MIDDLEWARE INFRASTRUCTURE FOR CONTEXT-AWARE UBIQUITOUS COMPUTING SYSTEMS

by

CAMUS Team[1]

Technical Report

RTMM, Kyung Hee University

Feb. 2005

Approved by _____
Project Supervisor

_____

Date _____ 15th Feb

[1] Anjum Shehzad, Hung N. Q., Kim Anh P. M. , Maria Riaz, Saad Liaquat, Sungyoung Lee, & Young Koo Lee. (Note: All in alphabetical order)

RTMM, KYUNG HEE

UNIVERSITY

ABSTRACT

MIDDLEWARE
INFRASTRUCTURE FOR
CONTEXT-AWARE UBIQUITOUS
COMPUTING SYSTEMS

by CAMUS Team

Project Supervisor: Professor Sungyoung Lee
Department of Computer Engineering

Context-awareness is one of the fundamental requirements for achieving user-oriented ubiquity. Ubiquitous computing environment consists of diverse range of hardware and software entities, and is about the interactivity of such entities. Context-awareness is one of the fundamental requirements for achieving user-oriented ubiquity. In this report, we present the design and approach to a middleware solution that expedites context-awareness in a ubiquitous computing environment. Context-Aware Middleware for Ubiquitous computing Systems (CAMUS) envisions a comprehensive middleware solution that not only focuses on providing context composition at the software level but also facilitates dynamic features retrieval at the hardware level by masking the inherent heterogeneity of environment sensors. Complexity is handled by providing 'separation of concerns' between environment features extraction, contextual data composition and context interpretation. The entities and contextual information provided/utilized by them must have invariant meanings in order to have a common understanding among them. This results in sharing of information with common semantics, at different times and at different places and provides testability of formalized knowledge, emerging as a pool of consistent contextual knowledge available to different context-aware systems. Different reasoning mechanisms are incorporated in CAMUS as pluggable services. Ontology based formal context modeling using OWL is described. With a systematic approach, CAMUS is proved to be a flexible and reusable middleware framework.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION TO UBIQUITOUS COMPUTING

1. Introduction

1.1 Ubiquitous Computing Vision

The term "Ubiquitous Computing" was originally introduced by Mark Weiser [1] in the year 1991. In his fundamental article *"The Computer for the 21st Century"* [2], he elaborated about "the computer that disappears". For Weiser the way into the 21st century was obvious: Computer and Network technologies are getting smaller, cheaper, and more powerful. Therefore, more and more everyday artifacts are going to be equipped with a reasonable amount of computing power and, maybe even more important, are networked together into a virtually unique network of communicating "things that think". In the pure sense of the word, computing gets "ubiquitous", anywhere, any time. Computers in every thing that is calmly doing what we intend it to do, in a way that is non-obtrusive and user-friendly, in a sense that we do not have to focus our attention on the trivia of running an electronic system.

Research on Ubiquitous Computing (Ubicomp) is related to very many other disciplines from Robotics and Embedded Systems, Networking and Distributed Systems, to Artificial Intelligence and Psychology. Thus Ubiquitous computing is a very difficult integration of human factors, computer science, engineering, and social sciences.

1.2 Context-aware Computing

One goal of Context-aware Computing is to acquire and utilize information about the context of a device to provide services that are appropriate to the particular people, place, time, events, etc. For example, a cell phone will always vibrate and never beep in a concert, if the system can know the location of the cell phone and the concert schedule. Often, the term "Context-aware Computing" is used in a sense synonymously to Ubiquitous Computing. This is because almost every ubicomp application makes use of some kind of context. Ubicomp is mainly about building systems which are useful to users, which "...weave themselves into the fabric of everyday life until they are indistinguishable from it" [2].

For ubicomp systems, *Context* is essential. How can a system be helpful for a user? Users tend to move around often, doing new things, visiting new places, changing their mind suddenly, and changing their mood, too. Therefore, a helpful system seems to need some notion of *Context*.
*In the Human point of view***,** we have a quite intuitive understanding of *Context*. Here, *Context* is often referred to as "implicit situational understanding." In social interactions *Context* is of great importance. A gesture, a laugh, or the tone of sentences builds up the implicit "picture" of what is meant or what communication partner is thinking. The same words can have a completely different meaning in different contexts.
In Computer Science, Context is quite a familiar concept, be it within the discipline of Artificial Intelligence ("Thinking machines"), in Robotics ("Adaptive Systems"), in User Interface Design (like adaptive UIs or office assistants like the Microsoft Office assistant called "Clippy"), or basically any other discipline (to some extent). Especially, every discipline dealing with human users tries to take into account human behavior one way or the other, with the generated output loops back as part of the vector of input values.

From the variety of definitions commonly used by Ubicomp researchers we can imagine how difficult it is to find a common ground. Context definitions are far away from mathematical precision and a particular definition often strongly depends on an authors' subjectiveness:

- Schilit and Theimer [3]: "Context is location, identities of nearby people and objects, and changes to those objects."
- A. Dey and Abowd [4]: *"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."*
- Pascoe [5]: "Context is the subset of physical and conceptual states of interest to a particular entity."

So what is this leading to? Are those definitions helpful or misleading? In the sense of a functional definition they are only helpful as a general description of what to do. As an application designer they are only stating what they are doing anyway: trying to figure out what input is needed to produce the desired output. Hence, it is of topmost importance to have some common ground or a common "vocabulary" when talking about what *Context* is. We need some sort of *formal approach* towards handling and describing Context. Furthermore, in a software engineering sense, we need building-blocks for building context-aware applications in a structured way. The Context Toolkit [6] by A. Dey is a step into this direction and a good example for this

principle (fig.1.) The Toolkit includes building blocks called "Widgets", wrapper classes for Sensors which serve as a hardware-abstraction layer, "Aggregators", which concentrate multiple input values to a single output value, and "Interpreters", implementing some application logic and generating application dependant "higher-level" output based on the input given. They interpret the incoming data according to a pre-programmed scheme.

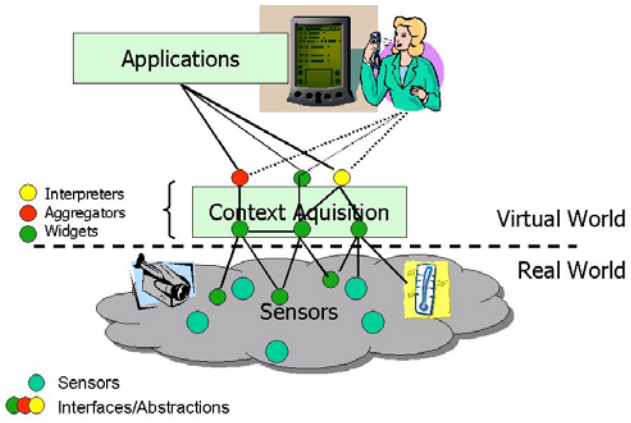


Figure 1: The Context Toolkit Core Components

With the Context Toolkit, the development of Context-aware applications basically consists of several distinguishable steps including
1) The real-world is sensed;
2) Context is detected, aggregated, "interpreted", and
3) Applications are custom-built to match the "context-detection" technology.

However, we believe that there is more tool-support necessary for software engineering and the design of Context-aware applications than provided today. We want to emphasize that the way applications are developed is very dependant on the underlying technology used, which we consider as bad practice in the long run. Research in the direction of decoupling applications from data acquisition seems to be important. This is detailed in the section 2, *Middleware for Context-aware Ubiquitous Computing Environments.*

1.3 Middleware and Its Evolution

The role of middleware is to ease the task of designing, programming and managing distributed applications by providing a simple, consistent and integrated distributed programming environment. Essentially, middleware is a distributed software layer, or 'platform' which abstracts over the complexity and

heterogeneity of the underlying distributed environment with its multitude of network technologies, machine architectures, operating systems and programming languages [7].

Different middleware platforms support different programming models. Perhaps the most popular model is O*bject-based Middleware* in which applications are structured into (potentially distributed) objects that interact via location transparent method invocation. Prime examples of this type of middleware are the OMG's CORBA [7] and Microsoft's Distributed COM [7]. Both of these platforms offer an *interface definition language* (IDL) which is used to abstract over the fact that objects can be implemented in any suitable programming language, an *object request broker* which is responsible for transparently directing method invocations to the appropriate target object, and a set of services (e.g. naming, time, transactions, replication etc.) which further enhance the distributed programming environment.

Sun's Jini [8] may also be categorized into object oriented middleware. In Jini, service objects register with a centralized lookup-service which plays the role of matchmaker between clients and services. After a client finds a service, all interactions are performed in a location-transparent manner and without the aid of the lookup-service. Typically, object-based systems assume that a connection between a client and a service object is long-lasting, and therefore these systems do not address the possibility of disconnection in ubiquitous and mobile computing.

Not all middleware are object based, however. Two other popular paradigms are *event based middleware* and *message oriented middleware* both of which mainly employ 'single shot' communications rather than the request-reply style communication found in object based middleware. *Event based middleware* is particularly suited to the construction of non-centralized distributed applications that must monitor and react to changes in their environment. Examples are process control, Internet news channels and stock tracking. Ubiquitous and Mobile Systems cover applications characterized by the heterogeneity of systems and devices, as well as the (spontaneous) patterns of interconnection. Due to the unpredictability of interaction schemes and the preferred use of asynchronous communication patterns, system design based on the notion of events seems to be superior to classical client/server interaction schemes. It is claimed that event based middleware has potentially better scaling properties for such applications than object based middleware. *Message oriented middleware*, on the other hand, is biased toward applications in which messages need to be persistently stored and queued. Message oriented middleware supports data exchange and request/reply style interaction by publishing messages and/or message queuing in a synchronous

and asynchronous (connectionless) manner. Workflow and messaging applications are good examples.

The issues of mobile devices (heterogeneity, scare resources), network connection (limited bandwidth, high error rate, higher cost, and frequently unpredictable disconnections), and mobility (dynamic changes of the environment parameters) of ubiquitous and mobile systems posed new challenges to the design of middleware systems. The middleware need to be designed to achieve optimal resource utilization, adaptivity and dynamic reconfiguration. *Reflective middleware* is concerned with applying techniques from the field of reflection in order to achieve flexibility and adaptability in middleware platforms. *Reflection* is the capability of a system to reason about and act upon itself. A reflective system contains a representation of its own behavior, amenable to examination and change, and which is causally connected to the behavior it describes. "Causally-connected" means that changes made to the system's self-representation are immediately reflected in its actual state and behavior, and vice-versa.

Tuple Space Middleware systems exploit the decoupled nature of tuple spaces for supporting disconnected operations in a natural manner. By default they offer an asynchronous interaction paradigm that appears to be more appropriate for dealing with intermittent connection of mobile devices, as is often the case when a server is not in reach or a mobile client requires to intentionally disconnecting and saving battery and bandwidth. By using a tuple-space approach, we can decouple the client and server components in time and space. In other words, they do not need to be connected at the same time and in the same place. However, since JavaSpaces [9] and TSpaces [10], the common Tuple Space Middleware, typically require at least 60Mbytes of RAM, they are not affordable by most handheld devices available on the market nowadays.

While traditional middleware for Distributed and Mobile environments do provide the basic mechanisms for different entities (or agents) to communicate with each other, they fall short in providing ways for agents to be context-aware. The ultimate goal of middleware for traditional computing environments is providing complete transparency of the underlying technology and the surrounding environments. Such an approach does not work for pervasive computing applications because being aware of the surrounding environment is the key to their effectiveness as we stated earlier. Moreover, Ubiquitous Computing environments feature a large number of autonomous agents. Various types of middleware (based on CORBA, Java RMI, SOAP, etc.) have been developed that enable communication interoperability between different entities, however, existing middleware have no facilities to ensure semantic interoperability between the different entities. Since agents are autonomous, it is

infeasible to expect all of them to attach the same semantics to different concepts on their own. This is especially true for context information, since different agents could have a different understanding of the current context and can use different terms and concepts to describe context. A middleware for context - awareness will address this problem by ensuring that there is no semantic gap between different agents when they exchange contextual information.

Chapter 2

MIDDLEWARE SOLUTIONS

2. Middleware Architectures for Context-Aware Computing

A lot of work has been done in the area of context -aware computing in the past years, among which much of them are only concerned with one or more aspects in an ad hoc manner. In addition, most of them rely on proprietary a protocol, thereby set a barrier to interoperability of different systems, and exclude developers from reusing existing components. We are going to give a summarization of the most prominent projects in this research field.

2.1 Context Toolkit [6]

The seminal work of Context Toolkit developed a set of abstractions for sensors data processing in order to facilitate reuse and make context aware applications easier to build. Context Toolkit separates the low-level sensing from high-level applications, and introduces a middleware layer whose functionalities are collecting raw sensor information, translating it to an application-understandable format, and disseminating it to interested applications. In Context Toolkit, a "context widget" is a wrapper component that provides unified access interface to context. To handle context query and event notification, each context widget has a state that is a set of attributes and a behavior that is a set of callback functions triggered by context changes. The widget obtains raw contextual information from sensors and passes them either to interpreters or to servers for aggregation. Interpreters and servers use simple HTTP protocol for communication and the XML as the language model.

The context is represented in the form of name value pairs which is not quite comprehensive and therefore lacks representation of all sorts of context. Also, no pluggable and reusable reasoning modules are present in the system.

2.2 Solar [19]

The Solar system architecture proposed a graph-based abstraction for context aggregation and dissemination. The abstraction models the contextual information sources as event publishers, and context aware applications as event subscribers. A number of event processing and routing mechanisms are designed to avoid redundant computation at context aggregation and interpretation nodes, and reduce data transmission in large scale context aware systems. Applications

use a subscription language to construct a logical event tree, based on event streams registered in a context -sensitive naming hierarchy.

But it too lacked proper context representation and support for inferring higher level context from lower level context/s in an appropriate way.

2.3 Context Fabric [20]

Context Fabric is actually an extension of the pioneering work of ParcTab system. It provides a distributed context-aware infrastructure with two fundamental built-in services, namely event service and query service, to support the acquisition and retrieval of context data. Context Fabric uses an entity-relation styled logical context data model to represent the information about four kinds of concepts: entities, attributes, relationships, and aggregates. Context about each kind of entities are assigned network-addressable logical storage units called infospaces that can be directly queried from network. Context data in Context Fabric is encoded using XML, and stored in local file systems. XPath is utilized as the query language for addressing parts of the XML tree structure.

Since context is encoded in the form of XML and it uses XPath, which may not be efficient for querying large scale data and may become bottleneck for the system.

2.4 Gaia [15]

The Gaia project developed at the University of Illinois is a distributed middleware infrastructure that provides support for context aware agents in smart spaces. Gaia adopts a predicate model of context data to enable agents to be developed that use first order logic rules to decide their behavior in different contexts.
It also proposed that different logic reasoning and machine learning techniques can be adopted to support context interference according to different application requirements. DAML encoded ontologies is used to ensure semantic interoperability between different agents, as well as between different ubiquitous computing environments. All agents are implemented on top of CORBA, and use CORBA Naming Service and the CORBA Trading Service for service discovery.

2.5 Motivation of Context-aware Middleware

Different approaches have been suggested for promoting context-awareness among agents. The Toolkit approach proposed by Anind Dey *et al* [6] provides a framework for the development and execution of sensor-based context-aware

applications with a number of reusable components. The toolkit supports rapid prototyping of certain types of context-aware applications. The other approach is developing an infrastructure or a middleware for context awareness. As said earlier, Context-aware Computing involves acquisition of contextual information, reasoning about context and modifying one's behavior based on the current context. A Context-aware Middleware would provide support for each of these tasks and greatly simplify the tasks of creating and maintaining context-aware systems. Middleware can help in continuous data acquisition, analysis, and pattern detection to infer higher level context, and let developers focus mainly on developing the applications' functionality rather than diverting their effort to hardware-specific issues. Besides, a middleware would be independent of hardware, operating system and programming language and provide uniform abstractions and reliable services for common operations. It would, thus, simplify the development of context-aware applications. It would also make it easy to incrementally deploy new sensors and context-aware agents in the environment. It would define a common model of context, which all agents can use in dealing with context. It would thus ensure that different agents in the environment have a common semantic understanding of contextual information. Finally, a middleware would also allow us to compose complex systems based on the interactions between a numbers of distributed context-aware agents.

2.6 Desired Elements of Context-aware Computing System

Context-aware computing relies on multiple independent and cooperative enabling technologies. Based on the extraction of literature review, we have identified a set of necessary functional elements that a context-aware system needs to support essential context aware mechanisms. As we argued in previous section, only a general middleware approach can combine independent functional elements in context aware computing, and melt them into a coherent system to provide a complete solution. These functional elements include:

- **Context Sensing**

In order to use context in services, there must be a mechanism to obtain the context data from diverse context sources. For example, the indoor location of a user can be obtained from an Infrared location sensor system, which detects the presence of a badge to conclude the location of the user wearing it. Context Sensing could be tightly-coupled with hardware sensors, while a component approach to decouple low-level sensing with high-level context usage can achieve reusable context sensing, thereby enabling the evolving of large scale context aware systems.

- **Context Model and Representation**

Context model forms the foundation for expressive context representation and high-level context interpretation [11, 12]. Existing context models vary in the expressiveness they support and the types of context they represent, while their common considerations should be how to capture general features such as properties of an entity and interrelation between contextual objects. On the context representation layer, 'raw material' of context is transformed into a machine-readable format based on the context model. This is actually an abstraction layer that acquires sensor data from context sources, and then annotates raw data with semantics that are structured around a set of contextual entities (e.g., 'user', 'location' and 'device'.) and the relations (e.g., 'locatedIn') that hold between them. A uniformed context model is needed to facilitate context interpretation, context sharing and semantic interoperability.

- **Context Repository**

Context information is obtained from an array of diverse information sources. A centralized context repository can provide a persistent storage for distributed context, guarantees integrity of context, and offers shared mechanisms, relieving context-aware services from overheads caused by querying from distributed sensors [13]. When context is represented based on shared context model, context repository provides a foundation to merge interrelated information and enables further data interpretation.

- **Context Query/Aggregator**

To explore general means of access to interrelated context spread across distributed context repositories, we need a high-level mechanism for context - aware services to issue queries without explicitly handling underlying data manipulation [14]. For example, a notification service for conference attendees require context like "find a list of researchers in this hall whose publications are in the same session with mine". The low-level operations of such complex context retrieval task should not be exposed to end users. Context query poses design issues such as context query language, event notification, and query optimization.

Similarly, Context Aggregator is responsible for satisfying certain context queries. Each context aggregation service performs a specific function. An example service can be detecting that user has awaken and performing certain actions. Based on required contextual information, it can either utilize the ontology reasoning module or context reasoning module or both. Upon detecting that certain context is composite one, it will retrieve meta-information from the repository about the specific context reasoning module providing the composite context. Once retrieved, it will invoke the corresponding reasoning module and return the result back to the application requesting the context.

•Context Reasoning/Inferring

Low-level information usually can not be directly understood and utilized by software services. Hence, there is a need to interpret low-level information and derive additional, high-level context. For example, a location based service wants to know the relative location with different levels of granularity (e.g., room-level, building-level, block-level) instead of sensor-driven position (the coordinates retrieved by the GPS system). In this case, we need to derive high-level location (e.g., 'which block is the user in?') from related contexts (e.g., GPS coordinates, the mapping between GPS coordinates and corresponding blocks). The context interpretation layer leverages reasoning/learning techniques to deduce high-level, implicit context needed by intelligent services from related low-level, explicit context.

For example, the rule-based reasoning engine can deduce user's current situation based on his location and environmental contexts. The inferred context might suggest that the user might be sleeping currently, since the time is 11 pm and he is staying at a dark, quiet 'Bedroom'. Another example of context interpretation could be machine learning based behavior prediction. The intelligent system could learn the pattern of user's actions from historical sequences of context data and then use this learned pattern to predict next event. For example, it could be predicted that once the user finished showering (turn off the electronic water heater) after 10:30 pm, he will check emails using the hand phone, and then go to bed after finishing reading them. Currently, context interpretation tasks are performed through various approaches including ad hoc interpretation, rule-based reasoning [15], and machine learning [16].

•Context Discovery/Delivery

In order for a context-aware service to use a certain kind of context, there is a need for context requestors to find the sources providing it. The aim of context discovery is to locate and access interested context sources in a self-configure manner. Issues of context discovery include service description, advertisement and event subscription [17].

The context delivery services perform the job of searching appropriate context aggregators and delivering them to the applications. These include registration, query and notification services. Context Aggregators register with the registration service to provide the information about the context they can deliver. Interested applications and agents query the registration service to find services of their interests. The registration service upon finding appropriate aggregator, returns the handler to the requested clients.

Each context aggregator specifies the context it provides, by utilizing the concepts defined in the ontology repository. This standard schema sharing allows the different kinds of entities to be described and utilize by registration service to find useful services needed by the applications, thus allowing a flexible mechanism for exchanging descriptive information of various entities. In our framework, this semantic matchmaking [44], [CD4] is based on querying the Racer [18] Server which allows subsumption and classification of different concepts defined in the ontology.

- **Context-aware Application/Service**

Finally, on the uppermost level (context utilization layer), context aware services utilize both low-level and high-level context to adjust their behaviors. The smart phone might then decide that the user probably does not want to answer any phone call when he is sleeping at home and forward those calls to the voice message box. It also could take account of predicted context and response by automatically adjusting the air-conditioning of the bedroom and downloading emails after the shower before sleep.

Chapter 3

CONTEXT-**A**WARE **M**IDDLEWARE FOR **U**BIQUITOUS COMPUTING **S**YSTEMS (CAMUS)

3. Context-Aware Middleware for Ubiquitous computing Systems

Based on the desired characteristics of a context-aware middleware for ubiquitous computing explained in previous chapter, here we outline briefly the main architecture of our middleware infrastructure. Further details are presented in the chapters following this one.

3.1 A Layered Model for Context-Aware Computing

From the functional elements defined in previous chapter, we proposed our Middleware Framework as a layered model depicted in figure 2.

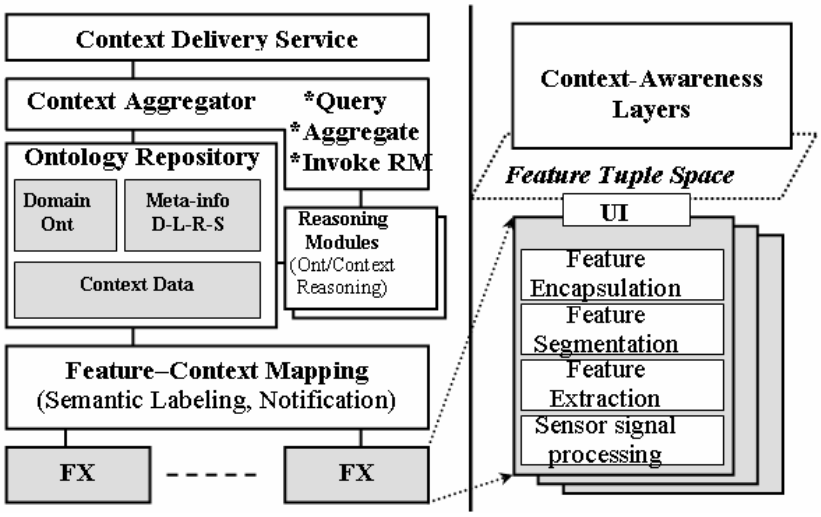


Figure 2: CAMUS Core Architecture. At Feature Extraction Agent, sensor signal will be preprocessed e.g. filtering, conversion, or contrast enhancement. Then features will be extracted, quantized/segmented, and encapsulated into feature tuples. Feature tuples are injected into Feature Tuple Space through Unification Interface (UI) for deducing context in upper layers (Context-Awareness Layers)

The lowest layer of CAMUS consists of *Feature Extraction Agents* (FX Agents). These sensing agents extract the most descriptive features for deducing contexts in upper layers. In order to have a more expressive representation of contextual

information, features are further quantized or segmented, resulting in a set of symbolic values that describe concepts from the real world. The quantized features are encapsulated in the form of Feature Tuple.

Feature - Context Mapping layer performs the mapping required to convert a given feature into elementary context using reasoning mechanisms and base on the meta-information saved in the ontology repository.

Ontology Repository provides the basic storage services in a scalable and reliable fashion and contains the domain ontology (concepts and properties), contextual information (including both elementary and composite contexts), and meta-information (D = devices, S = sensors access mechanisms, L = Feature - Context Labeling, as well as the meta-information about the input, output and capabilities of pluggable reasoning modules = R).

Reasoning Engine is a collection of various pluggable reasoning modules to handle the facts present in the repository as well as to produce composite contexts. *Reasoning Mechanisms* could be various kinds of logics e.g. Description Logic (DL), First Order Logic (FOL), Temporal and Spatial Logic, Fuzzy logic; or Machine Learning Mechanisms e.g. Bayesian networks and neural networks.

On top of CAMUS, *Context Aggregator* is responsible for satisfying certain context queries and providing context to interested applications through *Context Delivery Services*.

Chapter 4

UNIFIED SENSING FRAMEWORK

4. Unified Sensing Framework

4.1 Motivation

Acquiring the users input from the real world is one of the challenges in context-aware computing. However, the most interesting kinds of context are those that humans do not explicitly provide, and gathered from user's environment. The environment in general contains a diverse nomenclature of sensors having different access mechanisms, different sensory data and dissimilar representation schemes of such data. This diversity leads to potential problems and complexity in design and implementation and is a hurdle in the realization of context aware ubicomp. Thus a mechanism is required, which serves to extract information from the heterogeneous sensors and present to the upper layers for deducing contexts, in a standardized and unified manner.

4.2 Feature Extraction through Unification Interface

In CAMUS we provide a unification interface for each sensing agent (named Feature Extraction, FX, Agent), as depicted in figure 3. Unification Interface (UI) is aware of the definitions of the feature constructs based on the sensor types. This aids in deciding which sensor values are to be extracted and aggregated from sensor outputs. At this lowest layer in the CAMUS design, containers that hold access mechanism implementations (or wrappers) are also provided for individual sensors, to hide the communication details and data polling frequency of sensors from the above layers.

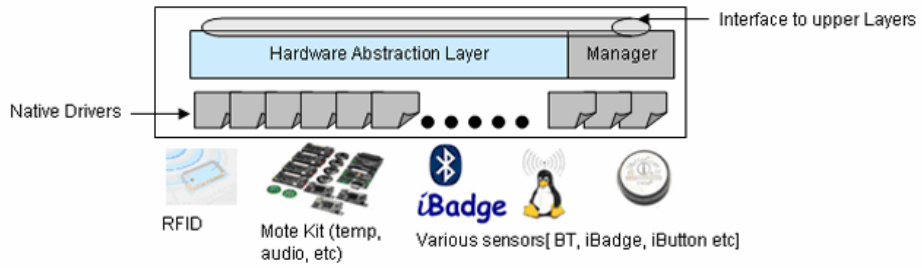


Figure 3: Unification Interface - Native drivers exist for individual sensors which are exposed to upper layers through a hardware abstraction layer, providing a standardized access to all sensors.

The features, formulated after sensor access modules retrieve raw sensor values from the sensors, should be as descriptive of the contexts they are attempting to model as possible. Following diagrams show the use case diagrams and activity diagrams of sensor access module / unification interface working to make features available.

If the features are discriminative enough, the recognition mechanisms would be simple and lightweight. In order to have a more expressive representation of context information, features are further quantized or segmented, resulting in a set of symbolic values that describe concepts from the real world. Fuzzy sets [21], [22] or crisp limits can be applied to quantize/segment the features. The probability (or confidence) associated with outputs of fuzzy quantization can be used as inputs of probability based context reasoning mechanisms [22], [23]. Numerous useful features might be generated by each sensor; and the feature representation must facilitate the identification of individual features uniquely as well as collection of features by the same source. This is achieved by allocating a unique feature identifier in conjunction with the sensor and type identifiers to each Feature Tuple (FT) in the space.

$$FT = \{Sesnor\_ID, Type\_ID, Feature\_ID, Feature\_Value, Probability, Timestamp\}$$

Table 1. Sample Feature Tuples. Sensor ID is assigned to each sensor board at development stage and mapped to corresponding location/device at deployment.

Attribute	Meaning
Sensor_ID	The unique ID of the sensor board, assigned at development stage & mapped to corresponding location/devices at deployment stage
Type_ID	Sensor type e.g. audio, temperature etc., for further distinction of context information sources, especially in case multiple sensor types on a single board.
FeatureID	Refers to feature category of each sensor type. It is actually the signature of corresponding feature extraction function in the library of the sensing device.
Feature Value	Symbolic (e.g. light intensity) or absolute numerical value (e.g. temperature absolute value) of the extracted and segmented sensor feature.
Probability	The uncertainty or confidence of the context information, could be 0/1 if the feature is segmented using thresholds, or within [0, 1] range if fuzzy sets are applied. This attribute is used to decide which feature values will be sent to backend system (probability > 0) and useful for probability based context recognition mechanisms e.g. Bayesian networks.
Timestamp	The absolute time when the feature is extracted. Used to keep consistency of context information, and sometimes useful for temporal reasoning mechanisms.

Table 2. Shows some sample features extracted from audio and video sensors (following the MPEG-7 specifications) with their symbolic values.

