REFERENCES

1. Owl 2 web ontology language document overview http://www.w3.org/tr/owl2-overview/.
2. C. B. Almazan. *ROVER: Architectural Support for Exposing and Using Context*. PhD thesis, University of Maryland, College Park, 2010.
3. C. Bolchini, C. Curino, E. Quintarelli, F. Schreiber, and L. Tanca. A data-oriented survey of context models. *ACM SIGMOD Record*, 36(4):19–26, 2007.
4. C. Bolchini, C. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. Context-addict. Technical report, Dip. Elettronica e Informazione, Politecnico di Milano, 2006.
5. H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003.
6. H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. the First Annual International Conference on*, pages 258–267. Ieee, 2004.
7. A. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.
8. A. Dey and G. 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.
9. P. Fahy and S. Clarke. Cass–a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*. Citeseer, 2004.
10. K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. *Pervasive Computing*, pages 79–117, 2002.
11. T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. 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.
12. 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.
13. S. Krishnamoorthy, P. Bhargava, M. Mah, and A. Agrawala. Representing and managing the context of a situation. *to appear in The Computer Journal*, 2012.
14. M-Urgency. http://m-urgency.umd.edu/.
15. OpenCyc. http://opencyc.org/.
16. Protege-OWL. http://protege.stanford.edu/overview/protege-owl.html.
17. A. Ranganathan and R. 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.
18. D. Salber, A. Dey, and G. 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.
19. X. Wang, D. Zhang, T. Gu, and H. 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.
20. S. Yang, A. Huang, R. Chen, S. Tseng, and Y. 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.
21. J. Ye, L. Coyle, S. Dobson, and P. Nixon. Ontology-based models in pervasive computing systems. *The Knowledge Engineering Review*, 22(4):315–347, 2007.