Sensor ID	Sensor Type	Feature ID	Value		Time Stamp
			Numeric Value	Quantized Value (Symbolic, Probability)	
3	1 (Audio)	1(Intensity)	x (dB)	1 (Silent, 0.9) 2 (Moderate, 0.1)	Xxxxx
7	2 (Video)	3(Motion Pattern)	NA	1 (Stable, 0.8) 2 (Regular, 0.2)	Xxxxx
		6(Posture)	NA	2 (Lying, 0.9)	Xxxxx
		7(Luminous Intensity)	y (cd)	1 (TotalDark, 0.2) 2 (Dark, 0.8)	Xxxxx

In CAMUS, Feature Tuple Space (FTS) is employed as underlying communication and storage mechanism. Features are stored directly as objects independent in space and time and decoupled from the generating processes; an important advantage in the ubiquitous environment in terms of interoperability and scalability. Various sub-modules for feature extraction and context formation dynamically interact in the middleware by mere flow of objects in and out of the FTS.

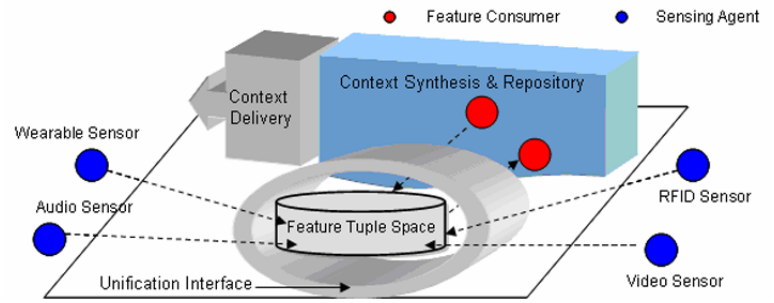


Figure 4: Sensory data is retrieved through the unification interfaces (native drivers) and stored into Feature Tuple Space in the form of features. The features are extracted by Context Synthesis layer to generate context.

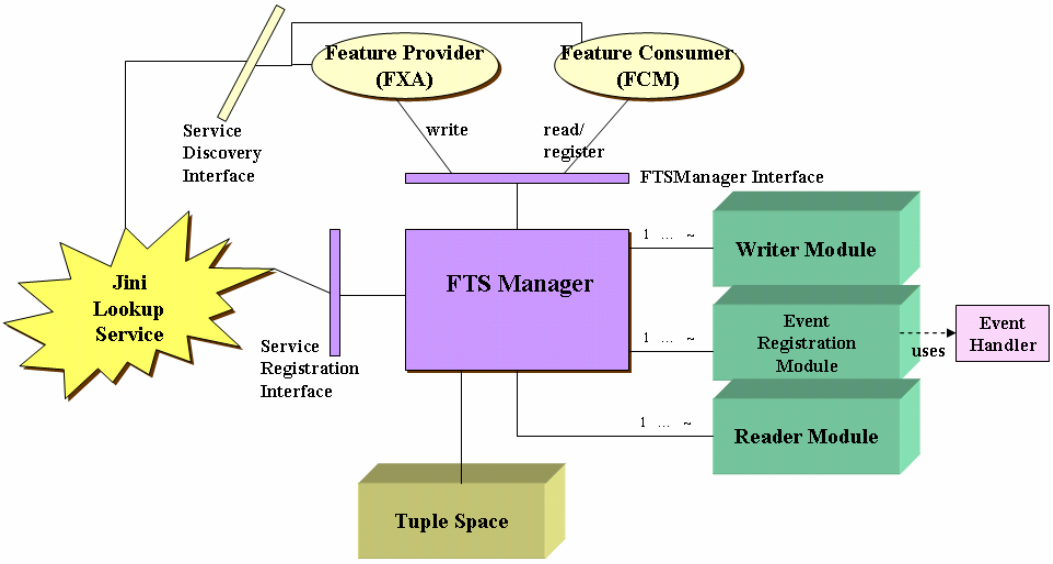


Figure 5: Feature Tuple Space Architecture

The notions of uniform feature tuple encapsulation and feature-context mapping meta-data empower CAMUS with the capability of incorporating nearly any sensing agents ranging from commodity off-the-shelf sensing devices e.g. microphone, camera; to autonomous smart sensing devices developed by other researchers.

4.3 Feature - Context Mapping: Utilizing the Strength of Reasoning

Upon the notification of feature change in the tuple space, this layer performs the mapping required to convert a given feature into context. The Feature - Context Mapping meta-information is saved in the ontology repository. Information such as user Id mapped to his name as well as his profile is saved in the ontology as a meta-information which enables this layer to do necessary mappings. If certain context is not present in the ontology repository and is requested by either context aggregator or reasoning module, it will register to the Feature Tuple Space for the feature, corresponding to that context. For example, the feature tuple in table 2

$$FT1 = \{3, 1, 1, 1, 0.9, xxxxx\}$$

would be mapped to corresponding context information

{Location.Bedroom, Environment.Sound.Intensity = Silent,
Probability = 0.9, TimeStamp = xxxxx}

Every time a new sensor type is added into the system, the feature ontology will be updated with the meta-information of all new features provided by that new sensor type. Based on this ontology, a feature can be converted into markup format, and vice-versa. For example, the feature tuple {6,1,1,null,00001001,1,20041210120000} can be mapped to corresponding context information:

```
<RFIDTag>
    <absValue>00001001</value>
    <probability>1</probability>
    <timestamp>20041210120000</timestamp>
<RFIDTag>
```

This layer includes one Mapping-Manager, and some client mapping service. Mapping-Manager provides the essential mapping from feature tuples to feature markup, and allows other services to register for some context events (notice: this is differ from the Feature Tuple Space which allows registration for feature tuple events). The client mapping services are domain-specific. Each client mapping service uses the Mapping-Manager registration service to register for some low-level contexts. When notified about the arrival of those concerned contexts, the client mapping service can utilize a reasoning engine to infer some high-level contexts. The strength of the reasoning engine decides the level and accuracy of mapped contexts.

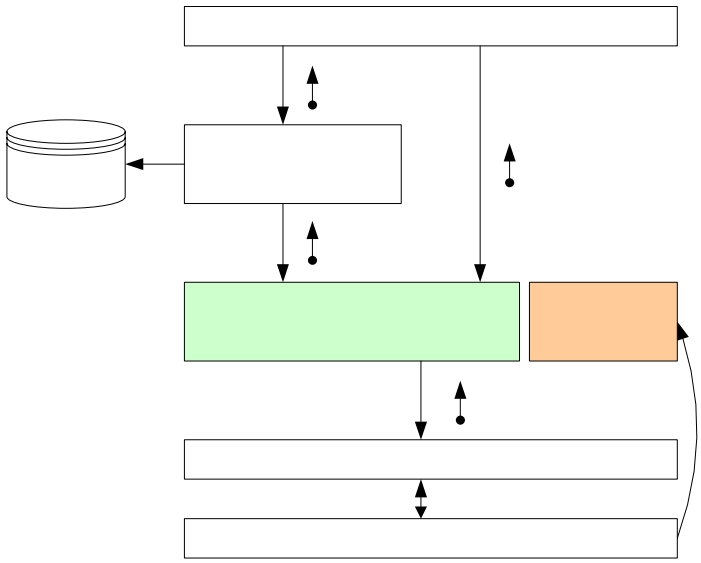


Figure 6: Feature – Context Mapping Layer Architecture

For example, a location mapping service can register for all kinds of features related to location such as features from RFID, IDTag, Berkeley mote, i-Badge … then the service bases on those feature contexts to infer the location of agents. However, using fuzzy logic to fuse the information from many sensors, or using Bayesian net to calculate the probability of location based on the probabilities of features, will provide more accurate location contexts than just inferring the location using description logic.

Hence CAMUS provides a flexible mechanism for mapping from feature data to context data. Domain developers can freely decide whether they will create new contexts using a reasoning engine or not, and which kind of reasoning engines will be used.

4.4 Discussion

The basic benefits of using this approach are two fold. Firstly sensor access is unified through hardware abstraction layer and standardization which results in easy access for upper layers and masking of sensor heterogeneity. Secondly, the use of features allows better description of different sorts of environment parameters than raw sensor values and features can be organized, stored and delivered in an efficient manner.

Context DB

4.5 Summary

Ubiquitous environments contain diverse range of sensors each utilizing its native access mechanisms and output formats. To provide a standardized representation of sensor data to the internal system modules and applications, CAMUS makes use of a unified access pattern to homogenize sensor interaction. As a first step, sensors are masked by a hardware abstraction layer in the unification interface. Access to sensors is still regulated by their native drivers and data formats. These native drivers and data formats are interfaced with the upper layers through unified and standardized interfaces, making access patterns similar for different types of sensors. Secondly, feature extraction agents use sensor data to formulate features and store these features in feature tuple spaces. These two mechanisms are used to provide the basis of a unified sensing framework.

5. Formal Context Modeling and Representation

5.1 Motivation

Context presentation is an important part of pervasive computing environments. Because context-aware applications must adapt to changing situations, they need a detailed model of users' activities and surroundings that lets them share users' perceptions of the real world. These entities may have different meanings associated with them in different pervasive environments. In order to have *invariant* meanings of these entities, when used at different times, in different situations, by different applications, they must be *formalized*, i.e. the context semantics should be formalized. Formalizing the context of an application has a number of clear advantages. First, it allows us to store the context for a long term since its meaning will remain same for future uses. The second advantage is for communicating the context universally with other systems. Third, formal meaning of the context leads to its testability of being a formalized knowledge. So, formalizing, the context model, helps to make a growing pool of well-tested context knowledge available to different context-aware systems.

Most context-aware systems to date mainly focus on the contents of context, neglecting the importance of interactivity among applications. Some have model the context as name-value pairs [6] and entity relation model, while others have used objects [23] to represent context, with fields containing state of context, methods to access, modify and/or register for notification changes to context. However, context reuse and sharing among wider application domains demand a need for formal context modeling enabling common understanding of the structured context. A survey on Context Modeling Schemes is provided in Appendix A.

•Formal Context Modeling using OWL Ontologies

Context entities are the concepts in a domain of discourse, and to provide formal meaning of these concepts, *ontologies* are used. Within the domain of knowledge representation, the term ontology refers to the formal, explicit description of concepts, which are often conceived as a set of entities, relations, instances, functions, and axioms, leading to shared and common understanding that can be communicated between people and application systems [25]. Formalizing domain

not only contains the vocabularies of concepts but relationships among them as well. W3C's OWL (web ontology language) [26] allows us to achieve this goal in two steps. First, it allows us to define concepts and their inter-relationships e.g. describing person, devices, location etc. Second, it allows us to define instance data pertaining to some specific time and space e.g. Bob is watching television in the living room. Traditionally, ontologies are only used to describe domains (as mentioned above) but in OWL, the horizon of ontology has been broadened to include instance data as well, effectively making the knowledge base.
OWL, a knowledge representation language, has explicit semantics associated with the knowledge, which provides reasoning capabilities used by intelligent systems and agents to infer useful contexts. As OWL is based on meta-modeling language (RDF [27]), it can be used to represent meta-information about sensors, we can also use OWL to represent access mechanisms to the sensors and associated policies.

The following example shows the context ontology that describes a user named Hung.

```
<User rdf:about="Hung">
<name>Hung</name>
<mbox>nqhung@oslab.khu.ac.kr</mbox>
<homepage rdf:resource="http://ucg.khu.ac.kr/~nqhung"/>
<office rdf:resource="#RoomB07"/>
<officePhone>2493</officePhone>
<mobilePhone>9999</mobilePhone>
<!—More properties not shown in thisexample—>
</User>
```

•Benefits of Semantic Web Ontology for Context Modeling
There are several potential advantages for developing context models based on Semantic Web Ontology. First, its e*xpressiveness*. Web ontology is modeled through an object-oriented approach, with expressive power entailed by their class/property constructors and axioms. Therefore it is more expressive than existing context models, allowing us to capture more features of various types of context.

Second, *Knowledge Sharing*; the use of context ontology enables computational entities such as agents and services in pervasive computing environments to have a common set of concepts about context while interacting with one another. By allowing pervasive computing entities to share a common understanding of context structure, OWL ontologies enable applications to interpret contexts based on their semantics.

Third, based on ontology, context -aware computing can exploit various existing *logic inference* mechanisms to deduce high-level, conceptual context from low-level, raw context, and to check and solve inconsistent context knowledge due to imperfect sensing. Because contexts described in ontologies have explicit semantic representations, Semantic Web tools such as Federated Query, Reasoning, and Knowledge Bases can support context interpretation. Incorporating these tools into smart spaces facilitates context management and interpretation.

Fourth, *Knowledge Reuse;* Ontologies' hierarchical structure lets developers reuse domain ontologies (for example, of people, devices, and activities) in describing contexts and build a practical context model or compose large-scale context ontology without starting from scratch. And lastly, it provides e*xtensibility*. Concepts in the context ontology are organized in form of taxonomies or hierarchies. Newly-defined concept can be easily added into the existing context ontology in a hierarchical manner.

5.2 CONTEL – Context Model in CAMUS

While context entities are conceptual entities, the information provided by them is called the contextual information. This contextual information has its own syntactic and semantic meanings. Some of the context entities are the producers of contextual information while others are consumers or both. Contextual information gathered from at least one sensor is called the '**elementary context**' while '**composite context**' is any combination of elementary contexts or elementary and composite contexts as shown in figure 7.

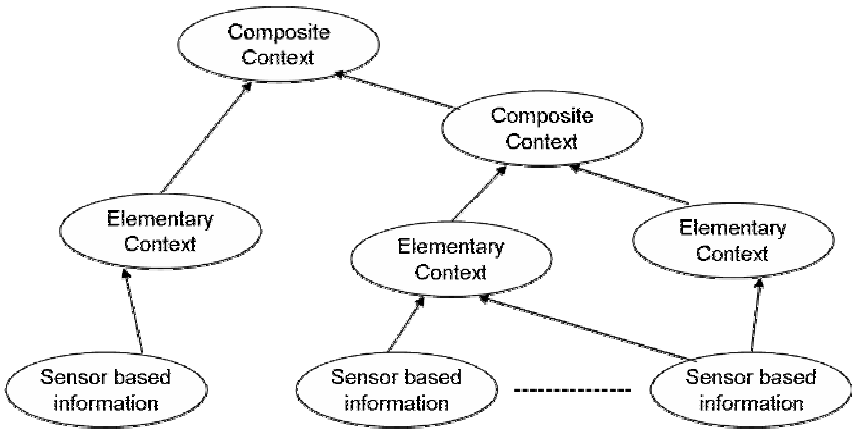


Figure 7: Contextual information hierarchy

- **Basic Model**

Diverse context entities ranging from various kinds of devices e.g. PDAs, mobile phones, ambient displays etc., running various applications, to various environment conditions e.g. sound intensity, light, temperature, traffic etc., are utilized by various kinds of agents e.g. software agents, persons, groups etc.
This variety leads us to categorize context entities, in our framework, mainly into agents, devices, environment, location and time. Location and time are kept separate from the other concepts to emphasize on the spatial and temporal aspects of the ubiquitous computing environment. These conceptual entities and their relationships are described in the ontology repository. Figure 8 shows the main context categories and few domain concepts of our context model, termed as, cont-el.

The shadowed ovals show, in figure 8, the main context categories while rectangles represent few of the concepts under the corresponding context category. Many new entities (devices, softwares etc.) may enter/leave the variable ubiquitous environment, but they can be made part of the system by adding their definitions at runtime into the ontology database and related to existing entities by various ontology language (OWL) constructs like subClassOf, disjointWith etc. So, representing context entities in the ontology brings all benefits of ontology world.

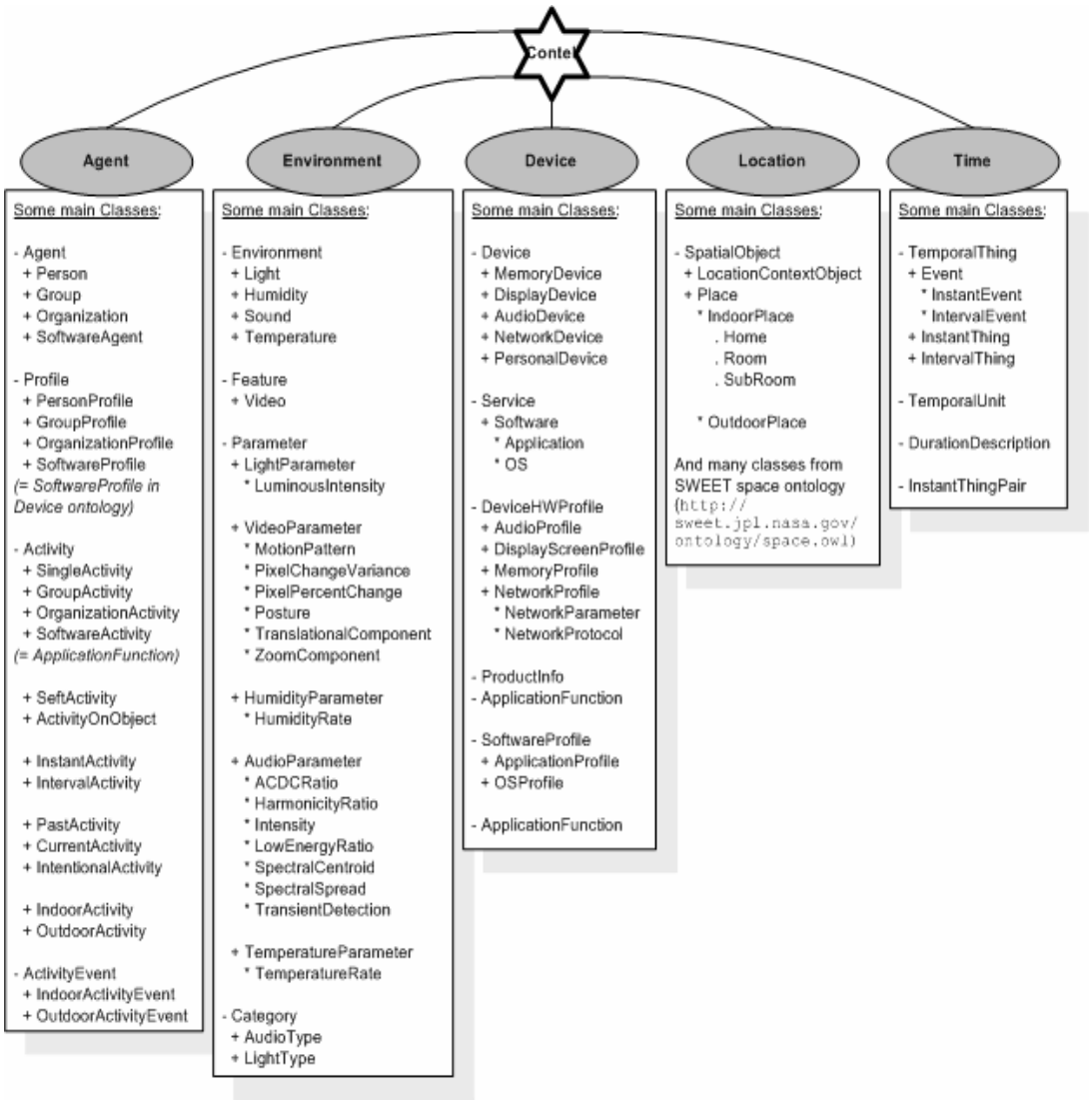


Figure 8: Expandable Cont-el Basic Categorization and Some Domain Concepts

- **Detailed Model**

Context entities and contextual information are described in the ontologies; facilitating various parts of the ubiquitous computing environment to interact with each other effectively. We have described ontologies for a home domain. The different ontologies made are based on basic categorization described above. In the following paragraphs, a part of different ontologies is described for the home domain.

For the entities related to Agent, we have top level concept called Agent. It has been further classified into SoftwareAgent, Person, Organization, and Group. Each Agent has property hasProfile associated with it whose range is AgentProfile. Also, an Agent isActorOf some Activity. Activity class, representing any Activity, can be classified based on the Actor of it e.g. SingleActivity (which has only one actor), GroupActivity (which has Group as its actor and can have many SinlgeActivity instances). An Activity having some object of action on which it is done called ActivityOnObject like CookingDinner, TurnOnLight, or WatchingTV etc., while SelftActivity has no object of action e.g. Sleeping, or Bathing. Activity itself is not related to time and location but whenever activity happens, it generates an ActivityEvent (subclass of Event and LocationContextObject), encapsulating both time and location information.

The Device ontology is based on FIPA device ontology specification [28]. Every Device has properties of hasHWProfile, hasOwner, hasService, and hasProductInfo. Device is further classified into AudioDevice, MemoryDevice, DisplayDevice, NetworkDevice. PDA is considered here as subClassOf AudioDevice, DisplayDevice, NetworkDevice, MemoryDevice and PersonalDevice. All different devices have associated device profiles e.g. DisplayDevice hasDisplayProfile of DisplayScreenProfile containing properties resolution, color, width, height and unit. The hasService property of Device class has Range of Service. Service, in our framework, has at present Software subclass which is further sub-classified into disjoint classes Application and OS.

The environmental context is provided by the various classes in the Environment ontology. Humidity, Sound, Light and Temperature are different environmental information we are utilizing in our framework. This sensed information is available though different sensors deployed in the smart environment, and used by the applications to adapt their behavior. An Environment is unionOf all different variables (temperature, light, sound and humidity) mentioned above. Each of them has hasParameter property which links them to the different information gathered from environment. For Sound, the hasParameter has the range of AudioParameter class, which has subclasses, namely, ACDCParameter (ACDC stands for Average Crossing Direction Change), HarmonicityRatio, Intensity, TransientDetection etc. VideoParameter has been classified into MotionPattern, PixelChangeVariance, PixelPercentageChange, Posture, ZoomComponent etc.

Location ontology, an important aspect of ubiquitous computing environment, has SpatialObject as its top level class. This class is equivalent of SpatialObject

defined at NASA Jet Propulsion Lab space ontology[2]. We have imported this ontology into our space ontology, as it describes useful information related to spatial objects. Place is a SpatialObject and has IndoorPlace and OutdoorPlace as it two subclasses. Each Place has hasEnvironment property which describes the environment conditions like temperature, humidity etc. A Place is a isPartOf some other Place. As we have defined ontology for the home domain, we have concepts like BedRoom, BathRoom, DinningRoom and LivingRoom etc. in our ontology. SubRoom isPart of Room, and represents an interesting place inside room such as OnBed, BesideDinningTable, InFrontOfTV, InSofa etc. LocationContextObject is anything which can have location context, having properties of locatedIn, locatedNearBy, locatedFarAwayFrom etc.

```
...
<owl:Class rdf:ID="Activity"/>
<owl:ObjectProperty rdf:ID="generatesEvent">
 <rdfs:domain rdf:resource="#Activity"/>
 <rdfs:range rdf:resource="#ActivityEvent"/>
 <rdf:type
   rdf:resource="&owl;InverseFunctionalProperty "/>
 <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="ActivityEvent">
 <owl:equivalentClass>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#InstantActivityEvent"/>
    <owl:Class rdf:about="#IntervalActivityEvent"/>
   </owl:unionOf>
  </owl:Class>
 </owl:equivalentClass>
 <rdfs:subClassOf
 rdf:resource="&contellocation;LocationContextObject"/>
</owl:Class>
<owl:Class rdf:ID="IntervalActivityEvent">
 <rdfs:subClassOf rdf:resource="#ActivityEvent"/>
 <rdfs:subClassOf
   rdf:resource="&conteltime;IntervalEvent"/>
</owl:Class>
<owl:Class rdf:ID="InstantActivityEvent">
 <rdfs:subClassOf rdf:resource="#ActivityEvent"/>
 <rdfs:subClassOf
   rdf:resource="&conteltime;InstantEvent"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="containsActivity">
 <rdfs:domain rdf:resource="#Activity"/>
 <rdfs:range rdf:resource="#Activity"/>
 <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>
...
```

```
...
<owl:Class rdf:ID="Environment">
 <owl:equivalentClass>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Humidity"/>
    <owl:Class rdf:about="#Light"/>
    <owl:Class rdf:about="#Sound"/>
    <owl:Class rdf:about="#Temperature"/>
   </owl:unionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Light">
 <rdfs:subClassOfrdf:resource="#Environment"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasParameter"/>
   <owl:allValuesFrom rdf:resource="#LightParameter"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Parameter"/>
<owl:Class rdf:ID="LightParameter">
 <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>
<owl:Class rdf:ID="LuminousIntensity">
 <rdfs:subClassOf rdf:resource="#LightParameter"/>
</owl:Class>
<owl:Class rdf:ID="Bright">
 <rdfs:subClassOf rdf:resource="#LuminousIntensity"/>
</owl:Class>
<owl:Class rdf:ID="TotalDark">
 <rdfs:subClassOf rdf:resource="#LuminousIntensity"/>
</owl:Class>
<owl:Class rdf:ID="Dark">
 <rdfs:subClassOf rdf:resource="#LuminousIntensity"/>
</owl:Class>
...
```

Figure 9: Few definitions from Activity and Environment ontology in OWL

Temporal information is ubiquitous in real world situations and also considered as common need for ubiquitous computing applications. For time, we are using

[2] http://sweet.jpl.nasa.gov/ontology/space.owl#

the concepts from DAML-Time ontology [29]. TemporalThing, a general concept, has subclass of InstantThing, IntervalThing and Event. InstantEvents (subclass of Event and InstantThing) can be thought of points which don't have any interior points e.g. entering a room, turning the TV on, and turning lights off. While IntervalEvents (subclass of Event and IntervalThing) denote events, that span some interval of time e.g. watching movie, playing games, or attending the meeting. Every TemporalThing has begins and ends properties pointing to the InstantThing and denotes its beginning and end. inside relation is between IntervalThing and InstantThing stating that some instant is inside the interval. before indicates that some TemporalThing (sleeping) has its end before the beginning of some TemporalThing (waking up). More details of our different ontologies can be found at our website[3].

5.3 Context Repository in CAMUS – multi-domain data management, knowledge sharing and querying

The Context Repository provides the basic storage services in a scalable and reliable fashion and contains the domain ontology and context information (including both elementary and composite context).

Domain Ontology.
Domain ontology contains the domain concepts and properties with formal semantics described in OWL and explained in detail in the context modeling.

Context Information.
Here context is any information saved by the Feature - Context Mapping layer gathered from the environment through lower layers of the architecture and explained in detail in the context modeling.

Meta-Information.
Having well structured meta-information about the various characteristics of the system allows flexibility and acts as a customizable solution for the specific needs of the use case. In our framework, we save meta-information about the devices (D in figure 2), sensors access mechanisms (S), feature to context labeling (L) (used by Feature - Context Mapping layer) as well as the meta-information about the input, output and capabilities of various pluggable reasoning modules (R).

[3] http://ucg.khu.ac.kr/ontology/0.1/

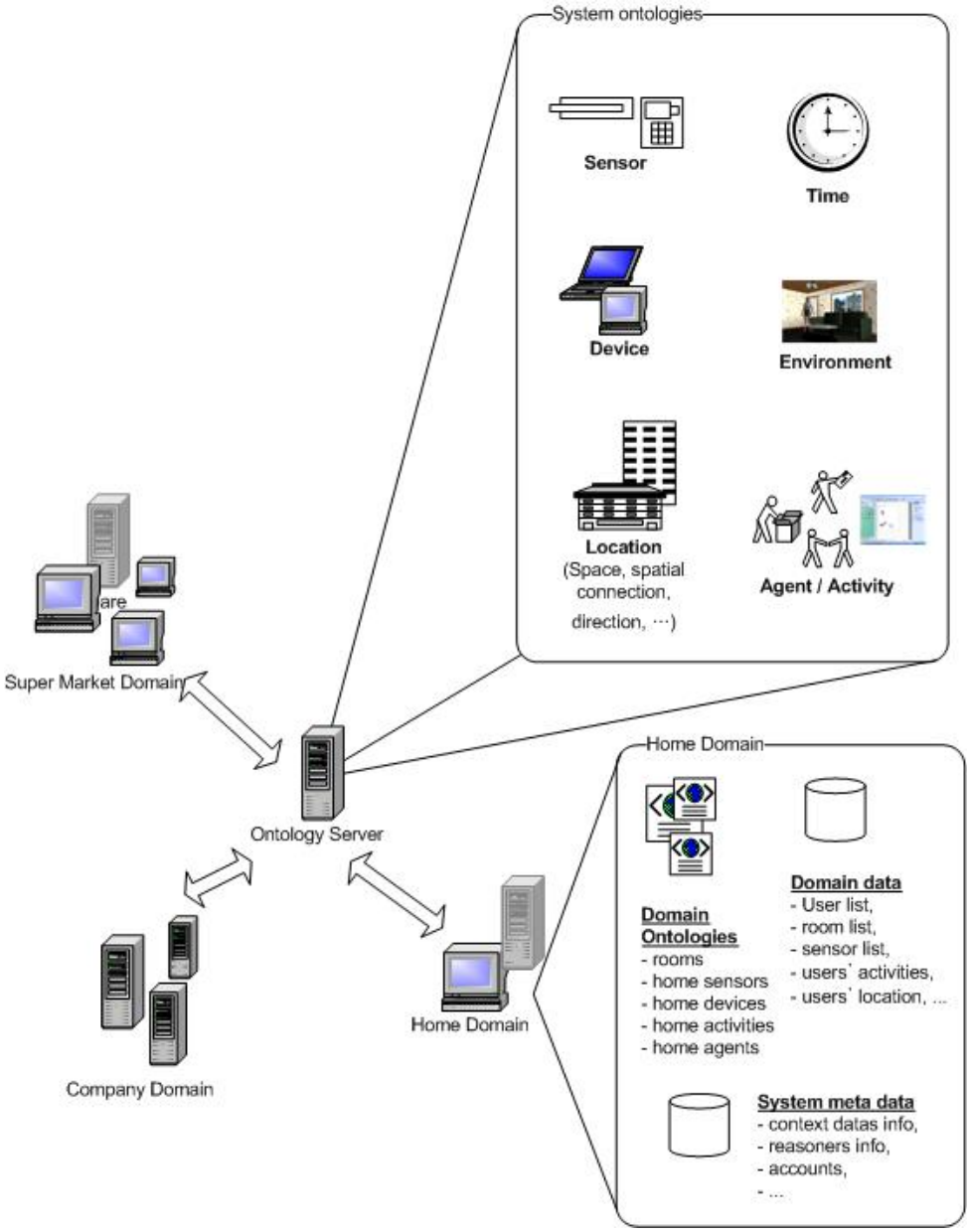


Figure 10: Context Repository Structure

One of the main issues in pervasive computing is that how to manage the context data over a large number of domains. A ubiquitous computing system can consist of many subsystems running on various domains such as home domain, office domain, university domain, etc. Furthermore, many ubiquitous systems can collaborate with each other to build a large pervasive environment. The use of

ontology can help sharing the knowledge about data among different domains and systems. However, such a distributed and dynamic environment requires an efficient mechanism to store and retrieve context data over multi-domain repository.

To solve that problem, CAMUS uses OWL format to *store* context data, and maintains a meta-graph to manage the meta-data about all the domain repositories. Using OWL format, the Context Repository can be backed by some kinds of DBMS such as MySQL, or just use text files if the system needs to run on some resource-constrained environment. When handling OWL data using Jena library [30], each database can be considered as a group of models, each model is a collection of contexts. The ontologies defining context data schemas have hierarchical structure, so each context data model itself is a sub-graph of the big graph combining all the ontologies. Consequently, it is feasible to build a meta-graph of all graphs in a ubiquitous environment. That meta-graph stores the information about the models of each domain, names and namespaces of the models, and especially the contexts provided by each model in a hierarchical structure.

Context data can be *retrieved* by RDQL [31] queries. The queries will be parsed into list of condition triples. Then the contexts mentioned in condition triples will be used to search all the models which can provide those contexts from the meta-graph. After that, for each concerned model, all the related statements will be extracted by using the template statement built from the condition triples. Jena library allows the possibility to integrate many statement sets into one model before executing the query.

Because each Context Repository Manager module runs as a service, so it can advertise itself as well as discover other Repository Manager services. Whenever it discovers a new repository, it will integrate the meta-graph of that new repository into its own meta-graph. Then every query will be executed following the algorithm described above.

Example:
The following pseudo code illustrates the query algorithm:

```
Query query = new Query(queryString);
Triple[] triples = get all the condition triples from query
Model model = ModelFactory.createDefaultModel();
for each triple i in the triples list {
  String[] modelLabels = ContextDataFactory.getContextDataNameForTriple(
      triples[i]);
  for each model label j in the model labels found {
    get all the statements related to the triple and add into model
  }
}
//set data source
```

```
    query.setSource(model);
    QueryExecution qe = new QueryEngine(query);
    //execute query
    QueryResults results = qe.exec();
    ContextRecordset rs = new ContextRecordsetImp(results);
```

5.4 Discussion

The purpose of the formal modeling is to invariant meanings of the terms used in the context aware computing. It allows incremental information pooling and interoperability with other systems. For permanent storage, OWL is converted into relational DBMS by using the Jena framework API. This has certain performance limitations and is not very good solution of efficient storage of large scale contextual data. We believe the database storage schemes especially for OWL should be investigated along with different efficient query mechanisms to retrieve stored data.

5.5 Summary

In this chapter, we described why formal modeling is important in heterogeneous context-aware systems. It should be applied to the degree allowed by the domain of the system. Ontologies based on formal modeling approaches support knowledge sharing, reuse and logical reasoning. Basic and detailed context model is presented followed by the context repository, our storage model for such modeling scheme. Jena was used to store data in the MySQL database and RDQL to query over it.

Chapter 6

REASONING ENGINES

6. Reasoning Engines

6.1 Motivation for Multiple Reasoning Mechanisms?

Different types of entities (software objects) in the environment must be able to reason about uncertainty. These include entities that sense uncertain contexts, entities that infer other uncertain contexts from these basic, sensed/defined contexts, *and applications that adapt how they behave on the basis of uncertain contexts.* A middleware infrastructure is expected to facilitate computing entities with a variety of reasoning and/or learning mechanisms to help them reason about context appropriately. Using these reasoning or learning mechanisms, entities can infer various properties about the current context, answer logic queries about context or adapt the way they behave in different contexts.

Agents can reason about context using rules written in different types of logic like first order logic, temporal logic, description logic (DL) [32], higher order logic, fuzzy logic, etc. Different agents have different logic requirements. Agents that are concerned with the temporal sequence in which various events occur would need to use some form of temporal logic to express the rules. Agents that need to express generic conditions using existential or universal quantifiers would need to use some form of first order logic (FOL). Agents that need more expressive power (like characterizing the transitive closure of relations) would need higher order logics. Agents that deal with specifying terminological hierarchies may need description logic. Agents that need to handle uncertainties may require some form of fuzzy logic.

Instead of using rules written in some form of logic to reason about context, agents can also use various machine learning techniques to deal with context. Learning techniques that can be used include Bayesian learning, neural networks, reinforcement learning, etc. Depending on the kind of concept to be learned, different learning mechanisms can be used. If an agent wants to learn the appropriate action to perform in different states in an online, interactive manner, it could use reinforcement learning or neural networks. If an agent wants to learn the conditional probabilities of different events, Bayesian learning is appropriate. Appendix B shows some sample reasoning mechanisms.

6.2 Reasoning Mechanisms for High-level Context in CAMUS

The need for context reasoning modules arises because not all information can be gathered from sensors. Some high-level contexts such as the location or current activity of user can only be inferred base on a combination of many other contexts. Once high-level context is saved in the repository; it is just a normal context like other elementary contexts, and can be used for further inferences.

Context Reasoning layer includes one to many Reasoners which handle the facts present in the repository as well as to produce composite contexts. The Reasoners can provide the entailed knowledge not formally present in the repository using various kinds of logics to support inference; description logic, first order logic, temporal logic and spatial logic to name a few. Moreover, since every context in CAMUS has probability property, many kinds of reasoning over uncertainty such as Bayesian inference or fuzzy logic can also be applied.

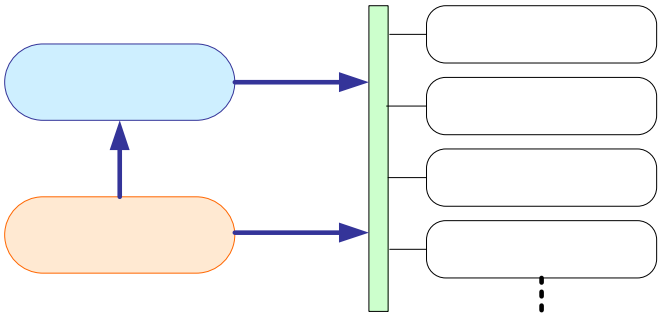


Figure 11: Reasoning Layer in CAMUS

The reasoning service is used by some context mapping services and context aggregators. They invoke the Reasoners through a fixed API, providing the Reasoners with a context data which can be considered as a knowledge base containing all the facts needed for inference. All new inferred facts will be inserted into that context data for later queries. The use of a fixed interface for all kinds of reasoning engine makes it possible to add and handle different Reasoners. The developers can then use any kind of reasoning they want.

To provide more help to developers so that they can concentrate on developing rules or networks for reasoning and not be burdened with the low-level details, CAMUS defines wrappers for each Reasoner type. For example, a wrapper of Jena generic rule Reasoner allows the developer to easily add a new Reasoner just by declaring the rule file name and some namespace abbreviations.

Example

The following example illustrates how to add and invoke a rule-based reasoner:

```
/* declare the prefixes for namespaces */
ContextReasonerManager.registerPrefix("conagnt",
rtmm.camus.vocabulary.contel.Agent.NS);
ContextReasonerManager.registerPrefix("env",
rtmm.camus.vocabulary.contel.Environment.NS);
ContextReasonerManager.registerPrefix("conloc",
rtmm.camus.vocabulary.contel.Location.NS);

/* add a new reasoner providing the fule file */
ContextReasonerManager.addReasoner("Location",   ReasonerType.GENERIC_REASONER,
"etc/contel.rules");

/* declare some statements */
ContextStatement     hasLocation     =     ContextFactory.createStatement(null,
"hasLocation", null);
ContextStatement PastLocationDescription = ContextFactory.createStatement(null,
null, "PastLocationDescription");
sms = new ContextStatement[] {PastLocationDescription, hasLocation};

/* invoke the reasoner to do reasoning, providing the reasoner name, the
context data name and the required statements */
cdm.invokeReasoning("Location", "Data", sms);
```

•Ontology Reasoning Mechanisms

High valued ontologies depend heavily on the availability of well-defined semantics and powerful reasoning modules. The expressive power and the efficiency of reasoning provided by OWL, (the semantics of OWL can be defined via a translation into an expressive Description Logics (DL)), make it an ideal candidate for ontology constructs. The facts gathered from context entities make a factual world in OWL, consisting of individuals and their relationships asserted through binary relations.

Ontology reasoning helps us to find subsumption relationships (between subconcept-superconcept), instance relationships (an individual i is an instance of concept C), and consistency of context knowledge base, provided by Racer Server. In the design phase of formalizing the context entities, OWL reasoning services (such as satisfiability and subsumption) can test whether concepts are non-contradictory and can derive implied relations between concepts.

Let us take an example to see how ontology reasoning can help deducing implied context. In location ontology, the property locatedIn is a TransitiveProperty, and isPartOf is subProperty of locatedIn. So when knowing that Bilbo is locatedIn Bed, and Bed is a part of BedRoom which is part of Home, the system can deduce that Bilbo is locatedIn BedRoom and Home.

Another example is how to map between the features we receive from Feature Extraction layer to simple contexts. Different sub classes of Parameter class have

different hasValue restrictions on sensorTypeID, featureID and quantiziedLevel. Receiving a feature tuple with sensorTypeID = 1, featureID = 1 and quantiziedLevel = 1, we can create an instance of class Parameter with them, and then use OWL Reasoner to infer that the new instance is of type AudioParameter, Intensity and Silence.

•Context Reasoning Mechanisms

However, many types of contextual information cannot be easily deduced using only ontology inference. In addition to ontology reasoning, we can also use logic inference. A set of rules can be defined to assert additional constraints for context entity instances when certain conditions (represented by a concept term) are met.

Over the concepts and relations defined in Cont-el, we can do a lot of reasoning based on many types of logics, such as description logic, description temporal logic, and spatial logic. We will take a closer look at how Cont-el supports these kinds of reasoning.

The spatial reasoning is based on the Location ontology and Region Connection Calculus [33]. We can infer about the spatial relations among the symbolic representation of space, such as `spatiallySubsumes` or `isDisconnectedFrom` relation between two `SpatialObject`. Here we illustrate one of those RCC rules:

```
[ (?x spc:spatiallySubsumedBy ?z),
  (?z rcc:isDisconnectedFrom ?y).
                  → (?x rcc:isDisconnectedFrom ?y) ]
```

Based on the Time concepts in our ontology, we can define a set of rules for temporal reasoning. Temporal relations e.g. `meets`, `before` etc. and their inverses e.g. `metBy`, `after` etc. are taken from [34]. So we can define a set of temporal reasoning rules like this example:

```
[instant-before:
   (?x rdf:type tme:InstantThing), (?x tme:at ?timeX),
   (?y rdf:type tme:InstantThing), (?y tme:at ?timeY),
   lessThan(?timeX,?timeY)
         ( (?x tme:before ?y)]

[interval-before:
 (?x rdf:type tme:IntervalThing), (?x tme:ends ?xE),
 (?y rdf:type tme:IntervalThing), (?y tme:begins ?yB),
 (?xE tme:before ?yB)
```

((?x tme:before ?y)]

The inferred temporal and spatial contextual information can be used for higher level reasoning. For example, categorizing activities into PastActivity, CurrentActivity and IntentionalActivity help defining some more complex inference. Following are some rules taken from our current implementation: (note that each time before calling the time reasoner, we have to update the at property of Now – a special instance indicating the current time, an individual of class NowInstantThing - with the current timestamp).

To infer that an activity is PastActivity
[past-act:
(?a rdf:type act:InstantActivity),
(?a act:containsActivityEvent ?e),
(?a rdf:type act:InstantActivityEvent),
(?e tme:before ?n) ,
(?n rdf:type tme:NowInstantThing)
((?a rdf:type act:PastActivity)]

If agent has Waken Up and is Bathing then the Oven will Reheat the Breakfast

In DLRUS syntax [17], this rule can be expressed like this:

((OvenReheatingBreakfast ┌ () ┌ (WalkingUp ┌ PastActivity ┌ Bathing ┌ CurrentActivity))

And here is the realization in Jena rule syntax:

```
[reheat:
 (?a1   rdf:type   tme:WakingUp),   (?a1   rdf:type
act:PastActivity),
 (?a2   rdf:type   tme:Bathing),   (?a2   rdf:type
act:CurrentActivity)
       → [(?o act:isActorOf ?a3),
           (?a3 rdf:type acthome:ReheatBreakfast)
              ← (?o rdf:type devhome:Oven),
makeInstance(?a,                        act:isActorOf,
acthome:ReheatBreakfast, ?a3)]]
```

Such temporal concepts and relations can play a useful role in the reasoning about contexts. All concepts and relations are written using the Protégé 2000 [35] which allows writing vocabularies in OWL. At present, we are using the Jena

Semantic web toolkit to insert the context information as it allows parsing, managing, querying and reasoning the ontologies programmatically.

However, Jena has limited support for other types of inferences, for example default reasoning and uncertainty reasoning. So we are considering using some other reasoning mechanisms, such as Bayesian network for uncertainty reasoning, and the Theorist framework for default and abductive reasoning. Cont-el ontologies and current reasoning mechanisms over it can provide contextual information, as input, for those higher inferences.

6.3 Discussion

To enrich the knowledge base of a ubiquitous system, using reasoning mechanisms to infer more information is a noble solution. Unifying the interface of all reasoning engines so that the developers can easily add new reasoners and handle the reasoning process is quite a good approach.

However, this approach requires a huge effort to learn as many types of reasoners as possible and then make each reasoner type a simplified interface. Currently, only the wrapper for Jena generic rule reasoner is done. There is still a lot of future work needed to build the wrappers for other kinds of reasoning engines, especially some engines which can deal with the uncertainty like Bayesian Net or Fuzzy logic reasoner. Those reasoners can provide more accurate inference results, and they are suitable for the ubiquitous environments which are full of uncertainty.

Furthermore, we should consider applying some data mining and AI techniques into our middleware. For example, from the historical information of user location, user activity, the environment features, combine with user profile, some data mining algorithms can be used to mine the association rules which describe user preferences or user route. Another example is that we can build the decision trees to predict future actions of user. Training the Bayesian net, creating the neuron network or applying some machine learning mechanisms to learn more about the expected behaviors of the ubiquitous systems to adapt the preferences of users are also some future directions.

6.4 Summary

CAMUS utilizes a variety of reasoning and/or learning mechanisms to help computing entities reason about context appropriately. Many reasoning mechanisms should be supported to provide the developers a large range of selection. Also each type of reasoning mechanism has its own advantages. Ontology reasoning (inferencing over OWL format ontologies and context data),

description logic, temporal logic and spatial logic reasoning have been studied to apply into CAMUS.

All the reasoners are managed and invoked through a unified interface. In future, we will continue to improve the reasoning layer in number of supported reasoner type, as well as in the quality of the reasoners to be used.

Besides, some data mining techniques and AI algorithms will be considered in the next step of CAMUS to make the ubiquitous environment more intelligent.

Chapter 7

CONTEXT PROVISION

7. Context Provision/Aggregation

7.1 Motivation

For success of large scale open distributed systems, the need for interoperability is not new issue. The emergence of different architectures for context aware ubiquitous computing systems, whether they are infrastructure based (middleware approach) or standalone application specific systems, will require handling their need for interoperability. Though much work is being carried out in different dimensions in context aware ubiquitous computing systems, and different research studies have proposed to use semantic web supported ontologies for contextual data representations, but our study/practical experience [36] of such approach has showed that it lacked performance. Since purpose was to support semantic interoperability with other systems, but for some reasons or the other, representing all the contextual data in OWL is not efficient approach. Context aware ubiquitous computing systems sometimes require different methods/techniques because of efficiency or lack of resources of mobile devices.

Context aggregation service is responsible for satisfying certain context queries. Each context aggregation service performs a specific function. An example service can be detecting that user has awaken and performing certain actions. Based on required contextual information, it can either utilize the ontology reasoning module or context reasoning module or both. Upon detecting that certain context is composite one, it will retrieve meta-information from the repository about the specific context reasoning module providing the composite context. Once retrieved, it will invoke the corresponding reasoning module and return the result back to the application requesting the context. The current architecture of context aggregation is based on service oriented architecture [37].

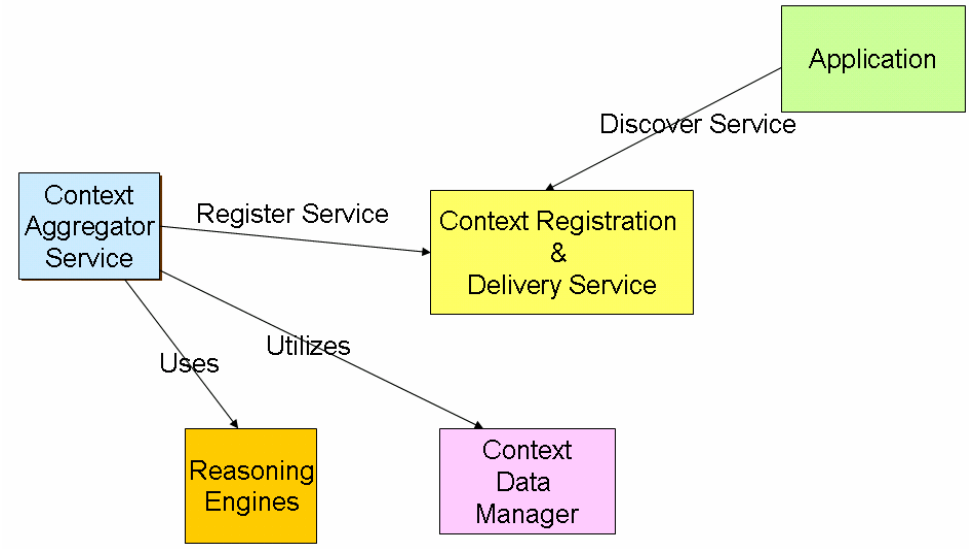


Figure 12: Context Aggregator Functioning

Then, applications can find it via querying the delivery service and requesting for their specified context aggregator. Since context aggregator produces composite context by using lower level basic context, it needs the support of reasoning engine modules to produce such composite context. The basic context is gathered from the CAMUS through the context data manager. Once, the composite context is found by applying reasoning techniques on the set basic of basic contexts, it is again saved in to context repository using the context data manager interface.

Interface to request for contexts from CAMUS

Context Consumers (applications) can query the contexts from CAMUS by calling the API of the Context Aggregators.

Request for contexts using RDQL-like query

Example: query for the location of Bilbo and find all the people who are in the same place with him.

```
/*********************************************************
* get the vocabularies from Context Catalog
* If the vocabularies are already known, we can skip this step
*********************************************************/
String uri = ContextCatalog.LocationOntology.Location.uri;
String name = ContextCatalog.LocationOntology.Location.name;
String locatedIn = ContextCatalog.AgentOntology.Agent.locatedIn.label;

/*********************************************************
* get the location
```

```
* SELECT ONE will select only one available context
*********************************************************/
ContextObject[] locations = LocationCA.queryContext(“ SELECT ONE ?location
WHERE (?user locatedIn ?location)
AND (?user name ‘Bilbo’)”);

if (location!=null) {
        ContextObject location = locations[0];

        String locationURI = location.getProperty(uri).getValue();
        String locationName = location.getProperty(name).getValue();

/*********************************************************
* get the list of people
*********************************************************/
ContextObject[] people = LocationCA.queryContext(“ SELECT ?pp
WHERE (?pp locatedIn ?location)
AND (?location uri ‘“ + uri + “’)”;
    }
```

Request for contexts using Template

Example: query for all information of Bilbo.

```
    /*********************************************************
    * get the vocabularies from Context Catalog
    *********************************************************/
    String agentClassName = ContextCatalog.AgentOntology.Agent.getClassName();
    String agentName = ContextCatalog.AgentOntology.Agent.name;
    String agentUri = ContextCatalog.AgentOntology.Agent.uri;
    String agentAge = ContextCatalog.AgentOntology.Agent.age;
    String locatedIn = ContextCatalog.AgentOntology.Agent.locatedIn.label;

    /*********************************************************
    * create the template
    *********************************************************/
    ContextClass agentClass = LocationCA.getContextClass(agentClassName) ;
    ContextIndividual agent = LocationCA.createIndividual(agentClass) ;
    agent.addProperty(agentname, “Bilbo”);

    /*********************************************************
    * search for contexts which match the template
    *********************************************************/
    ContextObject[] agents = LocationCA.searchContext(agent);

    if (agents!=null) {
        ContextObject bilbo = agents[0];

        String bilboURI = bilbo.getProperty(agentUri).getValue();
        int                 bilboAge                =                (new
        Integer(bilbo.getProperty(agentAge).getValue())).intValue();

        ContextObject bilboLocation = bilbo.getProperty(locatedIn);
    }
```

Request for contexts using simple search

Example: query for all context related to the string “Bilbo”.

```
    ContextObject[] contexts = LocationCA.searchContext(“Bilbo”);
```

The result will be an array of individuals which has the label contains "Bilbo", for examples : BilboLocation (of class Location), BilboActivity (of class Activity), …

Request for context using URI

Example: query for all info of Bilbo, given the URI of that agent

```
    ContextObject bilbo = LocationCA.getContext("http://khu.ac.kr/Bilbo");
Or:
    ContextIndividual                          bilbo                          =
    LocationCA.getIndividual("http://khu.ac.kr/Bilbo");
```

7.2 Dynamic Interaction

The approaches, mentioned (e.g. representation of contextual data in OWL) above, may not be used in all cases but even then, interoperability is an important issue for successful deployment of the context aware ubiquitous computing systems. Service oriented computing paradigm if applied to context aware ubiquitous computing systems can give promising results. One such example of a service can be user location service, providing user location in a particular area. The goal is to provide semantically rich and flexible model that can simplify development of context aware ubiquitous computing systems. If we think of facilities provided by some ubicomp system as services, we need to define the important parameters of services for such vision. The three entities in SOA are service provider, service consumer and the service brokers. In context aware ubiquitous computing systems, the analogy will result in context provision services (called context aggregators above), service consumers or applications and context brokers. Such brokers can be centralized or part of some federation as this is not the main issue discussed here. A workflow of such a system can be made as shown in figure below. Here composition is the final step which will be performed when user requested some service not available with the broker and it needs to find a set of services which can provide same goal.

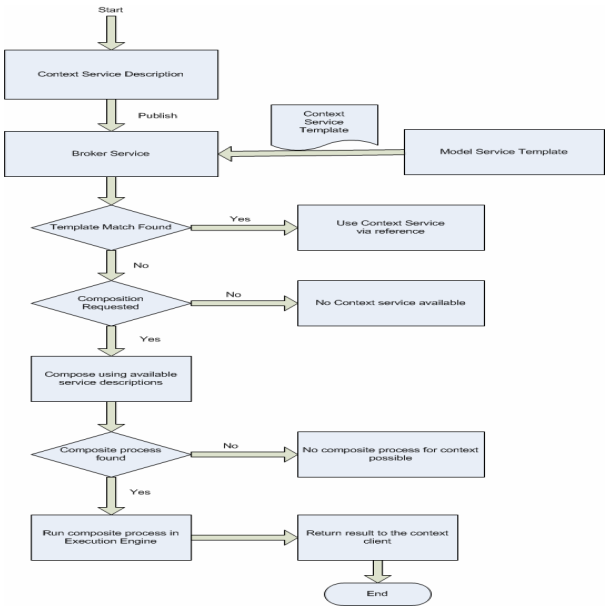


Figure 13: System flow for composition of context aggregators

The service is described semantically after being developed. It is then registered with broker service and the job of service provider is done. Now, if the client wants to find some service, she will make a template and submit it to the broker. If some match is found, it will return that service reference but if no such requested service is available, the broker will check if the client has allowed composing services if no single service match exists. It will then transfer the template to the composition component which will handle composition and run the composite process inside its engine.

7.2.1 Context Service Description

For discovery and composition of context services, the first and most important requirement is their appropriate description. Since only syntactic interface of the service is not sufficient for its discovery and usage, Quantitative and Qualitative (or Q) semantics are needed. Quantitative semantics is related to context service specification i.e. service name, operations/methods provided by the service through its exposed interface along with the mechanism used to infer higher (composite) context from the raw data (basic context) in those operations. The following attributes of a service are considered important for attaching quantitative semantics.

Table 3. Quantitative attributes of a context service

Ontological class that closely resembles the *context service name* to signify its domain

Ontological class of *operation i*
Ontological classes of the *Pre-conditions of operation i*
Ontological classes of the *Inputs of operation i*
Ontological classes of the *Outputs of operation i*
Ontological classes of the *Post-conditions of operation i*
Ontological classes of the *Possible exceptions thrown by operation i*
Ontological classes of the *reasoning mechanism used to infer higher level context*

Since operations of a context service correspond closely to its name, both the ontological concepts for the operations and the service domain are kept in one ontology named service ontology while concepts matching different data types for inputs and outputs of operations are kept in data type ontology. Similarly, preconditions and post-conditions are maintained in conditions ontology while to mention the quality of context/s being provided by a service, reasoning mechanism used to infer higher context is considered here as a metric and represented in reasoning approach ontology. This reasoning approach ontology will help us to select the best service if a group of services are providing same context but using different mechanisms to infer higher context. The reason we have included this parameter in each service is that we believe that one reasoning mechanism cannot be used for inference of different contexts. For some higher level context, simple rule based mechanism may be efficient to infer it from simple raw context but for some other higher context, complex reasoning mechanism may be needed. An example can be detecting whether user is sleeping. One simple way may be to use simple rule like if it is no noise in room, AND light is off, then user is sleeping but another approach may use complex Bayesian network to infer the status of user in the room.

Qualitative semantics of an operation i of the context service represents its excellence. This excellence will be in terms of its execution time, context freshness, reliability and availability. Since all these excellence parameters are comparative parameters, they can be represented in the form of tuple as shown below.

```
{excellence parameter, comparison operator, value, unit}
```

An optional hardware used parameter can be included in qualitative semantics to mention the hardware which provided the basic/raw context to the service. An example usage can be user location because we can get location through different hardware providing different accuracy e.g. using WLAN, RFID, ibutton etc. Once context service is fully described, it can be then registered with the broker service. Here, the job of the service developer is done as he successfully registers his service after fully annotating it by describing its Q semantics.

7.2.2 Context Template (Target Context) – CT or TC

The client application submits the target context to the broker (The target context is used by the broker service) to find the best matching available service. There are two possible scenarios in our system. Either the application developer is interested in only finding one service which can fulfil his requirements submitted via target context, or he specifies that if the broker service cannot find one suitable service, then the job of finding a set of services which can perform the same functionality can be delegated to the context service process composer (CSPS). Target context specifies the abstract function the client application is interested in, with the set of input and output concepts it is looking for. It will be actually a set of service level parameters (SLAs) and Q semantics the client is interested in. E.g. if the client is interested in user location and it is willing to provide user URI and expecting Location in terms of GPS location, then it can be defined roughly as:

```
Domain (CT) = Location Provider Service
Ontological class of required operation = UserLocation
Ontological class of required input = URI
Ontological class of required output = GPSLocation
Context Freshness < 5 Sec
Hardware Used = RFID || iButton
```

In this way, the client specifies its requirements in more expressive way and there are more chances to find suitable service as compared to simple search based on keywords.

7.2.3 Context Discovery Service – Broker

Context discovery/broker service is registration/advertisement and discovery service whose function is to maintain a set of services registered with it and finding possible matches of service if requested by client application. As context service description is provided by each service to the broker, the search in broker is based on semantics provided by context services. The target context is matched semantically against a set of service descriptions available with the broker and is based on Q semantics. Such descriptions are maintained in the Context Service Descriptions (CSD) Repository. In CAMUS, at present, this job is being handled by Context Delivery Service.

7.2.4 Context Service/s Process Composer

Reusability is an important concept in traditional as well as service oriented paradigms. Process composer allows us to create new values from the existing context services. Composite services apply in many situations as one service

depends on another service for certain context. An example can be of a user status detection service which depends on another service for providing user location. This composition is either based on achieving a set of goals or following the workflow systems approach. Once a certain process is compiled, it is then enacted in the Execution Engine, which is responsible for scheduling, monitoring and managing events among a set of services in the process.

7.3 Architecture required for dynamic interaction

The system architecture is based on the required components which are necessary for successful operation of such a system. This is self explanatory because of the description given above and we will not provide its details here.

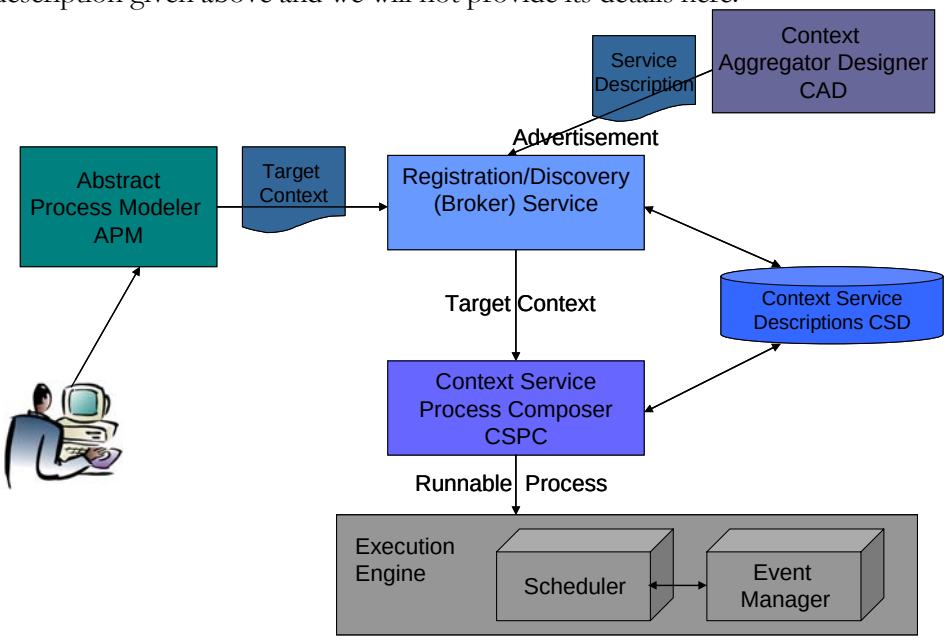


Figure 14: Required architecture for dynamic interaction

7.4 Discussion

Although in this chapter, and the chapter on delivery services, it is emphasized on semantic matchmaking and dynamic composition of service, there is still one issue which is not being dealt. The purpose of delivery and composition system is to find and compose services in most efficient way but the decision of which services should be required by the application are still with the application developer. There is no way that applications can decide intelligently that which context they need and in this way, query the delivery and composition subsystem to find those interested contexts. In simple words, it is two steps process, first is the identification of the context an application requires and second is provision

of the context to the interested application. First goal is presently left with the developers of context-aware applications while second is being dealt by the system we are currently enhancing i.e. context delivery and dynamic composition.

7.5 Summary

We presented why context provision is important along with dynamic composition of services. Once system gathers useful information from the environment and its surroundings, and saves it in the form of context, the question remains how to deliver this context to the applications. So, we discussed context aggregators and their goals to provide context. Also, dynamic composition of context-aware services is discussed if there is no one service which can fulfill application requirement and composition of multiple services is required at runtime.

Chapter 8

CONTEXT DELIVERY SERVICES

8. Context Delivery Services

8.1 Motivation

A context aware system gathers and dissipates information relevant to a user in order to enrich the software functionality and facilitate adaptive behavior. With the plethora of information gathered from the physical and computational environment, there is a need for an efficient delivery mechanism to filter out unrelated information and communicate the relevant contextual information to its respective clients. Context delivery is the final step in the context awareness process starting from gathering raw data from environment and logical sensors, feature extraction, context synthesis and reasoning to finally delivering context to the interested client or application. It facilitates the context-based interactions by relating the desired contextual information with the interested context consumer. The clients of a context delivery system include context producers (context aware middleware), that produce and advertise the existence of context; and context consumers (context aware applications), that query certain type of context they are interested in, which is consumed on availability.

In a simple service provider / consumer environment, it is assumed that the service provider and service consumer know each other and the service is published, discovered based on syntactic parameters i.e., the consumer can lookup the service solely basing on syntax queries. In case the service provider and consumer do not use the same syntax, the desired service could not be provided even if it exists. Similarly, the service interfaces are static and known in advance by the interacting clients. No new functionality might be added to the services while they are executing. Moreover, security and access control models for information are well understood in case of static service provider and consumer interactions.

These assumptions no longer hold true in case of context aware environments. A context aware system maintains a vast set of heterogeneous context clients (applications). Furthermore, new type of applications and contextual information might be added to the system at runtime which should be incorporated in the system without requiring an update in the service interface towards the existing client applications. Similarly, only syntax-based queries might not be able to

identify the desired service since the representations of the 'same concept' might be different at the client and the service. Finally, some properties of the information raise unique challenges for the design of an access control mechanism such as: information can emanate from more than one source and flow through nodes administrated by different entities and it might change its nature or granularity before reaching its final receiver; policies might not be installed on all nodes before-hand and the user might not possess knowledge about the complete policies existing in a context aware environment; and minimum human intervention should be needed to keep the system running in a reliable and secure manner.

In order to meet the additional challenges posed by the peculiar nature of context aware systems, we can no longer rely on traditional service registration and delivery mechanisms. The main motivation behind context delivery services in CAMUS is two fold:

- To provide a discovery and registration mechanism which can utilize the underlying contextual information's syntax as well as semantics in order to make more intelligent and accurate matchmaking decisions between the clients and services
- To incorporate dynamic and autonomous access-control mechanisms in the context delivery process to ensure privacy and overall integrity of the system

8.2 Requirements of Delivery Services

The basic requirements of the context delivery system can be enumerated as (a) the system should be able to deliver a vast set of context to a variety of diverse applications, (b) the system should be extensible in order to incorporate new applications and new contextual information as the context aware system matures, (c) capability to define logical constrains for more desired matching of the context advertisements and context queries, (d) cater for semi-structured data in case the context aware applications that need to utilize contextual information does not provide exact specifications of the required context, (e) policies defined for access control should be dynamic in nature in order to incorporate changing system trust level, (f) access control mechanism should be autonomous to minimize human intervention, and (g) access control should be interactive to cater for situations in which user has incomplete knowledge about the existing policies of the system.

These requirements suggest that the context delivery mechanism and corresponding representation scheme for advertising and querying context and defining policies must be flexible to incorporate a variety of applications and clients, yet expressive enough to avoid ambiguity in delivering desired contextual

information. In our middleware, context is represented using OWL and Ontologies which not only allow representation of available contextual information, but also facilitate representing relationships between different concepts and sharing of similar understanding between various entities in a context aware system. Keeping in view the requirements of the context delivery service and the representation scheme of the underlying data model for context, semantic web concepts for matchmaking [38] seem to be most closely related to the requirements stated above for the delivery of context from the middleware to the applications.

In CAMUS, the context delivery services are present at the top layer of the system performing the job of searching appropriate context aggregators and delivering them to the applications. These services perform the registration, lookup/query and notification functions. Context Aggregators register with the registration service to provide the information about the context they can deliver. Interested applications and agents query the registration service to find services of their interests. The registration service upon finding appropriate aggregator, returns the handler to the requested clients. Each context aggregator specifies the context it provides, by utilizing the concepts defined in the ontology repository. This standard schema sharing allows the different kinds of entities to be described and utilize by registration service to find useful services needed by the applications, thus allowing a flexible mechanism for exchanging descriptive information of various entities.

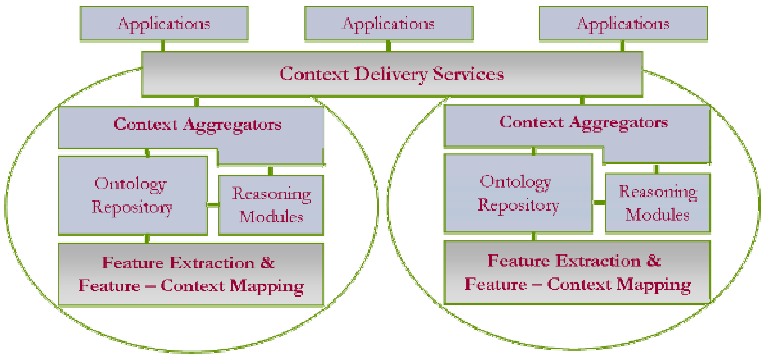


Figure 15: Context Delivery Service Placement in CAMUS

8.3 Semantics-based Discovery and Registration

The fundamental operations that take place in a service registration and matchmaking process include:

Advertising: The party interested in making available some service publishes an *advertisement* with the matchmaking service to let other interested parties know about its existence
Querying: *Query* is submitted to the matchmaking service to find out a relevant advertisement among the ones available. Constraints are provided in the query over aspects of interest to filter out unrelated services.
Browsing: The matchmaker stores the advertisements in a repository that can be *browsed* by interested parties for manual search of advertisement of interest.

A set of description languages such as WSDL, UDDI [39], ebXML present mechanism for similar operations but the limitation lies in the lack of ability to assign logical constraints on advertisement and queries, and ability to deal with semi-structured data input. Semantic matchmaking offers strong representation languages such as RDF and its extension DAML+OIL [40]. RDF is an ontology language that allows expressing contextual information along with their relationships but is limited in representing constraints on information. DAML+OIL extend RDF to incorporate Boolean operators and quantifiers which provides a great deal of expression in defining constraints for matching advertisement and queries. Currently, OWL is the standard language for ontologies description and will be used. In [41], a framework is presented which uses semantic web for matchmaking in e-commerce. Similar concepts can be extended for the provision of context delivery in the context aware middleware systems. We can view the context consumers and context producers as the interacting parties where context producers advertise the context they produce and context consumers query and utilize the corresponding contextual information.

At the modular breakdown, the context delivery using semantic matchmaking is similar to regular delivery services. The actual difference lies in the representation scheme for context announcement and query specification. The semantic meanings attached with the constraints specified for the contextual data available or required leverages flexibility and common understanding of concepts. In addition, since the underlying representation scheme is ontology-based, it enables extensibility of the concepts, reusability and inheritance. These benefits are reflected in delivering context as the delivery mechanism uses the same semantic concepts. The important aspects are to identify the format for context advertisement and query. Existential and universal quantifiers and Boolean expressions supported by the representation language will be dealt with in detail to use them for the purpose of specifying constraints. Inheritance, specialization, generalization and other relationships can exist between various concepts in ontology. These relationships can be used to process queries in which specific details are not provided by the subscriber.

8.4 Policy-based Interactive Autonomous Access Control

A pervasive context aware system senses information from the physical and computational environment and infers useful contextual situations in a user-centric manner. This information, more often than not, is very personal to the user and if available to an unauthorized person can be a serious intrusion in the user's privacy and security. Access control to information is as critical as it is challenging in context aware computing. The properties of the information raise unique challenges for the design of an access control mechanism in context aware environment such as: information can emanate from more than one source and flow through nodes administrated by different entities; it might change its nature or granularity before reaching its final receiver [42]; policies might not be installed on all nodes before-hand and the user might not possess knowledge about the complete policies existing in a context aware environment. Moreover, keeping in view the scale and complexity of the context aware systems, minimum human intervention should be needed to keep the system running in a secure and reliable manner.

The Semantic Web as a whole is largely conceived as a completely open system, in which everything is published for everyone to see. It is far from clear how access control could or should be applied, e.g., to the information in an ontology or a knowledge base. Reasoning engines typically can't enforce security policies, and the DAML language, for instance, has no facility to limit visibility of concepts or attributes [43]. It is clear that the capabilities of the context representation scheme can not be exploited to enforce access control over the information contents. The context delivery mechanism fills this void by incorporating dynamic policies at the services level as well as at the system level on the whole.

Various context aware systems are directing efforts now to incorporate privacy and security techniques in the pervasive environments. The foremost concern is to define access control policies that can be suitable in a pervasive environment. The major concerns are: the policies need to be dynamic in nature; the granularity of control to information needs to be identified; how to cope with changing policies basing on the context at run-time; how can the clients trust the system while providing personal information to it for access and validation. Some of these issues can be overcome if autonomic access control techniques are employed in context delivery. In [44], the authors discuss about the following three operations for making access control decisions and defining policies:

Deduction: Deduction is the default process that most access control techniques employ. In deduction, given a defined policy and a set of credentials provided by the client, it is decided whether to grant the access or not. It checks if a request for access can be granted or not.

Abduction: In abduction, given a defined policy and a request to access some resource or service, it is decided what minimum credentials are required so that the given request can be granted. This process is specifically useful in context aware environments where the client might not what credentials it needs to provide to access a specific service. In that case, the delivery service will request the client to provide the missing credentials and if the clients provide valid credentials, the request is granted, else denied.
Induction: Induction utilizes a heuristic function along with some positive examples of scenarios in which the request should be granted and some negative examples of scenarios where the request should be denied. Basing on this information, the induction process tries to identify the policy that satisfies the validity of the granted requests. The induction process is useful in the case where a single static policy can not be defined. This is true for context aware systems since the context changes at run-time so should the policy to incorporate the new situation.

By utilizing these three operations, an interactive and autonomous access control mechanism can be put in to effect. Existing approaches of access control do not rely on such techniques and hence the policies are mostly static which can not be modified at run-time to incorporate changing system privacy requirements in a pervasive environment. In [45], the authors extend the existing access control model by incorporating roles and delegation mechanism but they only focus on controlling access to devices such as printers and do not define access control over information which is more critical in a pervasive environment. In [42], the authors argue that the access control should be at the information level since the major concern is to maintain the privacy of the contextual information. They provide architecture to maintain access control in a distributed fashion. However, the access control mechanism is not interactive and autonomous. In CAMUS, we intend to incorporate an autonomous access control mechanism which can also utilize the underlying semantics of the system to provide services more closer to the users needs and capabilities.

8.5 Architectural Overview of Context Delivery Module

The context delivery module handles functions related to service registration and lookup but it is different from regular service repositories in that the matching process makes use of the underlying semantics of the system while providing controlled access to the context providers using autonomous policy based access control mechanism. The overall architecture of the context delivery module is represented below.

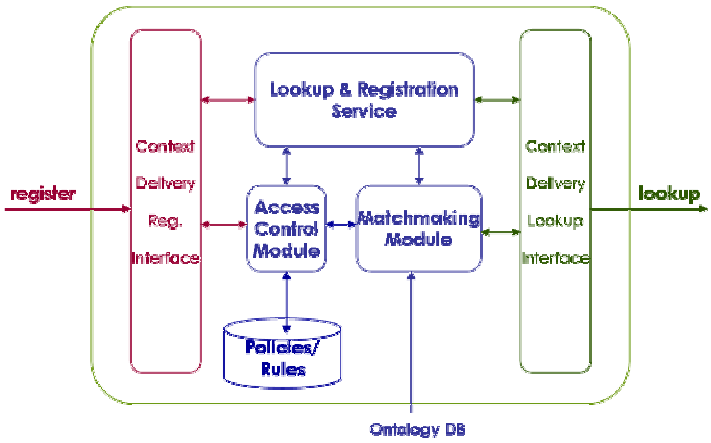


Figure 16: Overview of Context Delivery Service Architecture

The services (context aggregators) utilize the registration interface to make their information known to the applications. Lookup interface enables the applications to find appropriate matching context providers. Policies/rules database contains the system level policies as well as optional aggregator services level policies and rules defining the requirements or conditions to access some specific service provided by the context aware middleware. This process is handled by the access control module.

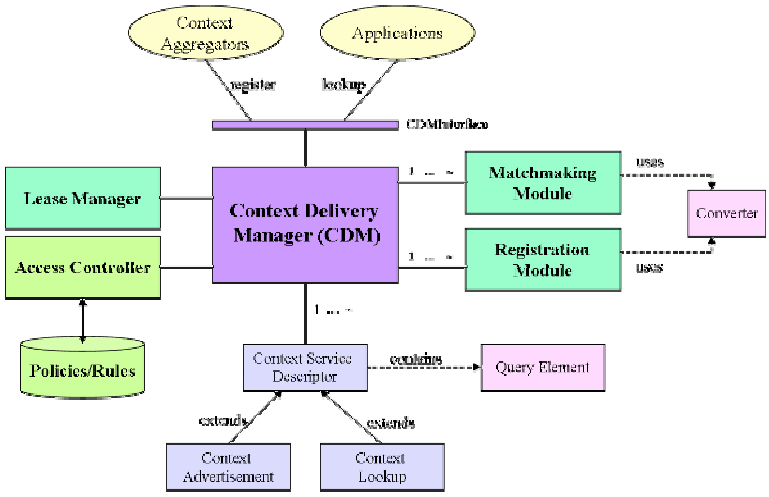


Figure 17: Context Delivery Service - Modular Breakdown

The matchmaking module matches the appropriate service with the client provided the access control policies are not violated. Further breakdown of the context delivery module is represented in the figure above.

The two main entities the delivery services interact with are the context providers i.e., the context aggregators in the CAMUS middleware and the context consumers i.e., the context aware applications that utilize the contextual information provided by the middleware. These modules utilize the interface exposed by the delivery services to perform actions like registering a service (aggregator), looking up a service or registering for event notifications regarding services or events of interest. Internally, the context delivery service is composed of a *Context Delivery Manager* module which handles the requests from the aggregators and applications and dispatches them to the corresponding sub modules. Every time a new registration request is received from the context aggregator, an instance of *Registration Module* is created to process the request. *Matchmaking Module* handles the lookup and matching request on behalf of the applications. This module incorporates semantic matchmaking capabilities to make use of the underlying semantics of the context data structures and ontologies to come up with a match to the most appropriate set of context aggregators to fulfill the applications' contextual requirements. *Access Controller* manages the *policy/rules database* which contains system level access policies and optional service (aggregator) level policies. A match is computed and returned to the service only if the access control policies are not violated. If a match is found and the matched service has some access policies, then the matchmaking module utilizes the access controller to identify whether the access can be granted or denied.

Lease Manager keeps track of the freshness and validity of the registrations and matchmaking requests present with the manager. It renews or cancels the requests basing on the time interval the service/client provided while requesting a service or event registration. This soft state registration process helps in identifying stale registrations and cleaning up the context delivery registry. The data structure used to keep the service registrations or matchmaking requests are represented by the *Context Service Descriptor* which is specialized into *Context Advertisement* and *Context Lookup* for specifying specialized attributes related to service registration or service lookup respectively.

8.6 Discussion

The research and development for context aware middleware systems is in an evolutionary stage. A standard development methodology or design specifications does not exist as yet. Same goes for various context delivery techniques. Many systems employ CORBA based communication and delivery infrastructure [24]. This facilitates in discovery and event notifications in a distributed environment but the context delivery mechanism only deals with the syntactic comparison and does not provide semantic matching capability. Agent based approaches for context sharing and delivery provides dynamism and flexibility but need an

extensive support platform for hosting agents which might not be feasible for all the context aware applications. Web services provide standard representation semantics which can support a variety of applications but they do not support logical constraints on the parameters of communication messages. Similarly, all the existing paradigms lack support for semi-structured data in case the context requesting application has limited knowledge about the context producer or is interested in only a subset of contextual information provided by a specific producer. Use of semantic matchmaking in context delivery facilitates the requirements specified for our context delivery service as explained later. Currently, efforts are underway to enable semantic matchmaking capabilities in the UDDI repository for web services [46] [47]. In [48], the authors give architecture and implementation regarding ontology based service discovery. They provide the degree of matching basing on weather the service is an exact, partial or no match. While semantic matchmaking enables support for incomplete or semi-structured information, it does not provide any access control over the contextual information. It is interesting to combine both paradigms to provide a delivery service that provides the two important functionalities of semantics based search and dynamic autonomous policies for access control. As a specific case, Jini Network Technology's discovery and registration mechanism will be enhanced to incorporate semantics based autonomous access control and search mechanism.

On the down side, incorporating access control requires a lot of information inflow on behalf of the applications i.e., the applications are required to provide some credentials to match the policies. This process might be slow in case some mobile user just wants to retrieve general information e.g., weather, light conditions, humidity, goods available in the market etc from the context aware system which are not subject to privacy constraints. In such scenarios, policies can be written to grant unhindered access to services that provide such contextual information. Similarly, incorporating ontologies and semantics in service search poses additional computation burden but given the better hit rate of semantics based matching and high computational capabilities of today's computers, it is not a major concern. However, there is a need to carefully define the data structure to represent policies and semantics so that the representation scheme facilitates these mechanisms. RDQL and OWL are the candidate languages for defining semantics as already used in the middleware. For access control, various information description languages such as Policy Description Language, XML Access Control Language [49] are present. Currently survey is being done to find the most suitable description language for our purpose.

8.7 Summary

Context delivery is the final step in the context awareness process starting from, gathering raw data from environment and logical sensors, feature extraction, context synthesis and reasoning to finally delivering context to the interested client or application. It facilitates the context-based interactions by relating the desired contextual information with the interested context consumer. In this chapter, it is identified that a useful context delivery system should be able to deliver a large number of contextual information to a diverse range of clients, should be extensible, capable of defining logical constraints for desired contextual information and able to deal with semi-structured data while upholding the access policies and rules associated with the overall system as well as the individual services it provides. Semantic matchmaking techniques are identified to be useful to achieve the desired characteristics in a context delivery system. While semantic matchmaking utilizes the underlying ontologies, it does not provide any mechanism to implement access control. Autonomous policy based access control is utilized to allow privacy of the contextual information that and integrity of the system as a whole. The result is a comprehensive delivery mechanism with effective matchmaking capabilities and reasonable privacy.

Chapter 9

COMMUNICATION MECHANISMS

9. Managing Distributed Communication Issues in a Context Aware Middleware Infrastructure

9.1 Motivation

Most of the current context aware systems that have been prototyped are limited to providing context at a small physical scale e.g. a campus environment [52], a laboratory, a home [53] etc. At such a scale, the problems of a distributed environment do not present themselves adequately since the sensors are confined to a limited space (allowing easy sensor management), the context synthesis and delivery services run on a single system, system contact points are known [4] and dynamic discovery of system services is not required. In a practical scenario, a fully functional and effective context awareness system will provide context services over a vast stretch of environments ranging from homes, campuses, market places to city blocks and larger precincts. To manage such extended spaces it is necessary to separate the overall environment into smaller logical domains and incorporate a robust coordination and management framework as dictated by following reasons:

- The limitation in the communication range of most sensors makes it necessary for input gathering software components to exist in physical proximity to the sensors. Since sensors are diversely deployed in a ubiquitous environment, multiple input gathering components of a context aware system require coordination and management for data procurement.
- The synthesis of context is a complex process because environment sensors cannot pinpoint users activities in an exact manner and user context has to be interpreted using logical, rule based systems or systems based on Bayesian networks etc. The complexity involve in context synthesis may require special computing devices for efficient performance e.g. a dedicate cluster of workstations set aside for context synthesis of all the entities registered with the system.
- Context is not limited to a confined physical space since a typical user of context provision services is mobile, moving from one domain to another e.g. a commuter traveling from home to office and then to some market place. Employing a single system to manage context synthesis and delivery for a

large environment consisting of many sub-domains can be performance limiting. It is best that the whole environment is segregated into separate logical domains and clone sub-systems handle the steps involved in the context delivery process in each domain individually. These sub-systems effectively appear as small active spaces which can coordinate amongst each other to present the overall active space environment.

In order to carry out these varying specialized tasks and incorporate a considerably large number of hardware and software clients and contributors, a distributed setup becomes inevitable. Objects that interact in a distributed system need to be dealt with in ways that are intrinsically different from objects that interact in a single address space [54].

Dynamic yet seamless integration and interactions among various components of a context aware system becomes significant in order to complement the working of the system and to accomplish a synergized effect. Various techniques such as agents based interactions [55] and broker services [15], [20] have been employed for this purpose.

In [36], the authors have proposed a Context Aware Middleware for Ubiquitous Computing (CAMUS) to address the discussed issues. As the details of CAMUS middleware architecture with respect to context synthesis and data procurement have been adequately discussed in [12], we focus our discussion of CAMUS towards the coordination and managements aspects in the middleware in Sec. 8.2. Sec. 8.3 deals with the design considerations for the coordination framework which leads to a service oriented design as explained in Sec 8.4. A runtime overview of CAMUS is given in Sec. 8.5. The discussion is concluded with an analysis of our framework and future directions in Sec. 8.6.

9.2 Coordination Challenges in CAMUS Infrastructure

The core CAMUS infrastructure consists of four main components that individually handle the tasks of sensor management and data acquisition, feature extraction, context synthesis, storage, and context delivery. Generally middleware infrastructures are designed in modular and layered fashion to enable separation of concerns and provide a smooth flow of operations.

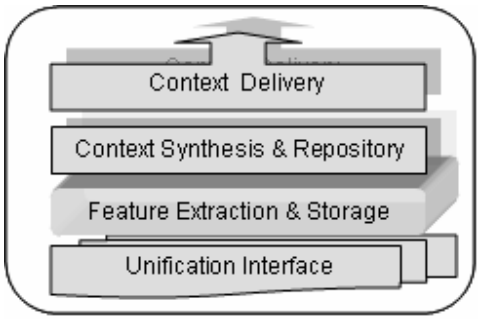


Figure 18: Layered abstraction of CAMUS architecture

In CAMUS architecture, following requirements for coordination arise when functionality is distributed amongst components on specialized systems.

9.2.1 Logical and Physical Separation

In CAMUS, the heterogeneity at the sensor level is handled through the Unification Interface which provides a unified access interface to hardware sensors.

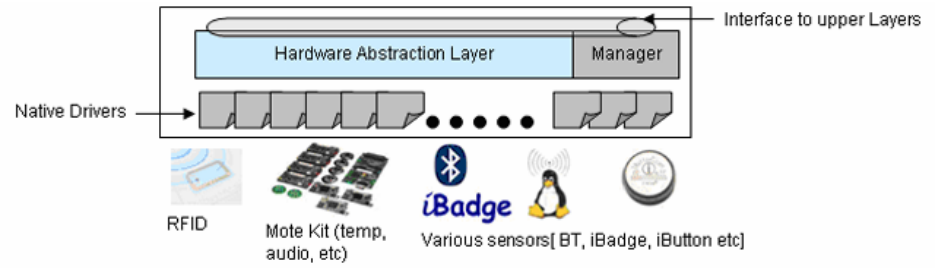


Figure 19: Unification Interface - Native drivers exist for individual sensors which are exposed to upper layers through a hardware abstraction layer, providing a standardized access to all sensors. Real-time

The foreseeable problem that is related to heterogeneous sensors deployed in the environment is the limited communication range of sensors e.g. RFID, infrared, blue-tooth radio enabled sensors. It implies that the software components responsible for managing the sensors (start, stop, reset, discover, register etc) and retrieving measurements of environmental parameters have to be located in proximity to the sensors. However, deploying sensor access components close to sensors is a tricky task due to the distributed nature of sensors. With the distribution of sensors being wide and sparse, deployment of access components close to sensors becomes difficult and we have to adopt a divide and conquer strategy by expending more than one access module to cover the set of sensors exhaustively and combining their results for context generation later on.

In context aware systems, the data retrieved from sensors is initially encapsulated in a data format before being synthesized into context. Since an entity's context is an interpreted result from a collection of features, it cannot be derived from a single sensory source. This constraint necessitates that such intermediate data is placed in storage till adequate information sources contribute and reasonable context can be inferred. In CAMUS, a tuple space [9], [10] is employed as an underlying storage mechanism, for data acquired from sensors, providing a domain-wide persistent space. Various sub-modules for feature extraction and context formation dynamically interact in the middleware by mere flow of objects in and out of the 'feature' tuple space (FTS).

Instead of multiple sensor access modules storing the procured data in a single, central repository, it serves the performance requirements best that multiple localized repositories are used in conjunction with multiple sensors access modules. This not only reduces the communication delays but also the load which would have been incurred in case of a central repository serving multiple concurrent read, write, and search operation requests. Moreover, it enables the storage of spatially related contextual information in separate spaces. Fault tolerance and scalability can be achieved by enabling multiple repositories to synchronize their stored contents in the manner of a distributed file system.

To incorporate reusability and context sharing we use OWL [26] for context modeling. A functional context aware system will be entrusted with context generation of a considerable number of entities at any given time and combined effect of context inference (calculation), repository access and context history storage amounts to a computationally demanding task. Instead of employing a single context synthesis component to interpret context on behalf of all entities present in the system, this task can be logically segmented into compartmentalized domains with each domain responsible for context management within its own boundary e.g. a home domain.

Context Delivery services lie at the top of the CAMUS layered architecture and consist of asynchronous Context Event Service and synchronous Context Query Service. The context delivery services retrieve context for interested applications based on semantic matchmaking techniques [41], [56].

9.2.2 Dynamic System State

Context aware environment is dynamic as the system needs to actively detect, consider and respond to changes in its runtime environment. Systems built for such environments should be able to respond to change both flexibly and automatically. Distributed systems are inherently unstable due to network

unreliability and this instability is increased in context aware systems due to participation of wireless technologies, devices and dynamic network topologies.

In order to ensure synergized operations of various distributed components of the system, it is important that each module has a realistic and loosely consistent view of the system. A mechanism to identify the state of the system and to generate relevant information messages for the interested components is required. A distributed even notification mechanism enables such communications. Note that this event mechanism is different than the one involved in context delivery services as it maintains the internal state of the system as perceived by the cooperating peer components instead of external state which is visible to the context aware applications interacting with the system.

9.2.3 Context Domains

Ubiquitous computing environment is characterized by various domains e.g. home, office, university etc. and context information can be formally modeled to represent a particular domain. To achieve this, individual components need to be affiliated with a specific domain to relate coherent environments and entities, and to confine them within a logical boundary. Figure 20 depicts a two domain scenario.

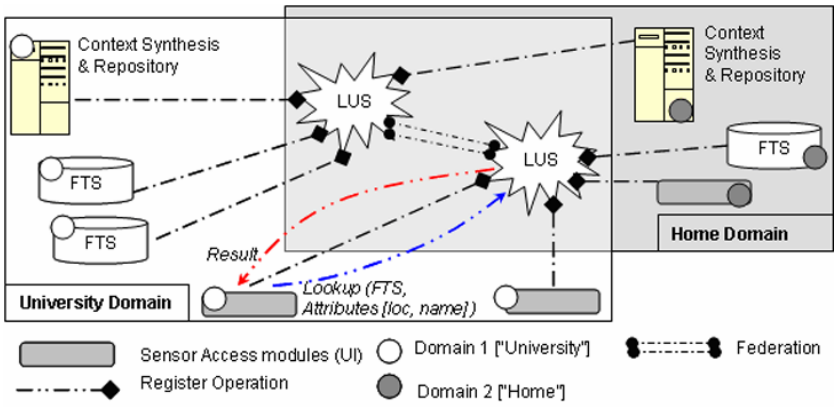


Figure 20: Home and University domains. Individual components join only one domain at a time. It is to be noted that Lookup Services (LUS) are not domain associated but serve discovery and registration services for more than one domain

9.2.4 Internal Security Considerations

Upholding system integrity demands that distributed components can trust each other mutually i.e. requirements of authentication, authorization, and integrity are met. For example, the context synthesis engine needs to be sure that it is taking input from a genuine FTS rather than a rogue process serving dubious information. This requirement pertains to system's internal trust requirement and

not to confidentiality requirements that are demanded by external applications interacting with the system.

9.3 Design Considerations for Coordination Framework

CAMUS has been designed to be deployed not as one system, but as multiple clone systems each responsible for managing context awareness in its allocated zone. For example, in Figure 20 the core components of CAMUS system, namely sensors access modules (Unification interface), feature (intermediate sensory data) repository (FTS) and context synthesis component, are logically bound to one domain. There functionalities are restricted to serving context in their area of allocation.

Various components of the context aware infrastructure will be deployed mostly distributed and physically apart from each other within a given domain. In such a scenario, the foremost issue that presents itself is that of discovery of individual components based on appropriate parameters. Since the whole system can work only when input from complementing components (i.e., sensor access modules to feature extraction to context synthesis and delivery) can be transformed and passed on sequentially by each layer up through the hierarchy, it becomes indispensable to allow individual components to be discoverable by each other. In such a scenario an approach is needed by which all he components can make themselves, their capabilities and their functionality known to others through a discovery and registration mechanism. Components must also be able to search for each other basing on the functionality they require. In general, for a coordination framework, requirements can be listed as follows:

1. Well defined interfaces for communication between components
2. Registration of components capabilities and services
3. Discovery of individual components by peers
4. Searching of components based on required functionality
5. Notifications for important events pertaining to system state
6. System safeguard against temporary failures of individual components
7. Scalability to allow for system growth without reconfiguration

The requirements and design constraints lead us to a service oriented solution presented in the next section. Individual components can be modeled as services which advertise attributes to aid their discovery and expose well defined interfaces for communication.

9.4 Service Oriented Approach to Middleware Coordination

The core CAMUS coordination sub-system is based on Service Oriented Architecture (SOA) [57] where core components are distributed services residing on the network to be published, discovered and invoked by each other. It also allows a software programmer to model programming problems in terms of distributed services offered by components to anyone, anywhere over the network.

The most important benefit achieved by employing the concepts of service-oriented architecture is that service's implementation is separated from its interface. In other words, it separates the "what" from the "how." Service consumers view a service simply as an endpoint that supports a particular request format or contract. Service consumers are not concerned with how the service goes about executing their requests; they expect only that it will. Consumers also expect that their interaction with the service will follow a contract, an agreed-upon interaction between two parties. The way the service executes tasks given to it by service consumers is irrelevant. This aspect of SOA is directly relevant to our design goal of achieving decoupling between the functional aspect of CAMUS (context provision) from the internal system coordination and management.

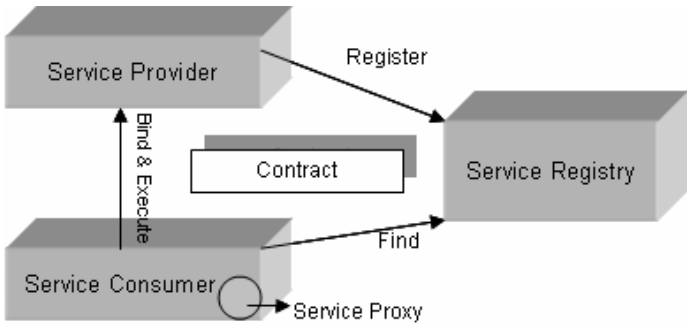


Figure 21: Service oriented operation

The main entities involved in an SOA interaction include a service provider, a service consumer and a registry. A *service consumer* can search for a service provider through the registry. Performance and reliability measurements include a *service contract* that specifies the interaction mechanism between the service and the consumer, a *service lease* specifying the amount of time for which the contract is valid and a *service proxy* acting as a convenience entity for interaction with the service. The service provider supplies a service proxy to the service consumer. The service consumer executes the request by calling an API function on the proxy. The service proxy, shown in Fig. 21, finds a contract and a reference to the

service provider in the registry. It then formats the request message and executes the request on behalf of the consumer. Proxies can improve performance by eliminating network calls altogether by performing some functions locally. Figure 21 shows a typical setup of a service oriented system.

To implement CAMUS architecture based on SOA, several existing technologies were investigated including Web Services, Java RMI, Jini [8], UPnP and Corba etc which are capable of, to one extent or another, satisfying the stated requirements. Web services and Jini are notable implementation of SOA and for reasons stated in Sec. 9.6, Jini technology was found to be the closest match to our requirements. Following figure shows an abstract overview of core CAMUS services interacting with a Jini lookup service component and highlights the independence between modules' implementation and coordination framework.

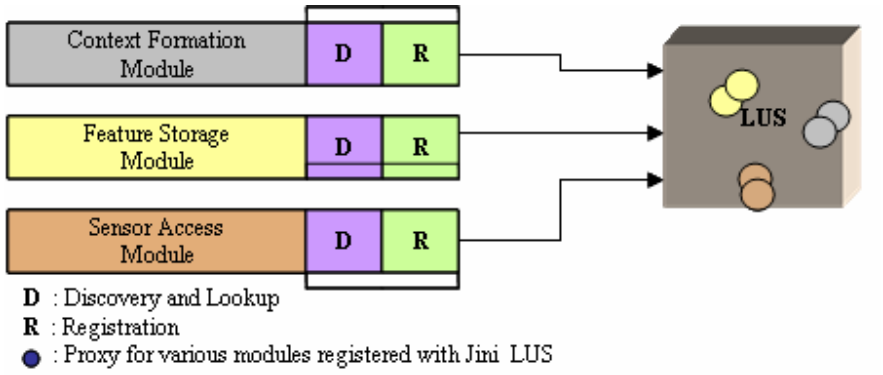


Figure 22: Abstract overview of core CAMUS services using a Jini Lookup service for discovery and registration

9.4.1 Registration Mechanism and Formation of Domains

When a component becomes available, it joins a specific domain by registering the attributes and capabilities it affords along with a downloadable proxy with a *service registry, specifically a Jini Lookup Service* (LUS) [58]. Other components in turn, can discover the service by looking up specific attributes they are interested in. This process can be either direct query or by a notification subscription for a specific module with the registry service. In any case, when the module of interest becomes available, its proxy is downloaded from the registry that exposes interfaces on which remote calls can be invoked for communicating back and forth between components.

The attributes published vary from basic properties such as name, service type, location, and status etc. to specialized capabilities of the module for more specific lookup provision. For instance, an FTS publishes following set of attributes during its individual registration:

```
Entry FTSAttributes [] = new Entry [7];
FTSAttributes [0] = new Location ( "room 312", "floor 3", "engineering building" );
FTSAttributes [1] = new Name ( "RTMM Lab FTS " );
FTSAttributes [2] = new Status ( StatusType.NORMAL );
FTSAttributes [3] = new Domain ( "University" );
FTSAttributes [4] = new FTSCapability ( FTSCapability.READ );
FTSAttributes [5] = new FTSCapability ( FTSCapability.WRITE );
FTSAttributes [6] = new FTSCapability ( FTSCapability.EVENTS );
```

Unification Interface can lookup the feature extraction module if it needs to write some newly gathered features, whereas a *Context Synthesis* module can access it for reading features or for registering events of interest with the FTS. It should be noted that components can specify the location during lookup operations so they can locate other components (FTS in this case) in their proximity. Similarly, queries can be further restricted to domains memberships, e.g. an FTS belonging to 'Home' domain may search only for Unification Interfaces of 'Home' domain.

9.4.2 Distributed Event Notification Mechanism and Leasing

The event notification mechanism has a two-fold purpose. Firstly, it allows distributed components to locate and communicate with each other asynchronously. Second, in conjunction with the leasing mechanism, it helps in maintaining the state of the system as explained later. In addition to the regular subscription notification events, Jini supports Event Mailboxes which can receive event notifications on behalf of their clients and deliver them later upon request. This concept is used in supporting disconnected operations, unexpected failures of components and maintaining history of events of interest.

In a dynamically changing environment where a module can become unavailable at any time due to network delays or failure, it is formidable to find faulty or erroneous components. The soft state maintained using a leasing mechanism resolves this problem by limiting the time for which a module can be registered. Failure to renew the lease by the end of the registration time implies that the module is no longer available. The module can re-register with same attributes when it later becomes available. Both the events for unavailability and availability afterwards are notified to the interested clients. This results in fault tolerance and easy maintenance of the system state and eliminates burden on the client of tracking voids in the communication.

9.4.3 Managing Trust

In a context aware system, where coordination between components is distributed and loosely coupled, maintaining trust between the end points i.e., the

communicating components is of significant importance. The issue is further aggravated when this procedure involves downloaded code such as a remote proxy of the service module. The service consumer needs to be sure that it is getting the correct services that it required and the service itself needs to be sure that unauthorized access to it is prohibited. Client authentication by the service is mostly based on asymmetric key pairs or use of digital certificates which is beyond the scope of this research. As of Jini Network Technology version 2, adequate security mechanisms have been incorporated in the Jini architecture itself to safely deduce that for purposes of large scale deployment, core Jini security components are enough for achieving reasonable security.

9.4.4 Scalability

Scalability refers to the ability of system to manage scale changes towards the larger extremes. In a scenario where system users increase in number or the system has to manage a larger area equipped with a greater number of sensors, change in system configuration becomes inevitable. A usual impact is an increase in the responsibility of system components which may serve as a performance penalty. To meet such a challenge at runtime, one of the solutions is to increase the number of system components handling the tasks that now offer an increased workload. For example, in case of CAMUS, an increase in the number of sensors in a domain can result in deployment of additional sensor access modules and/or feature tuple spaces. This challenge is addressed through federation of services which is the ability of services to be linked together, or federated, into larger groups.

Efficient coordination amongst components requires that the increase in number of components does not hamper the discovery and registration process (which is one of the corner stones of coordination in service oriented architecture). Multiple lookup services handling the task of discovery and registration can be federated and their clients (middleware services) can be configured to query these lookup services in a manner that distributes the system load across a number of lookup services thus avoiding a bottleneck. With the support of federation, the number of components in the system can be safely increased as per demand and the sphere of responsibility of these components can be broadened without affecting performance.

9.5 CAMUS Runtime synopsis

The authors have implemented a coordination framework for CAMUS middleware infrastructure based on Jini. Individual components of CAMUS are deployed as services and their responsibility is limited to logical domains. A Jini service browser provided by IncaX™ [59], which allows to discover and view Jini

services deployed across the network, is used to observe the CAMUS infrastructure at runtime.

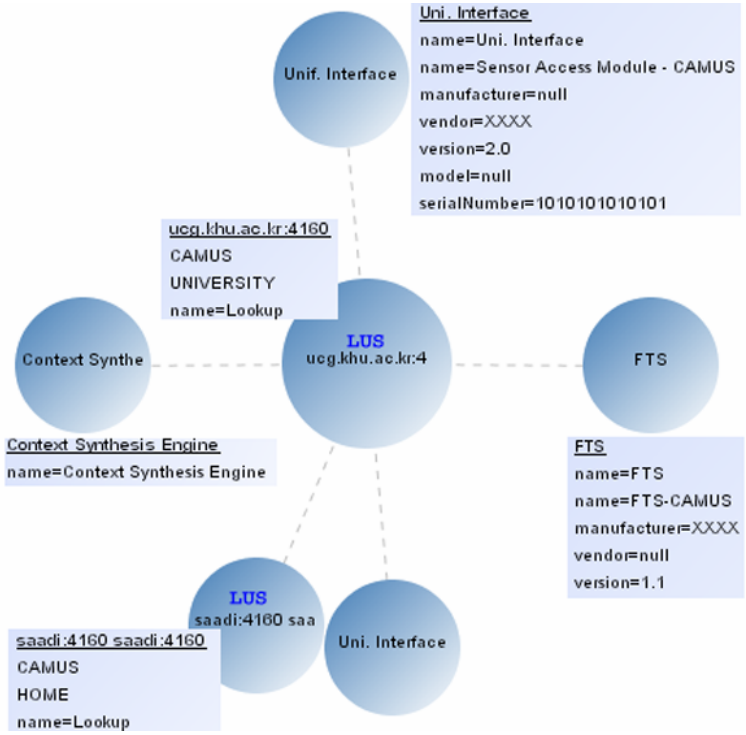


Figure 23: IncaX Service Browser snapshot of CAMUS system deployed in a UNIVERSITY domain. Though components register with a single LUS, lookup services available elsewhere form a federation by registering with each other, making it possible for registered clients of these LUS to broaden their effective known network boundaries. A selection of attributes of executing components is shown next to each service.

Figure 23 and 24 give a snapshot of the CAMUS system in a UNIVERSITY and HOME domain respectively. Components register with a central LUS services to announce their presence and to search for other components. (Detailed description of module attributes has been clipped for purposes of clarity). Though the components are shown to register with a single LUS in the snapshot, in reality their existence is announced to multiple lookup services and lookup services themselves register with each other to form a global community of CAMUS components. This is known as 'federation of services' in Jini terminology and greatly reduces the burden on registering entities in terms of time spent on finding lookup services, lease renewal and searching for other components.

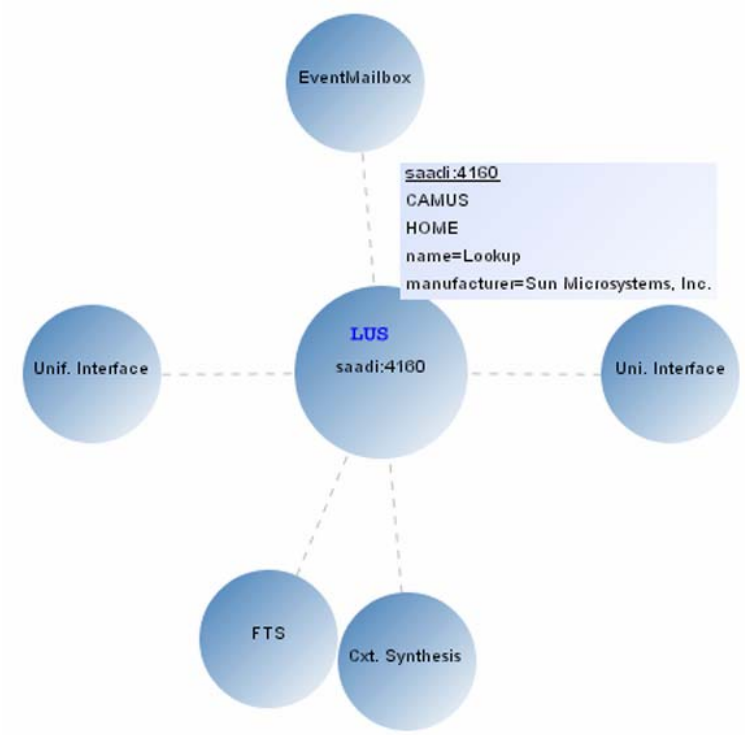


Figure 24: IncaX Service Browser snapshot of CAMUS system in a HOME domain. Jini Event Mail Box service is utilized for notification of asynchronous events.

9.6 Discussion

The functionality of the components in the CAMUS system is independent of the discovery, registration and coordination scheme. A discovery and registration service is attached with each component which enables them to announce and locate each other. These services keep track of all the available service registries (Jini Lookup Service) present in a given domain and facilitate the process of registration and discovery. The only thing required by the components is to provide a remote interface implementation that can be used as a proxy to communicate back with these components. This decoupling of the functionality from the communication scheme leverages flexibility to update or replace the components without affecting the communication infrastructure and vice versa.

The idea behind use of Jini as a discovery and registration technology and use of its event model is to separate the functions of context formation (data gathering, context inference, storage and delivery) from the interaction and management of various components of the system. Ample research has been done in pinpointing the requirement of such a setup and issues arising due to distribution of system

components. Some issues worth putting further effort into include testing system performance under network load conditions, disconnected operation and safeguarding single points of failure (absence of any registration service) by providing backup/fall back registration mechanisms (multicast announce).

As a task for future enhancements, we aim to extend the service search capability available in Jini by identifying and incorporating service attributes related to services in a context aware domain. Another area is the representation scheme for data acquired from the sensors. Most systems utilize a single representation scheme for both elementary sensor data and higher level context by using a variety of schemes such as name-value pairs in XML format [4], object oriented representation [15], black-board systems [60] and ontologies [61]. However, separation of concerns dictates a two-level representation scheme for representing elementary data gathered from the sensors at a lower level and contextual data formed after synthesis at a higher level. We employ the black-board approach (tuple repository) for representing sensor data and ontologies for representing context. The combination of these two representation schemes results in greater flexibility and loose coupling at the sensor level and provision of common understanding and higher semantic representation at the contextual level. Further detail on this issue is beyond the scope of this paper and is reserved for future work.

In the existing solutions for context awareness, a variety of approaches have been tried to enable coordination in the framework. In [4], XML messages over HTTP (TCP/IP) are used between various components for this purpose. This approach is useful since both these protocols are practically ubiquitous. However, the toolkit does not provide a discovery mechanism and the communicating components need to know the exact location of each other in order to interact, hence the coordination is not dynamic and scalable. A number of middleware solutions use Corba based coordination and communication [61], [62] which provides dynamic resource discovery but due to its hard-state registration and lack of support for leasing mechanism, the system behaves reactively instead of proactively to the changing system dynamics. Web services and UDDI based approaches [63] provide a platform-independent mechanism for service description, discovery and usage but it limits the flexibility and expressiveness in the service descriptions to simple attributes URL and string comparisons. Similarly, UPnP provides device-level coordination and low level abstraction but does not support service-level coordination mechanism as required by this approach.

Implementation of our system in Java allows us to utilize the Java activation framework [64] as a performance enhancing measure through which services can

be transferred from main memory to persistent storage when idle, and can be made available as and when they are requested by a client. Activation also aids in fault tolerance when sudden system crashes render services unavailable by restarting the services from the last known good configuration.

Jini was found out to be the closest match to the specific requirements of our system. Particularly, the possibility of querying components by attributes, downloadable proxies and independence from transport protocol were the main support features which were found lagging in other similar technologies such as UPnP. Moreover, the advantages of leasing mature remote and distributed event notification model and event mailboxes provided by Jini can be utilized to full extent in distributed middleware architecture as CAMUS.

9.7 Summary

To summarize, the chapter discusses the requirements related to distributed coordination within context aware middleware infrastructures in terms of component discovery and management, dynamic system state and multiple context domains. A service oriented coordination framework based on Jini Network Technology is discussed to address the issues.

Chapter 10

FUTURE RESEARH ISSUES

10. Some Research Issues for consideration/future work

10.1 Consideration Factors for System Paradigms

Let us have a look at some of the envisioned consequences for research in applications of Ubiquitous Computing systems. In this section we try to give taxonomy of typical Ubicomp applications. The relevant coordinates for the taxonomy are depicted in Figure 25, and we believe that these scales are important for characterizing Ubicomp applications on a qualitative level, which in turn affect the middleware system paradigm. This is especially useful for us in the construction of any real application or testbed system.

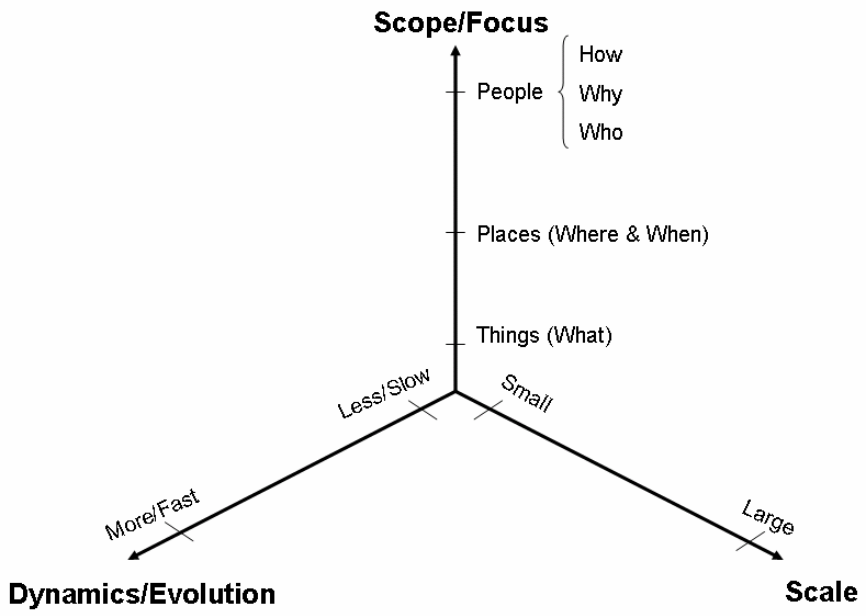


Figure 25: Three Dimensions of Ubicomp Systems

- **System Focus or Scope**

Applications are designed to cover some application domain. A common classification in this area is "People, Places, and Things" originating from the HP

Cooltown research project [65]. Usually, one can identify a central focus of an application related to one or two of these topics. For example, a tourist guide is focused on giving information to a user. This is the main focus. However, it gives information about places; therefore, it has places as a secondary focus. But to be successful it has to bring information about places to users in an appropriate way (for the user), hence, "people" is the main focus ("subject") and places are the secondary focus ("objects") of the system.

•System Scale

Scale is an important factor for system and application design. Is a system designed for a rather small place and covers a rather small range (e.g. a Smart Home application) or does it scale up to something like a public space (e.g., a train station) or even to cover something larger like a city or a country (geographical scale)? Obviously, this has some fundamental influence on system design. Another sub-dimension is the number of participants in a system. A Smart Home application might have to handle a couple of dozens objects and subjects (the fridge, the home entertainment center, the garage door, etc. Here the requirements for system design are completely different. For example for a smaller application it might be perfectly sufficient to have no common infrastructure for data exchange and communication, and an ad-hoc network and broad- or multicast communication might suffice. On a larger scale this approach is not feasible due to the traffic this style of communication implies.

• System Dynamics/Evolution

This is the most driving factor for Ubicomp applications. Ubicomp is inherently "mobility driven" and "***dynamic***"; People are moving, as well as things, and sometimes even the places are moving around (e.g., cars or trains). This imposes serious design challenges for system, as well as for application designers: Associations between communicating parties are volatile, so might be data/information (especially by third parties), data is belonging to someone else; everything is - in a sense - "floating around." And the consequences would be:

- System and application design will have to take dynamics (on every level) into account
- Bindings will be casual and volatile, due to mobility.
- Systems have to be built along with a changing society, not against.

Also a consequence of the highly dynamic nature of Ubicomp applications is the need for ***evolvable systems***. It can be expected that such systems must be open, flexible, and extensible. The reason is obvious; we cannot make assumptions about the capabilities, the operating systems, the installed software, or the style of communication devices have. On the other hand, we cannot make assumptions about what services can be used, what QoS can be expected, or what "version" of

service is "installed". Therefore, it is mandatory to build systems in an open, flexible, and extensible fashion. Especially in highly dynamic systems with many different users this requirement is extremely important. Systems might be more long-lived than devices the users bring along; therefore, they have to be extensible to new standards and requirements. On the other hand, users cannot make assumptions about the services and infrastructure they will find at different places. Their devices must be capable of adapting to the situation found at hand. For instance, service discovery is important. Not every public place has the same infrastructure and services "installed". Every party has to have the capability to evolve and adapt.

A mandatory requirement for Ubicomp systems and applications is that the actual functionality is determined at "runtime" opposed to determine the functionality at "compile time," i.e., the moment the device is deployed. Here an important requirement is to have an effective and efficient data-management in term of retrieval of relevant information, like relevant services, relevant data (e.g., for Context determination) and other information needed depending on the actual application.

10.2 Privacy and Security Issues

Security issues and privacy concerns must be addressed in CAMUS. The dynamism and ubiquity of the pervasive computing paradigm raise new issues for information security and user privacy. This is an especially difficult case because an important feature of context aware computing is to share information across users and systems. Therefore, we need a well-established privacy mechanism that can balance context sharing and information security for context aware computing [11], [66].

10.3 Programming Toolkits for the development of context-aware applications

A Toolkit layer will provide graphical application interfaces for developing Context-aware Application as well as specifying Privacy Policy, Application Policy for dealing with uncertainty, etc.

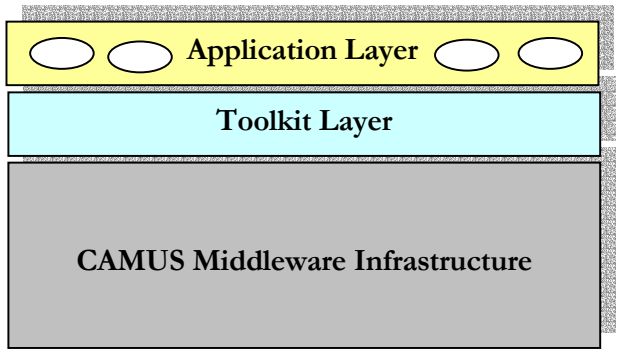


Figure 26: Toolkit approach

10.4 Future Research for Context Ontology

The idea of exploring Web Ontology Language to model context creates opportunities, while also opens up several open issues for further study. Here we discuss three main issues: 1) Model Transformation for Machine Learning, 2) Extending OWL for quantitative features, and 3) Privacy Control.

Model Transformation for Machine Learning. As we mentioned, in addition to logic reasoning, machine learning is another feasible approach to derive high-level context from low-level context. In fact, a large amount of context-aware tasks (e.g., a home-care service uses predicted user behaviors to optimize inhabitant comfort) require machine learning mechanisms. Unlike logic reasoning that can be directly supported by ontology models, machine learning requires training data in form of different dedicated models (e.g., Markov chains, feature vectors, etc). Therefore, it requires further study on how to transform ontology based context model to dedicated models for specific learning mechanisms. One direction is to study the model requirements of general machine learning algorithms (e.g., Markov chains, Bayesian learning, neural networks, reinforcement leaning, etc), and provide an algorithm-specific model transformation mechanisms.

Extending OWL for quantitative features. Through the above study in context modeling, we can see that OWL and entailed description logic are necessary for modeling general concepts of context. However, the limitation of description logic makes OWL insufficient for modeling quantitative features of context such as order, quantity, time, quality of information, or uncertainty/probabilities. Unfortunately, capturing such features is critical to certain tasks such as data fusion dealing with uncertain or incomplete sensor context. Therefore, we need study how to extend OWL models with capabilities to express quantitative concepts, thereby enabling temporal reasoning and probabilistic reasoning in a formal approach.

Privacy Control. The dynamism and ubiquity of the pervasive computing paradigm raise important challenges for information security and privacy. Moreover, Semantic Web as a whole is largely conceived as an open network to share information, and can not support any privacy control mechanism. Therefore, we require the context model to provide a working privacy mechanism that can balance knowledge sharing and information privacy for context aware computing.

10.5 Challenges in Context Reasoning

The decision on what kind of logic or learning mechanism to use depends on not only the power and expressivity of the logic, but also other issues like performance, tractability and decidability. According to the feature of the tasks, different learning mechanisms can be used in a hybrid manner. For example, learning based on Bayesian networks and explicit rules written in probabilistic or fuzzy logic are useful in different scenarios. Bayesian networks are useful for learning the probability distributions of events and enable reasoning about causal relationships between observations and the system state. They, however, must be trained before they can be used, but because they are flexible and can be retrained easily, they can adapt to changing circumstances. Probabilistic logic is useful when we have precise knowledge of events' probabilities; fuzzy logic is useful when we want to represent imprecise notions. Both probabilistic and fuzzy logic are useful in scenarios where getting data to train a Bayesian network is difficult. This is especially true in the area of security. Beside this first challenge in choosing the most relevant reasoning mechanism for each high level context, middleware system must also facilitate the ability to plug in new reasoning mechanisms. The use of fixed APIs between the reasoning engines and other software entities using them appears to be a feasible solution for adding different reasoning engines easily, and it is currently being applied in CAMUS. So now a crucial task for future is to improve the performance of the system by revising the query algorithm. Another work is creating more wrappers for various kinds of Reasoning Engines to provide more help to the developers.

10.6 Context Delivery – Semantics-based Autonomous Access Control and Matchmaking

The idea behind establishing context aware systems is to gather and dissipate information relevant to a user in order to enrich the software functionality and facilitate adaptive behavior. With the vast expanse of information gathered from the physical and computational environment, there is a need for an efficient delivery mechanism to filter out unrelated information and communicate the relevant contextual information to its respective clients while maintaining the

privacy of the information and integrity of the system. For a context delivery system to be feasible, the system should be able to deliver a vast set of context to a variety of diverse applications, should be extensible in order to incorporate new applications and new contextual information as the context aware system matures, have capability to define logical constrains for proper matching of the context advertisements and context queries, and cater for semi-structured data in case the context aware applications that need to utilize contextual information does not provide exact specifications of the required context. These requirements suggest that the context delivery mechanism and corresponding representation scheme for advertising and querying context must be flexible to incorporate a variety of applications, yet expressive enough to avoid ambiguity in delivering desired contextual information. In [41], a framework is presented which uses semantic web for matchmaking in e-commerce. Similar concepts can be extended for the provision of context delivery in the context aware middleware systems. We can view the context consumers and context producers as the interacting parties where context producers advertise the context they produce and context consumers query and utilize the corresponding contextual information.

Various context aware systems are directing efforts now to incorporate privacy and security techniques in the pervasive environments. The foremost concern is to define access control policies that can be suitable in a pervasive environment. The major concerns are: the policies need to be dynamic in nature; the granularity of control to information needs to be identified; how to cope with changing policies basing on the context at run-time; how can the clients trust the system while providing personal information to it for access and validation. Some of these issues can be overcome if autonomic access control techniques are employed in context delivery.

The important aspect of this research is to identify the format for context advertisement and query. Existential and universal quantifiers and Boolean expressions supported by the representation language will be dealt with in detail to use them for the purpose of specifying constraints. Inheritance, specialization, generalization and other relationships can exist between various concepts in ontology. These relationships can be used to process queries in which specific or exact details are not provided by the subscriber. Existing context aware middleware systems lack in the capability of assigning logical meanings to input and output parameters of delivery process and do not support semi-structured data. In the approach presented, these deficiencies are overcome by employing the semantic matchmaking with DAML+OIL as the representation language. Syntax and semantics for context announcement and context query are being analyzed.

10.7 Autonomic Sensing Agents - Scope, Vision, Challenges

As difficult it is to foresee what will happen in the communications area in the far future, there are some constant initiatives that drive technology throughout the years. The vision for new communication paradigms is formed by the fact that technology and consequently communications are mainly driven by the urge to make our lives better.

Throughout human evolution, human lives become better with the use of tools. The more intelligent the tools, the greater the degrees of freedom we possess to move towards better health, knowledge, productivity and safety. In this meeting there was a general consensus on the fact that the intelligence of communication networks should be increased. Taking a step forward, new communication paradigms should focus on the development of intelligent, self-cognitive networks that no longer act as a means to simply propagate information from one machine to the other, but become a living partner of individual and societal activities. In this context, it is foreseen that we will move towards the development of cognitive situated networks that will play a significant role in person- and society-focused communications. One key area in this field is the development of cognitive sensor networks that will be able to bridge the physical world with the digital world [68], [69] and to promote health [70], [71], [72] safety [71], productivity and knowledge through communication of the network with the environment.

Cognitive sensor networks will be built with the deployment of large numbers of autonomous sensor and actuator nodes. Using a large number of specialized sensors and actuators in a dense network we will be able to acquire localized and situated information of certain metrics gathered from the physical and/or digital environment. These networks will use this collection of situated measurements in order to recognize and control certain events in the physical and/or the digital world, for promoting health, safety, communications and knowledge. Thus, sensor networks will be used to monitor environmental phenomena and identify emerging threats, or they can be used to monitor the functionality of physical/digital/cyber networks and be able to foresee and cope with emerging problems. In other words, the use of sensing and measuring equipment in a physical and/or digital network can increase its self-cognition. On the other hand, with the use of specialized sensors, sensor networks will be able to monitor the behavior of living organisms, both in their individual and societal activities [72], or they will be able to communicate with the senses and minds of living organisms to expand the frontiers of human perception [68]. Finally, cognitive sensor networks will enable interaction with the environment in a new and visionary way, **forming civilization-focused networks**. This vision involves

communicating planned human activities to the environment, so that the impact of these activities can be evaluated in advance [73].

In cognitive sensor networks, the nodes are only significant to a certain level. The intelligence of such networks does not lay on the nodes themselves, which have very limited recourses and capabilities, but in the size and complexity of the network. Sensor and actuator nodes can be considered as simple context agents in a complex autonomous network. Since the goal is to form an intelligent network that will act as a living partner of individual and societal activities, cognitive sensor networks will be responsible of coping with a variety of diverse applications with contradictory characteristics. Therefore, the architecture of the network cannot be universal for all applications. In fact, the architecture of the network could be seen as a programming language that is used to solve a problem [74]. The applications will evolve around the user and the network will evolve around the applications. This will lead to a new concept of network, where the resources are deployed only where are needed and where are really necessary.

10.7.1 Objectives – Research themes

There are several challenges that have to be addressed in order to pave our way from the current technology to the vision of cognitive sensorized societies. These challenges include several issues integrating interdisciplinary areas of research, from implementing the nodes to building the intelligence of the network.

- **Implementation**

One of the main challenges is to be able to implement sensor and actuator nodes that will be able to support the concept of bridging the physical and the digital world. These nodes have to reach sub-mm levels, integrate silicon to neuron interfaces [68], [75] and maximize their computational power, memory and survivability, in very small dimensions, and with extended mobility capabilities.
Moreover, the communication [76], [77], [78], [79] and architectural issues [73], [74], [80] of the nodes have to be re-examined. The network deployment method should also be reconsidered; paving the way for "Spray Computers" that form randomly situated networks [81].

- **Cross- or non- layered architectures**

As mentioned before, the power of cognitive sensor networks lies in the networks themselves, rather than in the nodes. Since the intelligence of the network must be enhanced to cope with diverse goals, the layered structure of the protocol stack must be reconsidered. Cross-layer architectures or software-based non-layered architectures must be sought to achieve coordination between nodes and optimization. Moreover, the goals of optimization have to be reconsidered, in order to address other issues than the traditional goals to increase the

bandwidth of the network [80], or to achieve connectivity [82]. Special emphasis will be devoted to decentralized optimization strategies, based on game theoretical approaches, as a general tool to find out the communication strategies as a function of the operative environment, rather than imposing the communication structures a-priori [77]. New hybrid solutions should be analyzed for enabling a wider range of interactions among the user, the environment and the network. To describe this novel architectural model and assess the capabilities of the network, it will be of primary importance to establish the fundamental limits through the formulation of a *network* information theory that will encompass the partial results available nowadays [76], [77].

- **Autonomic Situated Communications**

There is a general consensus on the need to create a self-organizing communication network concept which should be technology independent, task- and knowledge- driven, scalable, and based on cross-layer or non-layer approaches. A major challenge in this vision is to create autonomous intelligent networks, which will additionally take into account the location of its nodes [83]. The goal is to create a network "self-ware" based on universal and fine-grained multiplexing of numerous policies, rules and events that is done autonomously, but facilitates desired behavior of groups of situated network elements [73].

- **Distributed and Context Based Applications**

Cognitive sensor networks that connect the physical and the digital world can promote health, safety, and communications from a microscopic to a macroscopic level in a seamless, secure and meaningful way. Key challenges in this context are to define how we can produce distributed communications [81], how we can provide access to healthcare everywhere, how we can identify individuals and objects in time and space [84], how we can apply semantic tagging on sensor data [84], [85], how to distribute management to achieve goals [84], [86]. Key aspects to be considered are principles to ensure the security and trustworthiness of distributed applications [87], [88], and the effects and interactions which these communication paradigms will have on human and social aspects [69], [83] in relation to the sensorized societies. Above all, strong ethical issues arise concerning security and privacy in human societies and the level of control in natural environments [68].

Chapter 11

IMPLEMENTATION DETAILS

11. Implementation Details

11.1 Implementation progress

In the first prototype, for sensor layer, we finished the HAL Manager to manage all the sensors. WLAN driver and WLAN Feature Extraction Agent have been done to get the location of PDA. We also made the driver and FEA for Berkely Mote to get the light value and light level of the environment.

The Feature Tuple Space is also finished, based on IBM Tuple space.

The core functions of Feature to Context Mapping module have been done. Based on the main mapping module Mapping Manager, a client Mapping service has been built to map from the location and environment features to some location and environment contexts. Moreover, this client mapping module can infer the location of user base on the location of PDA. To do this inference, a location reasoning engine was built based on Jena generic rule reasoner. The Context Reasoning Manager and the unified API for all kinds reasoning engines are also done.

To store the context data, we finished the context repository backed by MySQL. The repository manager module provides the context query over multi models.

There are three context aggregators have done so far : a general context aggregator which provides some general contexts, and two testing aggregator – the Agent Aggregator providing some agent information and agent location, the Environment Aggregator providing some environment aspects.

To deliver the context, we introduce a simple version of Context Delivery service, which does not contain the Match Making Algorithm.

Jini is used to communicate among the modules in Camus.

11.2 System workflow

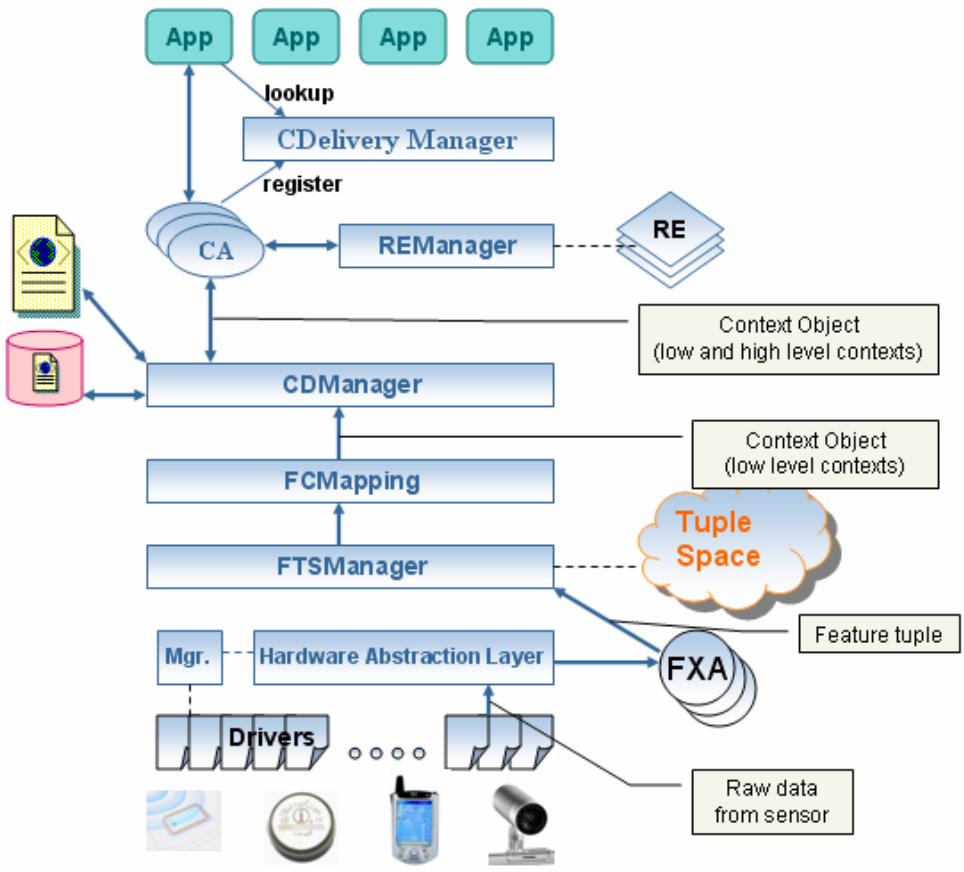


Figure 27: The system workflow

11.2.1 From sensor data to feature tuple

The context of users and environment is gotten by the sensor drivers, then supplied to Feature Extraction Agents as raw sensor data. Feature Extraction Agents fuses the sensed data and extract the valueable data. Those data will be converted into feature tuple format, and be inserted into Feature Tuple Space.

To insert the feature tuples into Feature Tuple Space, the Feature Extraction Agents use the write API of Feature Tuple Space.

11.2.2 From feature tuple to context markup

Knowing which features it needs, the mapping modules registers to Feature Tuple Space to be notified about the existing of certain new feature tuples.

When new feature tuples come, the Feature Tuple Space sends notify messages together with the coming feature tuples. Those feature tuples will then be converted into context marlup format, and saved into context repository.

11.2.3 From low-level context to high-level context

Here comes the crucial job of a context-aware system. Some reasoning engines will be invoked to infer some new facts from the facts that are existed in the context repository. Each reasoning engine will works over a group of context data and related ontologies, which acts as a knowledge base. By this way, system developers can use any reasoning method they want to get the high-level context data they need. The new infered facts will also be inserted into the context repository, and be treated as normal facts in the next reasoning time.

11.2.4 Context information is ready to be delivered

There are certain Context Aggregators to prepare the context data which is needed by certain kinds of application. Context Aggregators take care of building the RDQL queries for each required data and calling Context Repository Manager module to process those queries. After receiving the data from repository, based on each situation, Context Aggregators will decide whether they should invoke some reasoning engines to get more data.

Each Context Aggregator has to register to Context Delivery service about its provided services (the core of that registration is the information which Context Aggregators can supply).

11.2.5 Acquiring the context information

When an application needs some context information, it will send some messages to the Context Delivery service to lookup for the Aggregator which can provide those contexts.

Context Delivery service use semantic match-making mechanism to find the appropriate Context Aggregators.

After that, the application acts the role of context consumer, getting the context data directly from the Context Aggregators which act as Context Providers.

.

11.3 UML design

11.3.1 Unified Sensing Framework

11.3.1.1 Class Diagram

a. Feature Extraction through Unification Interface

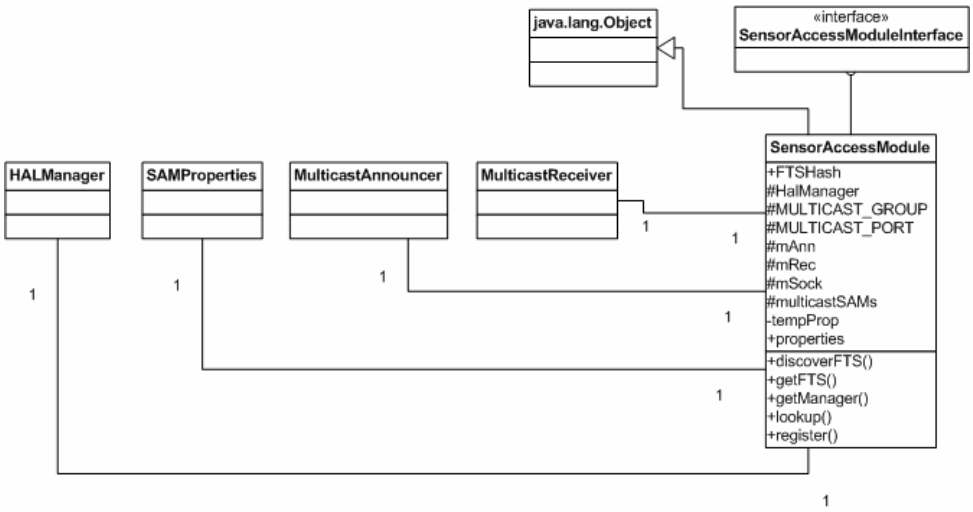


Figure 28: Sensor Access Module - The lowest layer of CAMUS Architecture interfacing with environment sensors.

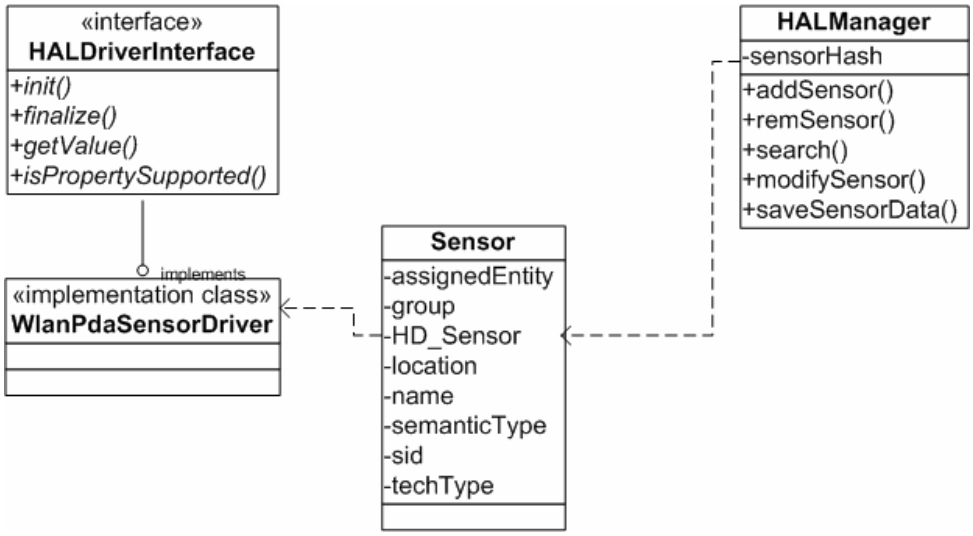


Figure 29: Hardware Abstraction Layer Manager (HALManager), Sensor, and Sensor Driver (WlanPdaSensorDriver in this case)

The above two diagrams show the main classes involved in achieving unified access mechanisms and are collectively called as Unification Interface. These classes make up a Sensor Access Module. The HALManager is an object

contained by the SensorAccessModule. It manages the attached sensors and provides housekeeping methods of adding, removing and modifying sensors. The sensors are represented in software by a Sensor class which contains, apart form some defining properties a reference to a sensor driver. In the above diagram, an example has been shown of an implementation class for WlanPdaSensorDriver.

b. Feature Tuple Space

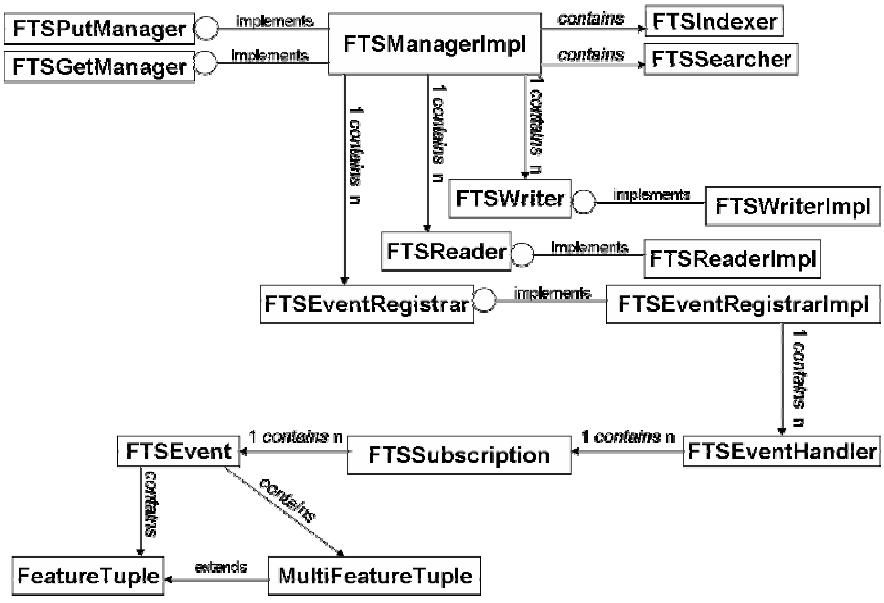


Figure 30: Class diagrams of FeatureTupleSpace related classes

The Feature Tuple Space module expose two interfaces to the clients. FTSPutManager interface is used to perform write-privilege operations such as write, delete, update on the tuple space whereas FTSGetManager interface is used to perform read-privilege operations such as read, take, and scan on the tuple space. This interface is also used for event registration by the upper layers. FTSManagerImpl implements and manages these interfaces from the inside. With every write request, an instance of FTSWriter is created which handles the operations on behalf of the writing clients. FTSReader facilitates the read operations by implementing the related methods on the tuple space. FTSEventRegistrar module keeps track of the registration for events from the interested clients and generates notifications to their specified callbacks. When a registration request arrives, an instance of FTSEventHandler is created which continuously listens to the events taking place in the tuple space and notifies whenever it sees an event in which it's client has registered interest. The data

structure used to represent a feature in the tuple space is encapsulated in the FeatureTuple class instance. To handle similar features from multiple sensors which provide redundancy, MultiFeatureTuple class instances are used. FTSEvent and FTSSubscription comprise the data structure used to store the event types and subscriptions. Descriptive class diagrams of each of these are shown below:

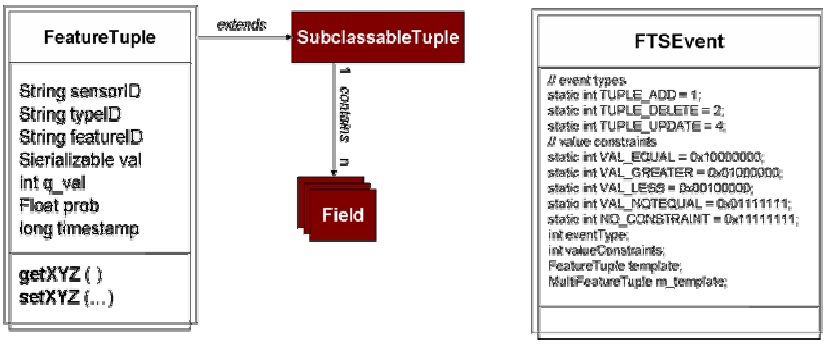


Figure 31: Class diagrams of FeatureTuple and FTSEvent Classes

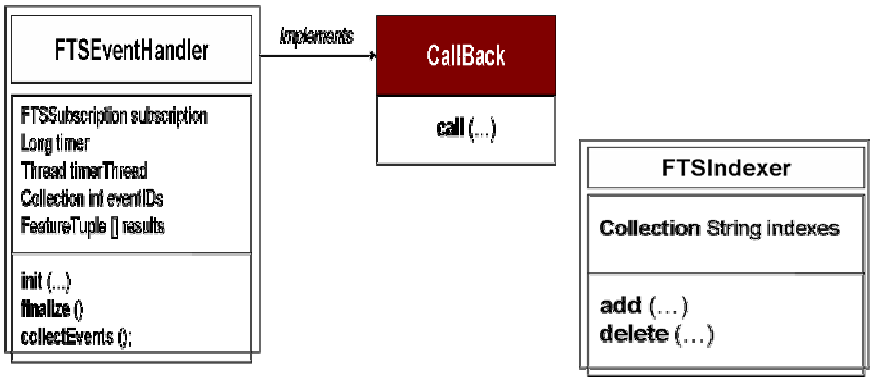


Figure 32: Class diagrams of FTSEventHandler and FTSIndexer Classes

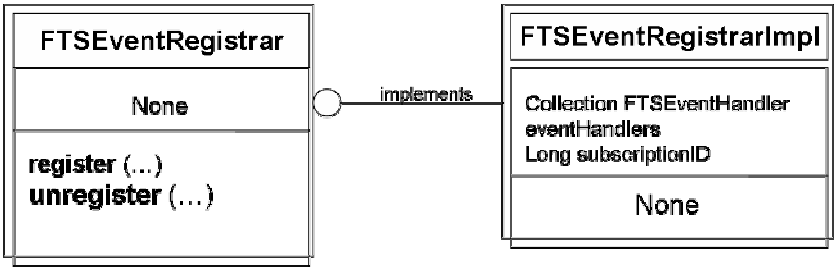


Figure 33: Class diagrams of FTSEventRegistrar interface and its implementation class

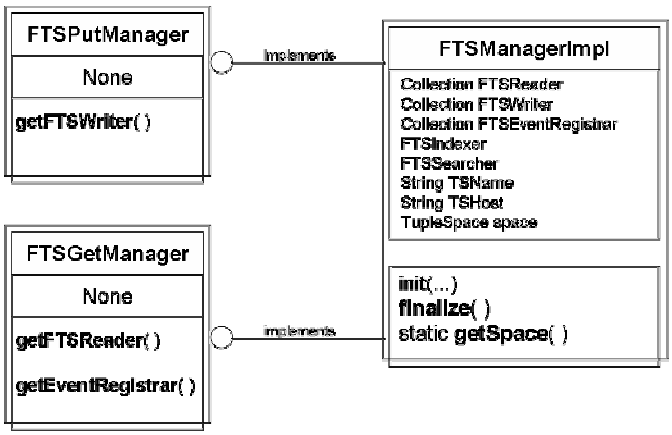


Figure 34: Class diagram of FTSManager and two implementing classes namely: FTSPutManager and FTSGetManager

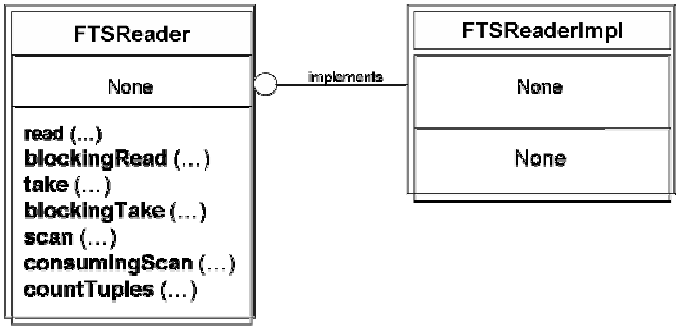


Figure 35: Class diagram of FTSReader interface and its implementation class FTSReaderImpl

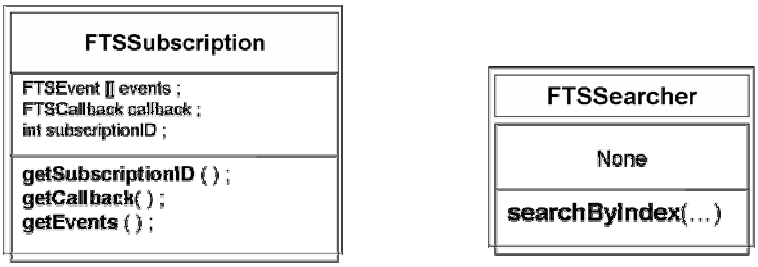


Figure 36: Class diagram of FTSSubscription and FTSSearcher classes

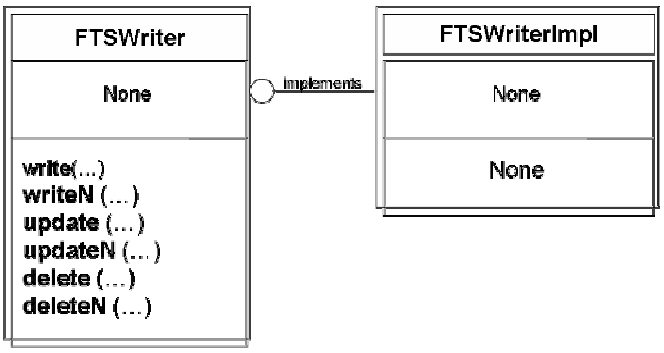


Figure 37: Class diagram of FTSWriter interface and its implementation class FTSWriterImpl

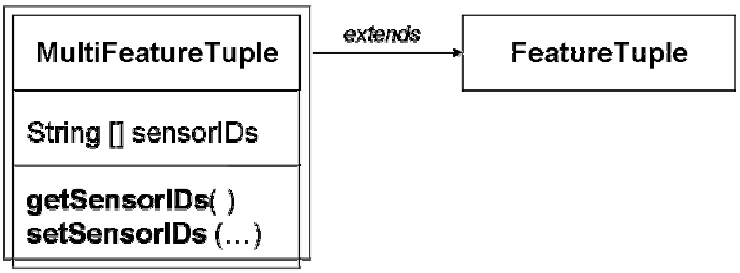


Figure 38: Class digram of MultiFeatureTuple showing that this class is an extension of FeatureTuple class

c. Feature to Context Mapping

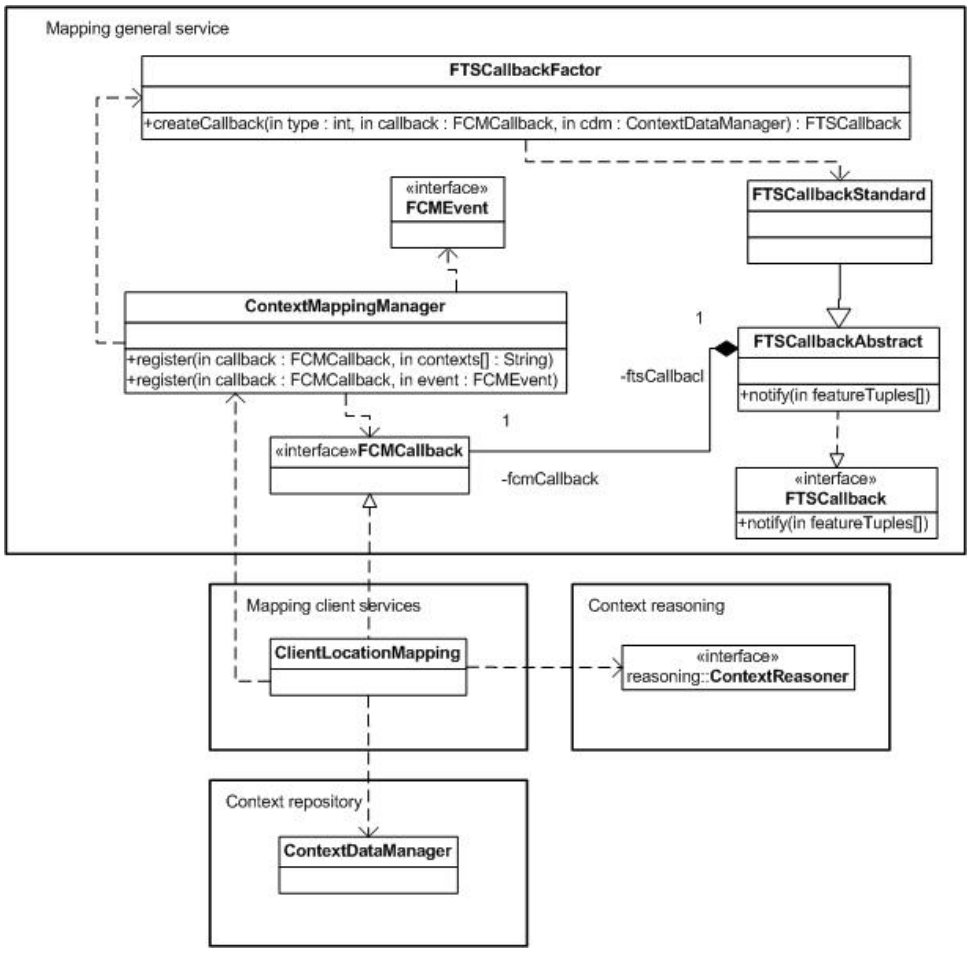


Figure 39: Mapping layer

The main class of Context mapping layer – ContextMappingManager – is a service which allows other services to register for some contexts. When other services register for contexts, they have to provide the callback handler which implements the FCMCallback interface, and the require contexts, or an context event which implements the FCMEvent interface.

Another class, FTSCallbackFactory, manages the built-in Callback classes. The callback classes implement the FTSCallback interface provided by Feature Tuple Space. One of the callback classes is FTSCallbackAbstract, which taking care of converting the feature tuple to context markup by default. All other callback handlers should inherit this class.

A mapping client can implement the FCMCallback interface. It registers itself to the ContextMappingManager. It can also utilize a ContextReasoner to infer some high-level contexts while mapping, and it calls the methods of ContextDataManager service to insert new contexts into context repository.

11.3.1.2 Sequence Diagram

a. Feature Extraction through Unification Interface

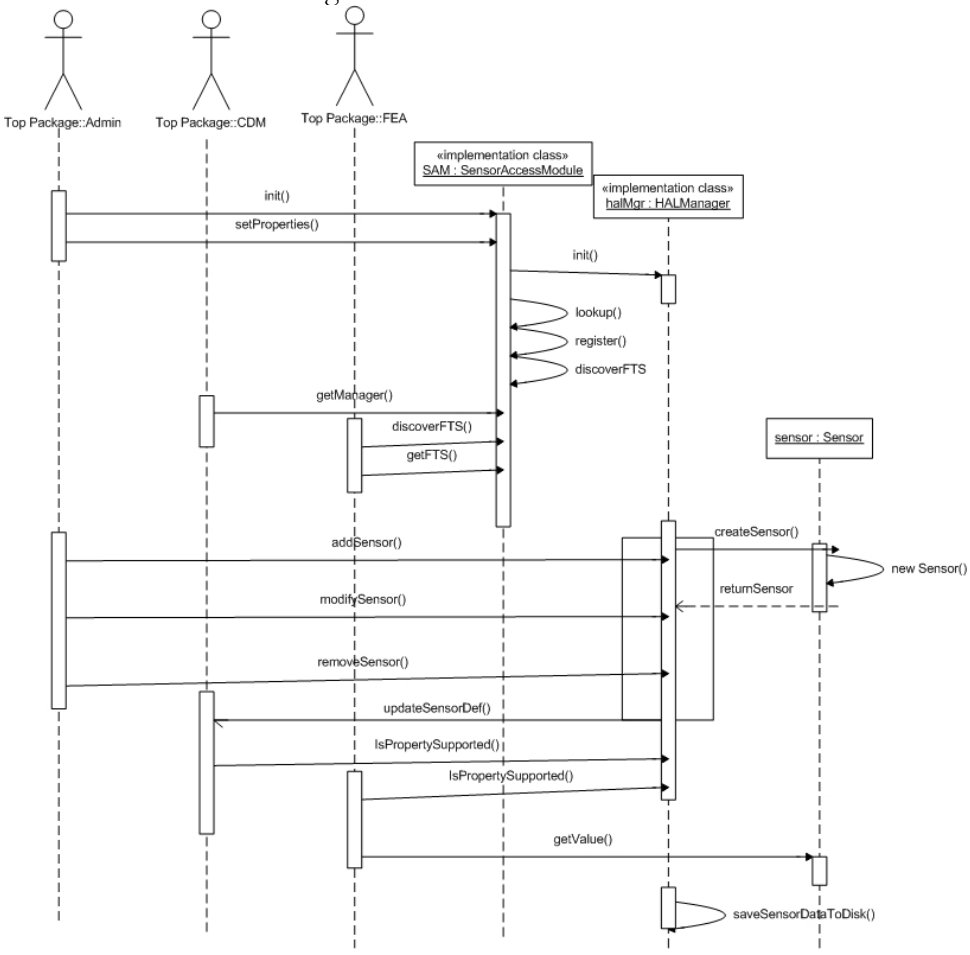


Figure 40: Diagram showing sequence of operations within the SensorAccessModule

The above figure shows sequence of operations that generally take place in the SensorAccessModule and what steps are involved in interaction of SensorAccessModule with the CDM and Feature Extraction Agents. The amdin/user can initialize and finalize the module and also add, remove and modify the sensors. Feature Extraction Agents access features through the

HALManager of the SAM module and store these features in the Feature Tuple Space. It is the task of the SAM module to discover Feature Tuple Spaces and make them available to the Feature Extraction Agents. The diagram can be studied in detail for purposes of clarity.

b. Feature Tuple Space

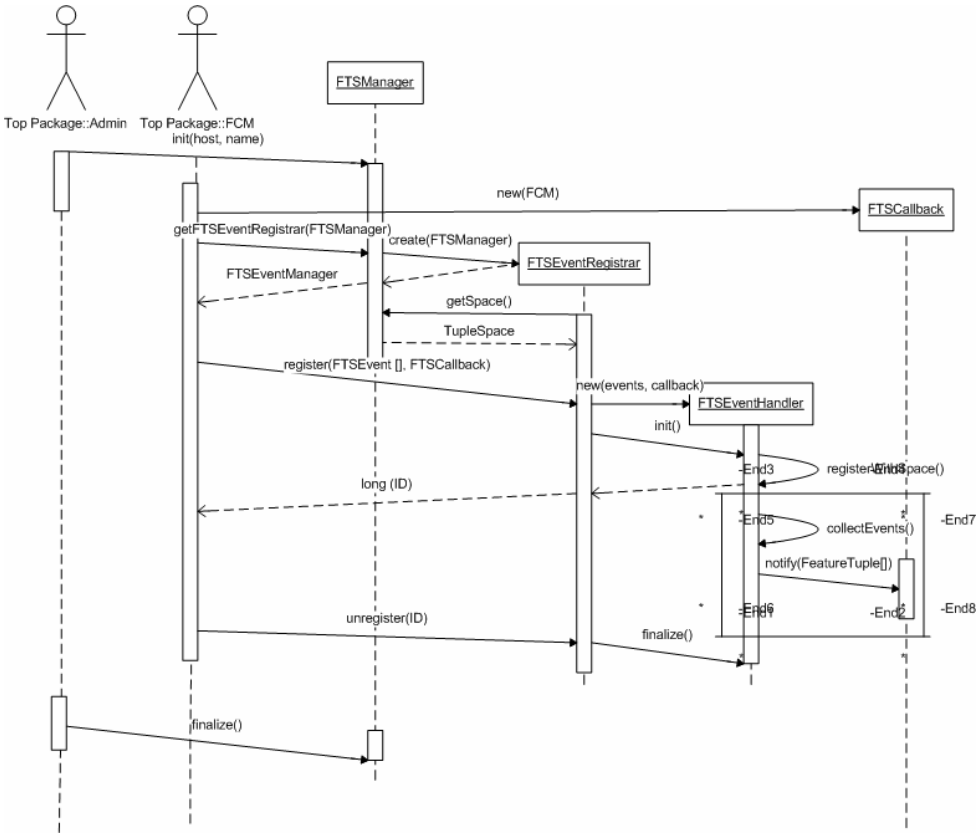


Figure 41: Registration for an event with FTSManager

The above figure shows, apart from steps involved in initialization of the FTSManager, the registration for an event with the FTSManager. A reference is obtained for FTSManager by the interested entity which calls 'register' method on FTSEventRegistrar object. The FTSEventRegistrar first gets a reference to the TupleSpace object from the FTSManager. The FTSEventRegistrar object creates a new FTSEventHandler which collects events that arise in the TupleSpace (write, delete, update etc) and filter the events that were requested by the interested entity. An object is provided by the interested entity which implements the callback interface to receive the event notifications. The notify method of this call

back object is invoked whenever an event of interest is generated. The process continues until the registration expires or the FTSManager is halted.

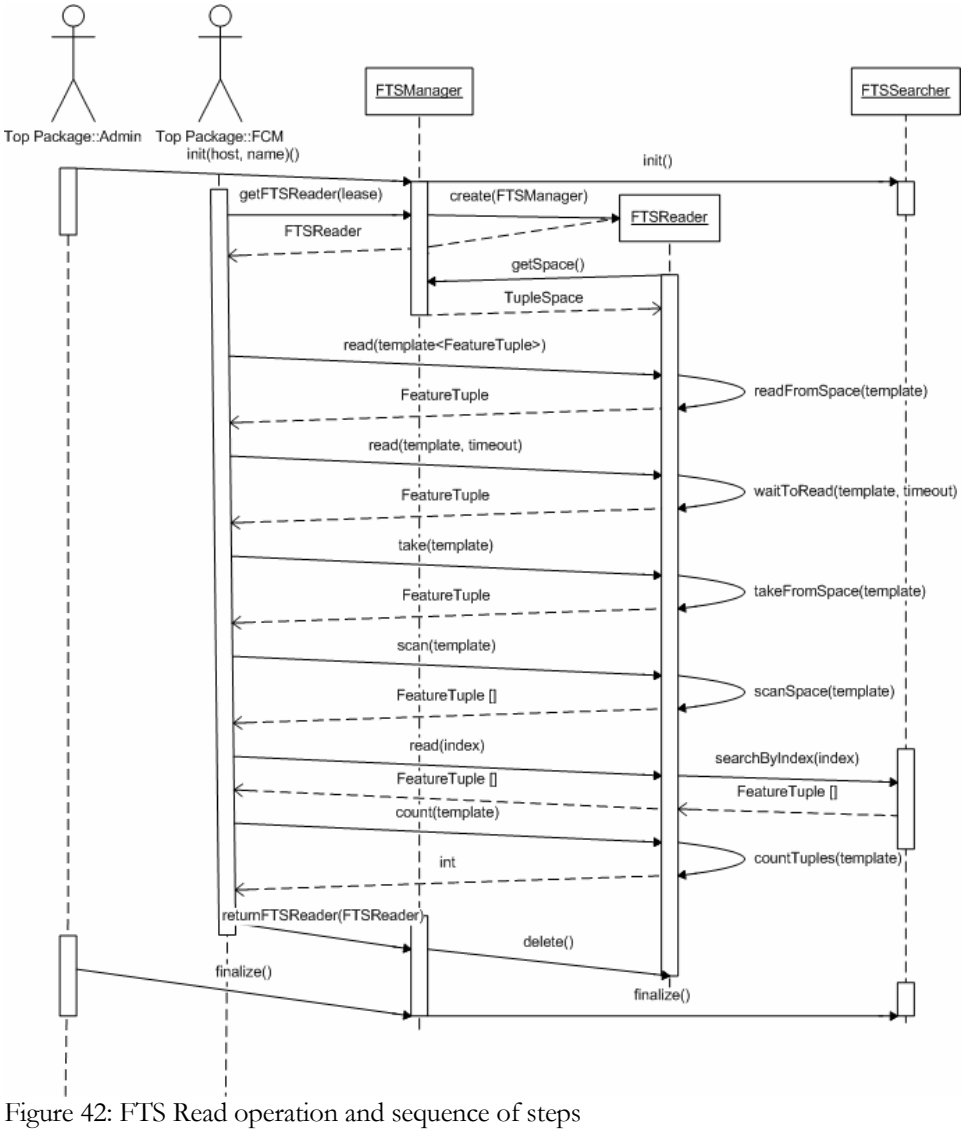


Figure 42: FTS Read operation and sequence of steps

In the above figure, it is shown that an FTSManager creates an FTSReader for each request from the FeatureToContextMapping object to read feature tuples from the FTS. The FTSReader first gets a reference to the TupleSpace object from the FTSManager. A combination of count, read and take methods are used to access the features from the FTS. To search for a particular feature, FTSManager also holds an FTSSearcher object.

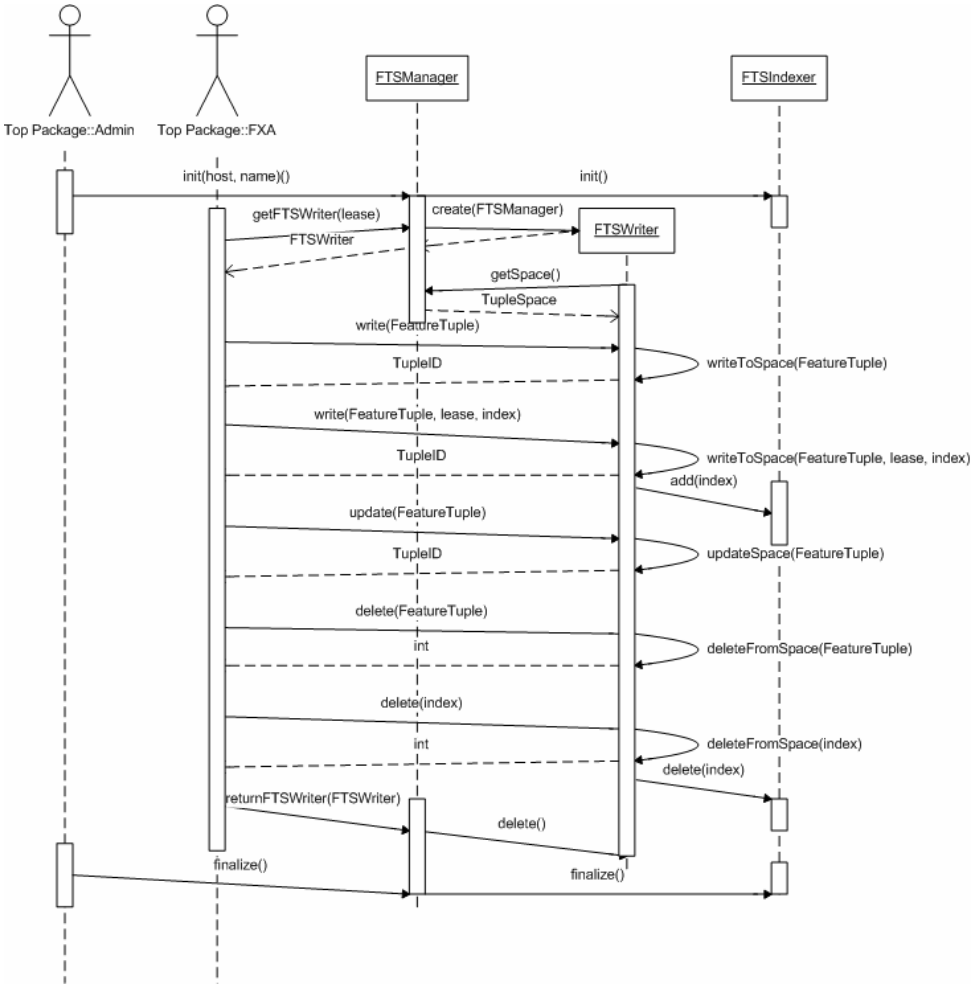


Figure 43: FTS Write operation and sequence of steps

In the above figure, it is shown that an FTSManager creates an FTSWriter for each request from the FeatureExtractionAgent object to write feature tuples to the FTS. The FTSWriter first gets a reference to the TupleSpace object from the FTSManager. A combination of write, delete, and update methods are used to add, delete or modify the features in the FTS. FTSIndexer maintains indexes provided during write operations for quick search and retrieval later on.

c. Feature to Context Mapping

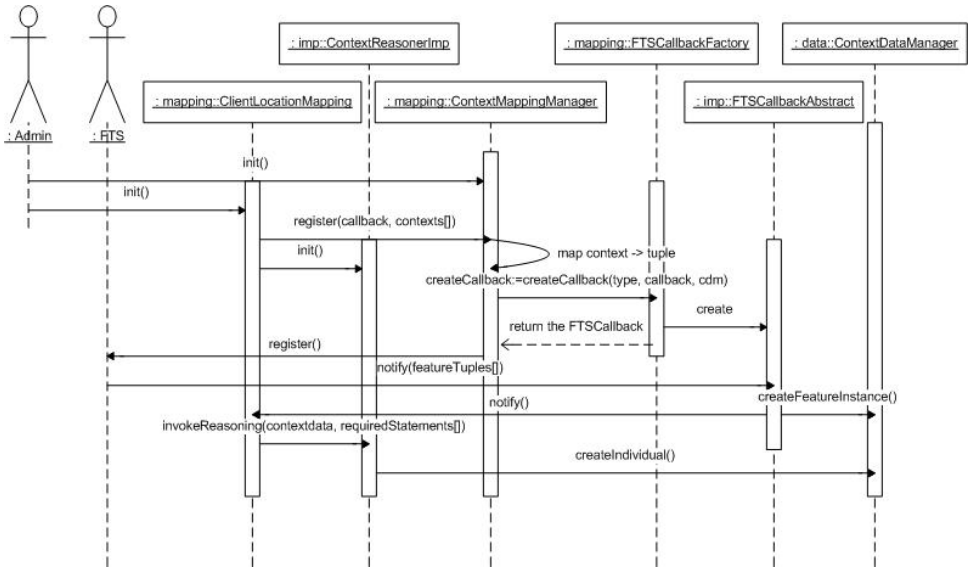


Figure 44: Mapping

To do the mapping, first, the ContextMappingManager service should be started. Then one or more client mapping services will be run. A client mapping service will register to ContextMappingManager for some contexts, and the ContextMappingManager itself will map those required contexts into feature tuples, call FTSCallbackFactory to create the callback handler, then register to the Feature Tuple Space.

When receiving the notification (which contains some new feature tuples) from Feature Tuple Space, the handler maps the new feature tuples into context markups, inserts them into context repository, and notify the client mapping service for new context event. Then the client mapping service can call the reasoner to do some reasoning for some composit contexts.

11.3.2 Context Repository

11.3.2.1 Class Diagram

Data classes

The following class diagram shows the basic data structures used to represent all kinds of context data in the context repository.

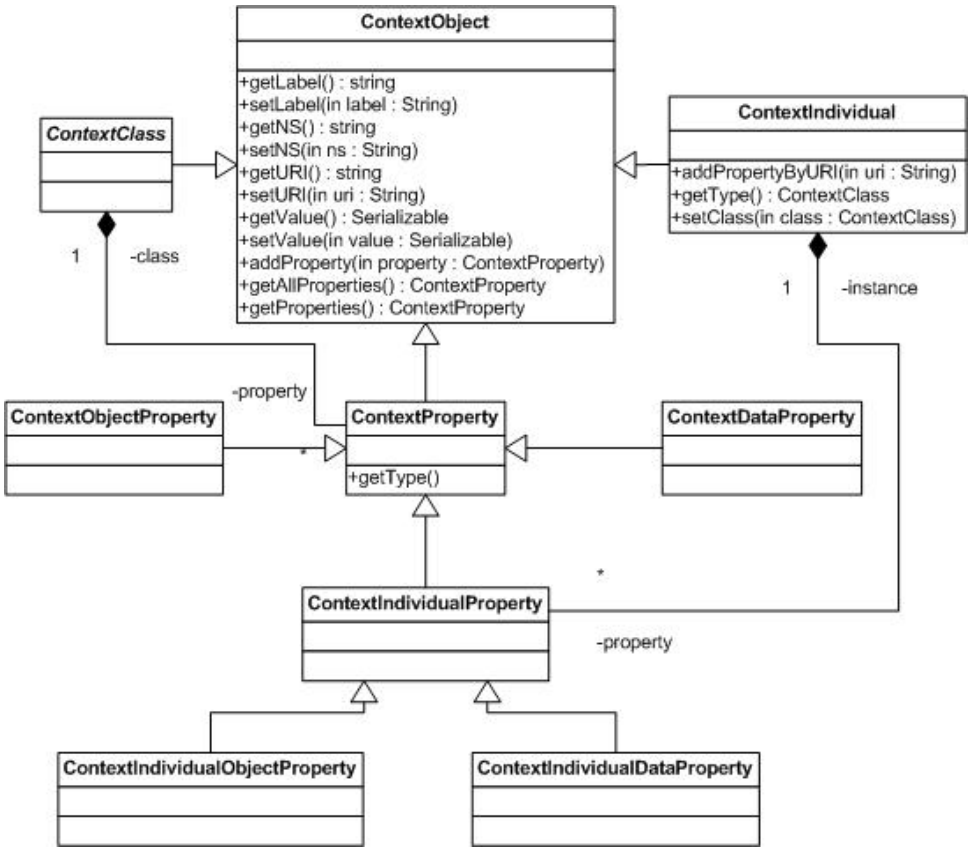


Figure 45: Basic data structure for context repository

All the context can be considered as a ContextIndividual, which have type of a ContextClass. For example, a user can be an Individual of Class Person. ContextIndividual and ContextClass are generalized to Context Object.
A Context Class can have many ContextProperty, and a ContextIndividual can have many ContextIndividualProperty, which is a specialized class of ContextProperty.

A Property can be a datatype property, which is a literal and has the getType method returning a RDFDatatype, or an object property, which is a ContextIndividual and has the getType method returning a ContextClass.

Control Classes

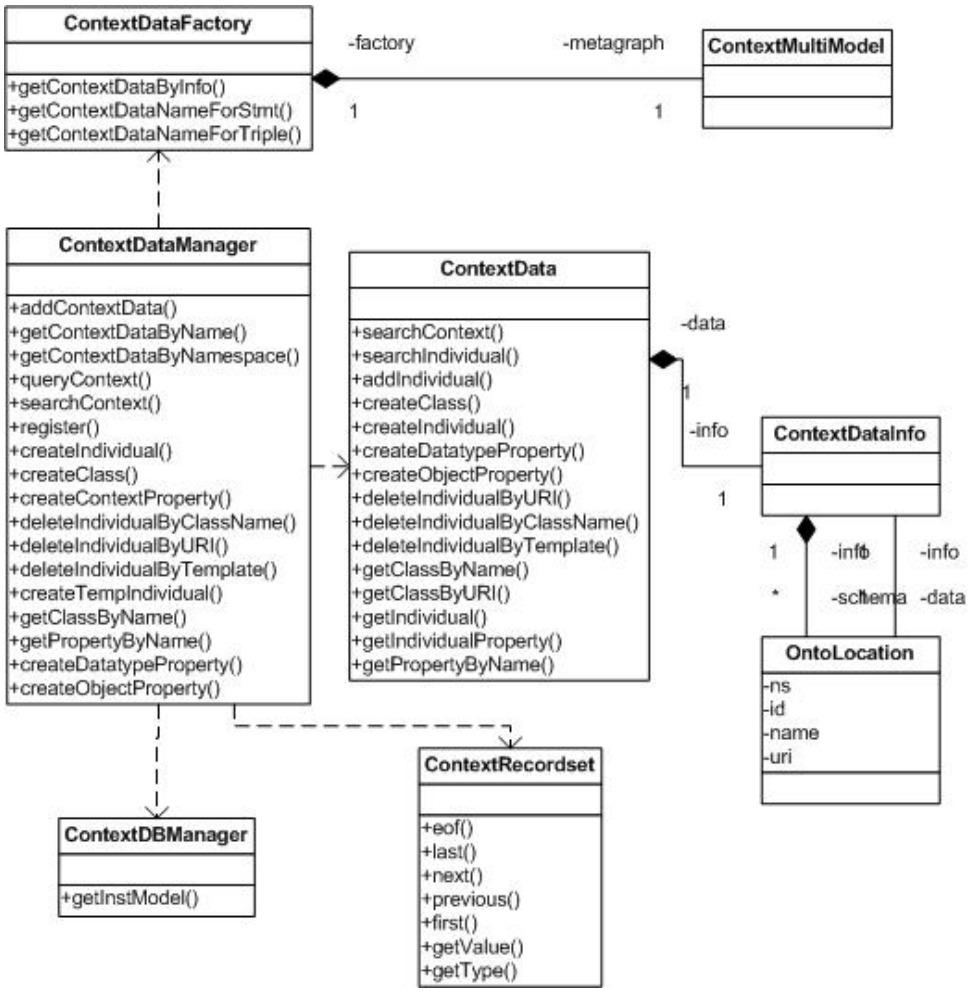


Figure 46: Control classes for context repository

The main class of the Context Repository is the ContextDataManager. This class provides all the needed methods to handle the contexts. Contexts can be handle in format of ContextIndividual, ContextClass, ContextProperty, … ContextDataManager manages many ContextData, which can be considered as some groups of context data. Each ContextData has its own info, including name, data location (namespace and URI), location of the schemas (each ContextData can have zero to many schemas).

ContextDataManager also contains the queryContext() method, which executes the RDQL queries over context repository and return the result in format of ContextRecorset – a data structure which is a simple imitation of Recordset in SQL.

Another similar method is searchContext() which searchs for the context data matches the provided template.

The ContextDBManager helps handle the database.

11.3.2.2 Sequence Diagram

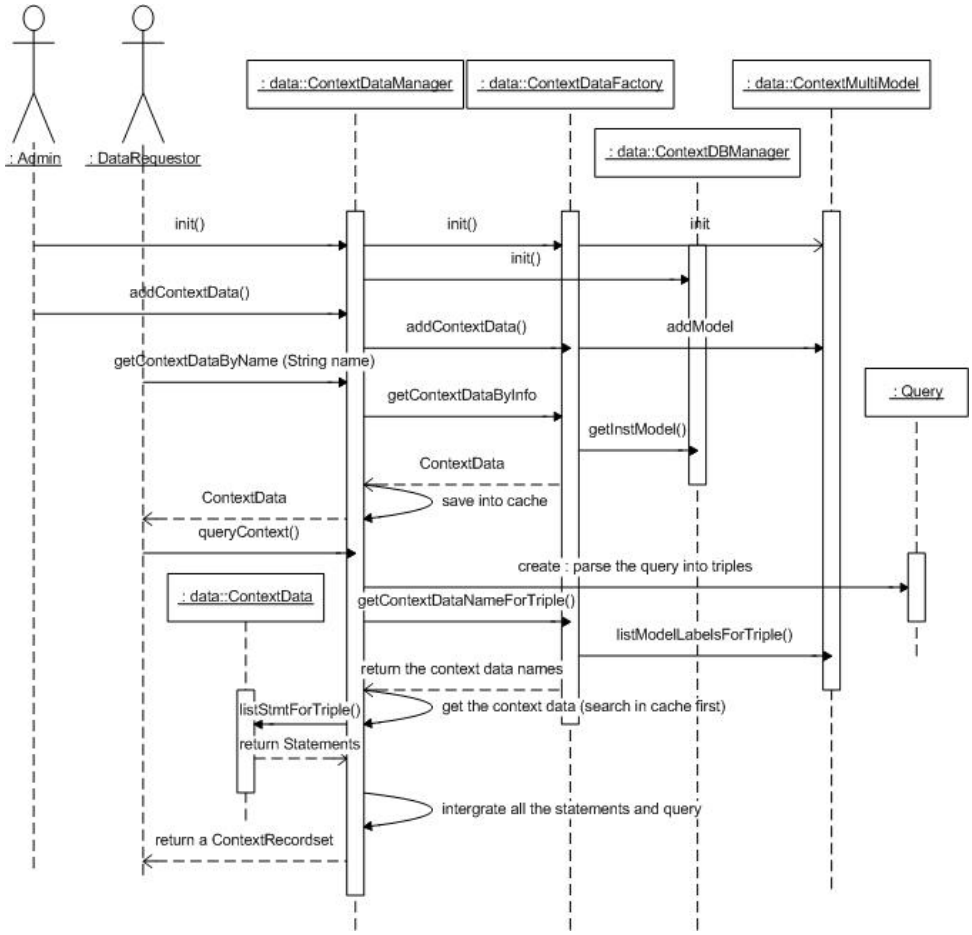


Figure 47: Data adittion by administrator

After starting the ContextDataManager service, the admin can add some context datas into the repository. Everytime a new context data is added, the meta graph of all the context datas will be updated.

Those context datas then can be retrieved by name or namespace.

The above sequence diagram also illustrates the multi-domain query mechanism which has been already describe in section 5.3.

11.3.3 CAMUS Reasoning Engines

11.3.3.1 Class Diagram

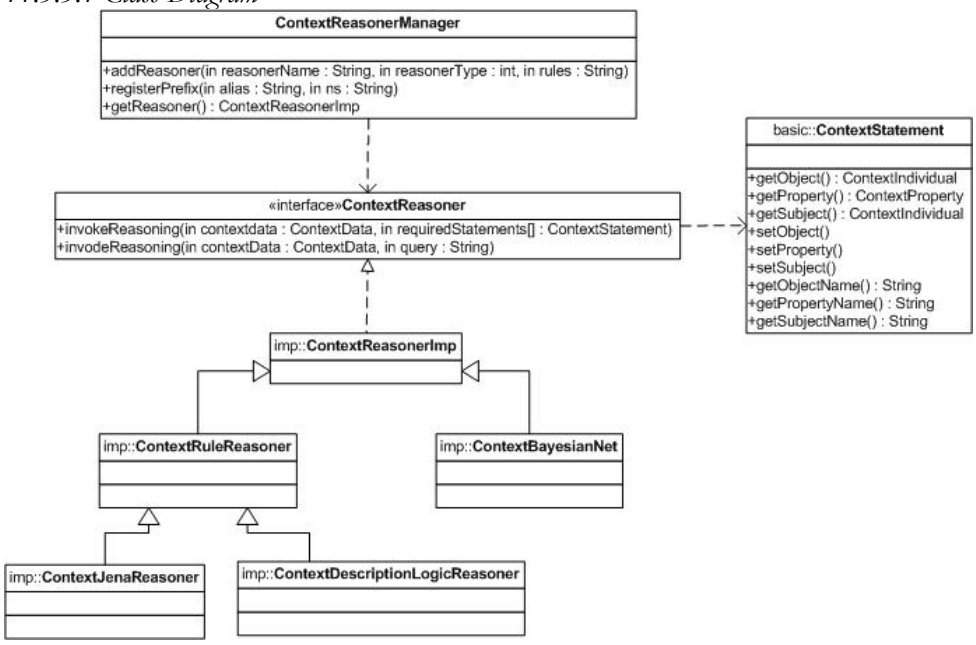


Figure 48: Reasoning Engine

The ContextReasonerManager manages all the reasoners in the system through a unified interface : Context Reasoner. All the reasoners will implement this interface. For each type of reasoner, one or more abstract class will be create to handle the basic functions of that reasoner type, help simplifying the job of creating a new reasoner.

The most common method of a reasoner is invokeReasoning, which do reasoning over a provided ContextData, to infer some statements, or infer the new contexts required by a RDQL query.

11.3.3.2 Sequence Diagram

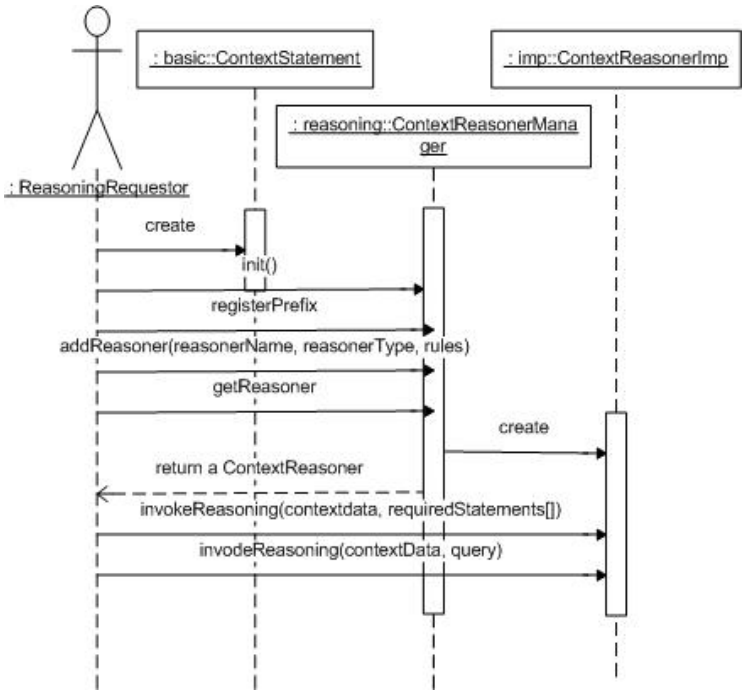


Figure 49: Addition of Reasoner

Any service like ContextAggregator, or client mapping service, can call the ContextReasonerManager service to add a new Reasoner, get an existing Reasoner, then do reasoning by calling the invokeReasoning of the reasoner.

This sequence diagram is used for all kinds of rule base reasoning engines. Hence it provides the developers maximum simplicity in handling the reasoning task in a ubiquitous system. They just have to compose the rule sets and decide the context data which be used, and the middleware take care of all other work from creating the reasoner to inserting the new infered data into the repository.

For other kinds of reasoning engines, the same simple mechanisms are being developed. Generally speaking, all of them will follow this sequence diagram, with a small different in the addReasoner() method.

11.3.4 Context Provision/Aggregation

11.3.4.1 Class Diagram

Since context aggregator is a service, it requires CARegistration class which is responsible for its registration with the delivery service so that the interested clients can discover it when required. While ContextAggreator is the basic interface for all kinds of aggregator implementations and is implemented by

ContextAggregatorImp, which provides generic methods of addition, deletion and searching of context (related to that aggregator) from the context repository, a storage location for all contextual data.

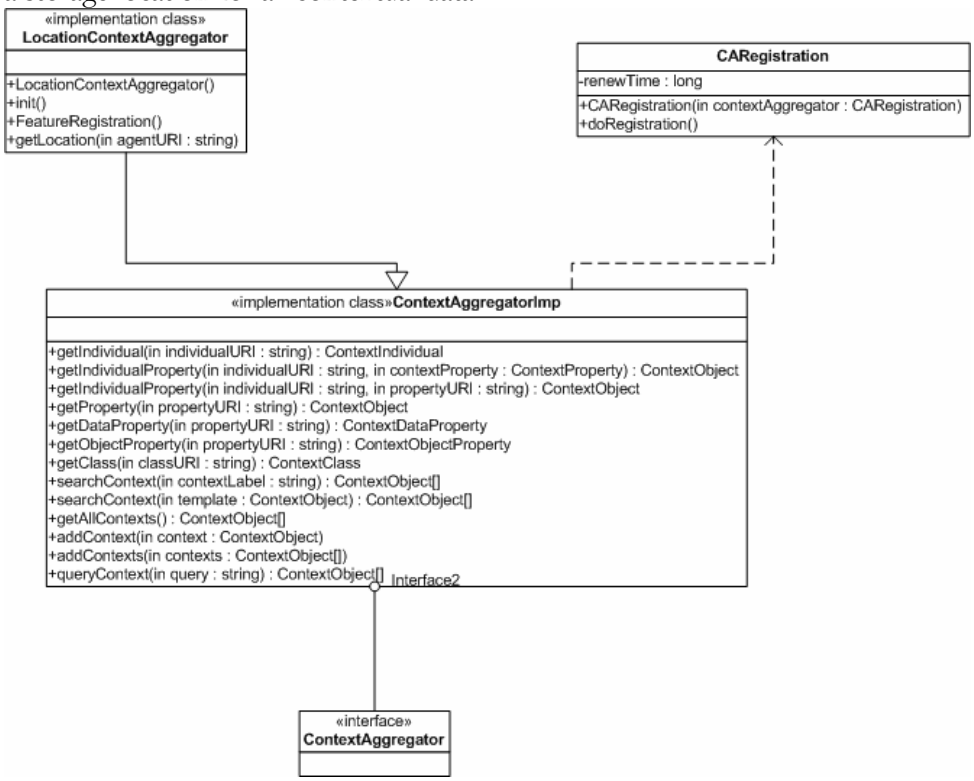


Figure 50: Class diagram for context aggregator module

Also, simple LocationContextAggregator is present here in diagram with functionality of locating a certain user managed by this aggregator.

11.3.4.2 Sequence Diagram

CA Initialization

Simple initialization procedure is mentioned below with aggregator registering it when initiated.

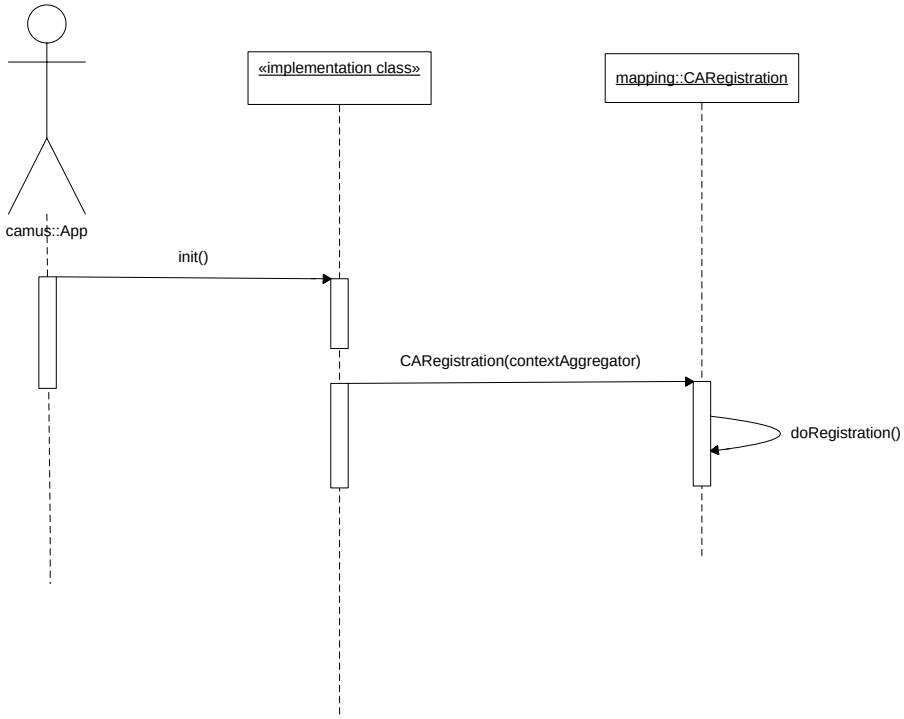


Figure 51: Context Aggregator Registration

11.3.5 Context Delivery Services

11.3.5.1 Class Diagram

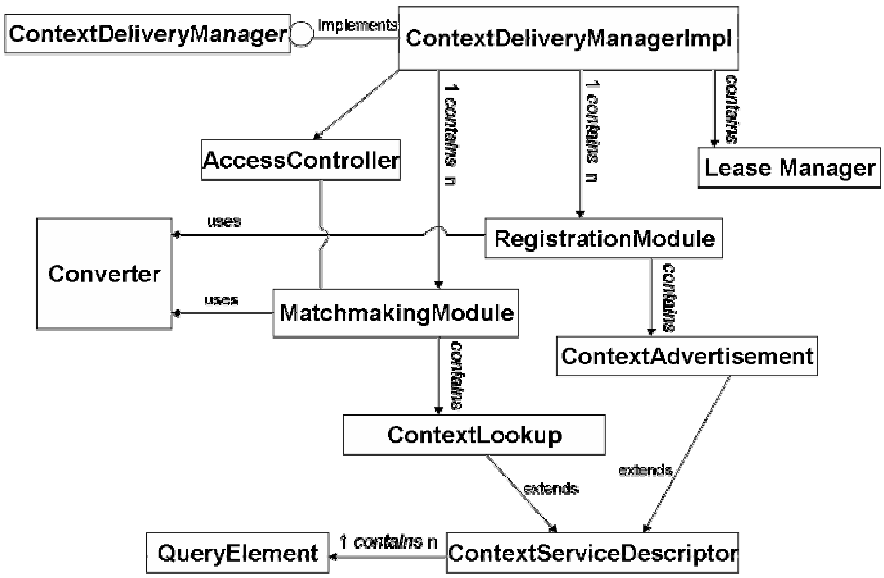


Figure 52: Context Delivery Module's Class Diagrams showing class relationship

ContextDeliveryManager is the interface that provides functions to register and lookup the relevant context aggregators. It is implemented by ContextDeliveryManagerImpl class. The implementation utilized techniques such as semantics based matchmaking and access control to provide a comprehensive context delivery system. These functions are supported by MatchmakingModule class and AccessController respectively. Matchmaking module contacts the underlying ontology database to incorporate semantics in the service matching process. Access controller module manages the policy/rules database which contains the system level and service level access control policies. RegistrationModule handles the service registration on behalf of the context aggregators. The queries and service registrations are encapsulated in ContextLookup and ContextAdvertisement respectively which extend the ContextServiceDescriptor. A QueryElement is a collection of one or more service descriptors. Since these data structures are not in OWL format, Converter module is used to convert them to the format required to map and match with ontologies. Further breakdown of these classes is shown in the individual descriptive diagrams as follows:

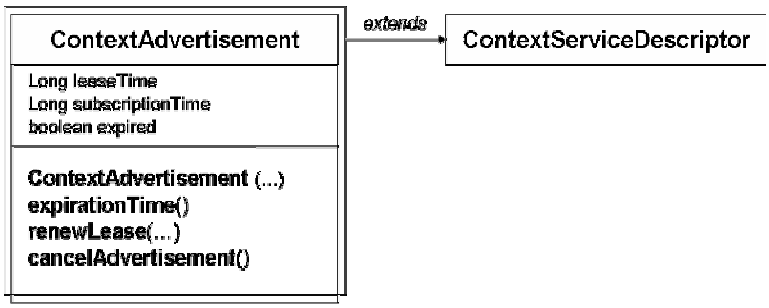


Figure 53: Context Advertisement Class Diagram

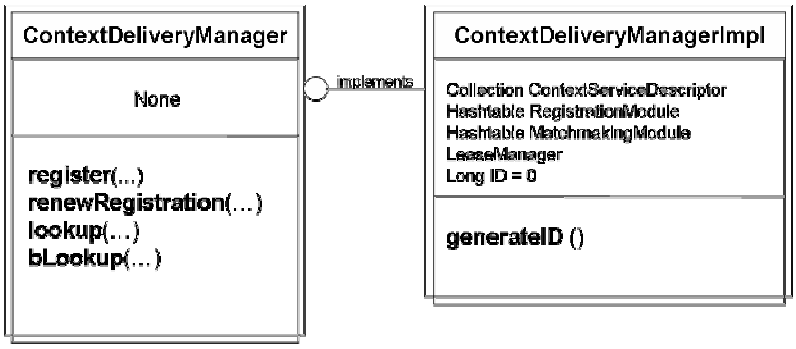


Figure 54: ContextDeliveryManger Class Diagram

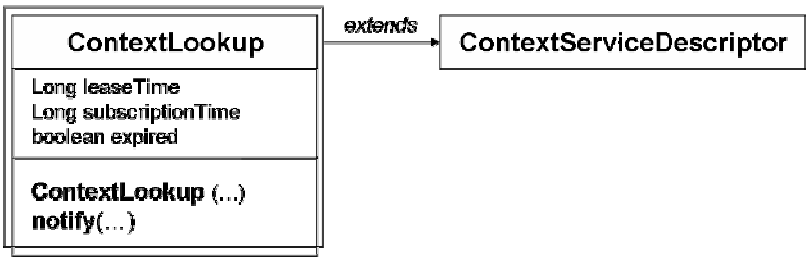


Figure 55: Context Lookup Class Diagram

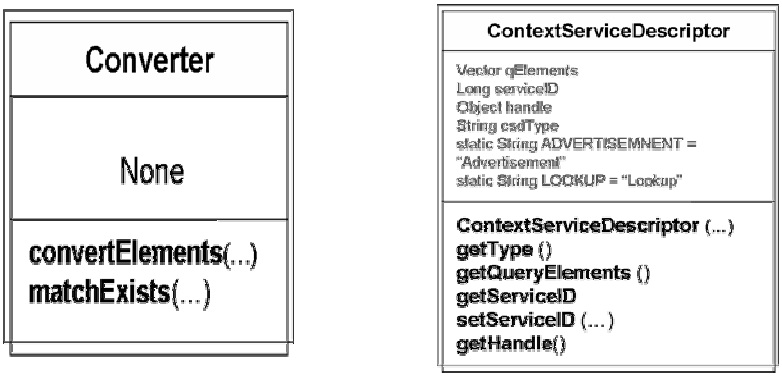


Figure 56: Converter Class and ContextServiceDescriptor Class Diagram

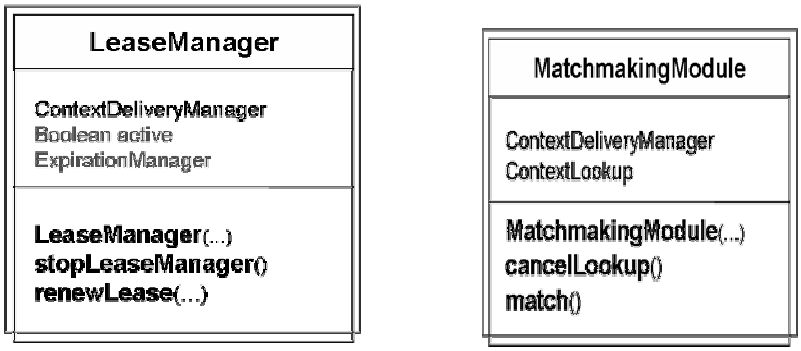


Figure 57: Lease Manager and MatchmakingModule Class Diagram

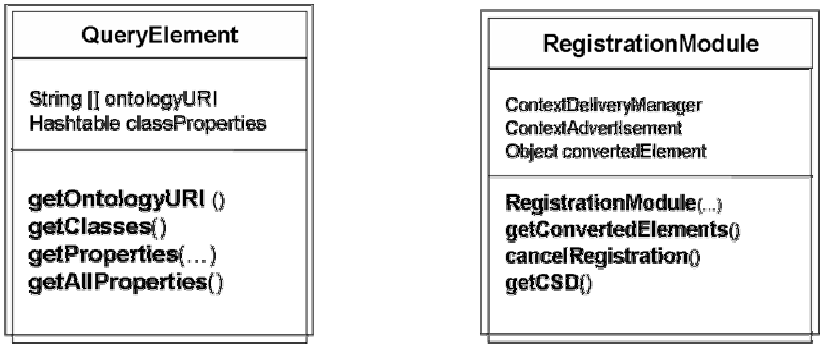


Figure 58: QueryElement and RegistrationModule Class Diagram

11.3.5.2 Sequence Diagram

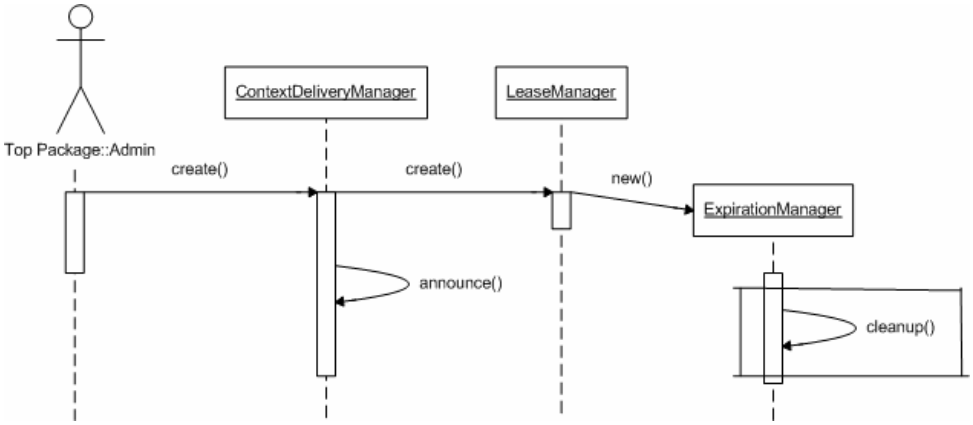


Figure 59: Sequence Diagram showing initialization of ContextDeliveryManger instance

In the above figure, an instance of CDM is created by the user/admin. As an initial step, CDM instantiates a lease manager and an expiration handler is created in the form of ExpirationManager to handle a clean exit of the CDM when it shuts down. This finalize operation and sequence of steps is shown in the following figure.

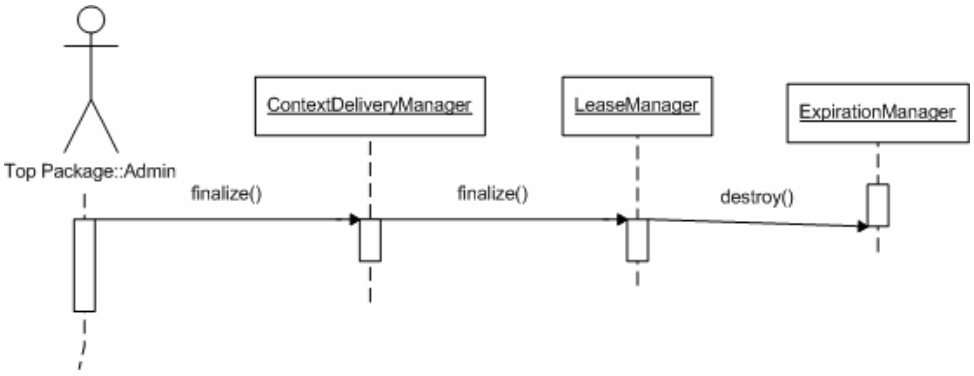


Figure 60: Sequence Diagram showing finalization of ContextDeliveryManager instance

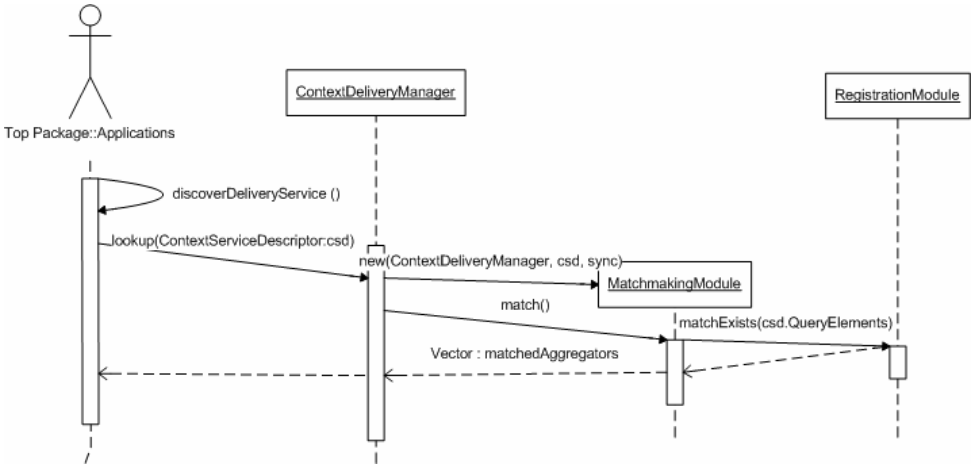


Figure 61: Sequence diagram showing Lookup operation

In the above figure, CDM manager is queried by the application by passing a ContextServiceDescriptor (CSD) object. CDM passes this CSD object to the MatchmakingModule which returns any matching results if and when they exist. It makes use of a 'register your interest' mechanism by registering the user query with the RegistrationModule in combination with the LeaseManager. For the

requested amount of lease, the RegistrationModule is bound to make matching results available when they are found. In case lease expires, it's the job of the LeaseManager to renew the lease if the application is interested to keep on looking for an extended period of time. In case lease is not required to be renewed, the query is dropped and the registration is cancelled. Registration of query (CSD) sequence is shown in the following figure.

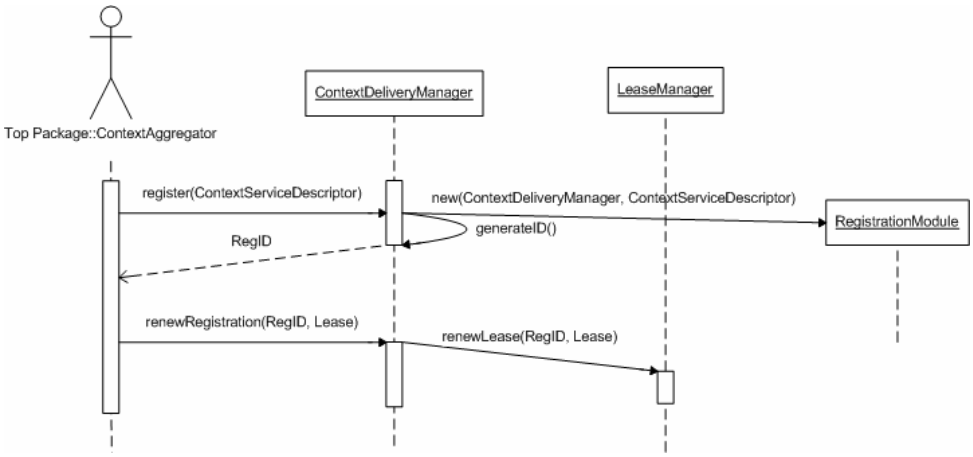


Figure 62: Sequence Diagram showing Registration Operation

11.4 Scenario Description – Meeting Room Scenario

Suppose it is scheduled that there will be a meeting in the Meeting Room 205. In the meeting, Professor L. will give a presenation. The system holds the information about all the expected audiences for the meeting. The presentation material has also prepared.

The students and professor has their own PDAs. Whenever a person enters the meeting room, the WLAN Feature Extraction Agent sends the information about presence of his PDA to the system, so that the system can infer that the person is in the room. When CAMUS regconizes that all the expected audiences as well as the presenter have come, it will assume that the meeting can be started. Then an alert message will be popped up to ask the presenter whether he wants to start the presentation or not. If he agree to start, the presentation material will be displayed. Concurrently the system will send the control message to the light dimming control, to adjust the appropriate light level (normally the light should be off during the presentation).

The system can also sense and tell the users about some environment information such as the light level and current temperature. Then based on the

adjustment of users, the system can "learn" the users' preferences and use that information for later automatically controlling the devices.

11.5 Prototype Description

11.5.1 Prototype modules

The Camus prototype contains following services :

- Sensor Access Module (with Graphic User Interface)
- Feature Tuple Space service
- Mapping Manager
- Location Mapping (the client mapping service)
- Context Data Manager
- Context Reasoning Manager
- Context Delivery service
- Agent Context Aggregator
- Environment Context Aggregator

Besides, there's one Meeting Broker application illustrating how to utilize the fun ctions provided by Camus services.

11.5.2 System environment

CAMUS demonstration runs on JVM and uses Jini lookup service. So you have t o set up Java and Jini

- **Set up Java runtime environment**

After install the JVM, set JAVA_HOME variable and put the path to JAVAHO ME\bin to the PATH variable.

- **Set up Jini**

After install Jini, set JINI_HOME variable

- **Set up IBM Tuple Space**

FTSManager service use IBM Tuple Space, so you have to set up IBM Tuple Spa ce. First copy the IBM tuple space into one folder. Then set the TSPACES variab le to that folder

11.5.3 System configuration

Change the config files

- **JINI_HOME\config**

- start-activatable-reggie.config
- start-phoenix.config

```
private static codebase = ConfigUtil.concat(new Object[] {
"http://", "IP OF THE JINI LOOKUP SERVER", ":8085/reggie-dl.jar"});
```

- **FTS:**
  - fts.congif

```
CAMUS_HOME ="FOLDER CONTAINS FTS";
ipToUse = "IP OF THE COMPUTER WHERE FTS IS RUNNING";

initialLookupLocators=new net.jini.core.discovery.LookupLocator[]{
  new LookupLocator("jini://IP OF THE JINI LOOKUP SERVER")
};
```

- **CD:**
  - cdm.congif : same to FTS
- **SAM:**
  - sam.congif : same to FTS
- **Mapping Manager**
  - fcm.congif : same to FTS
- **Context Data Manager:**
  - cdr.congif : same to FTS
- **Location Mapping service:**
  - loc-map.congif : same to FTS
- **Agent CA:**
  - ca-agent.congif : same to FTS
- **Environment CA:**
  - ca-envi.congif : same to FTS
- **Demo app:**
  - app.congif : CAMUS_HOME and initialLookupLocators

11.5.4 Initialize the communication enviroment : starting the Jini services

In JINI_HOME\scripts, run following services:

+ start-ws.bat
+ start-phoenix.bat
+ start-activatable-reggie.bat

11.5.5 Running the prototype midleware serivces:

11.5.5.1 FTS

When starting the runFTS.bat service, this console will be displayed:

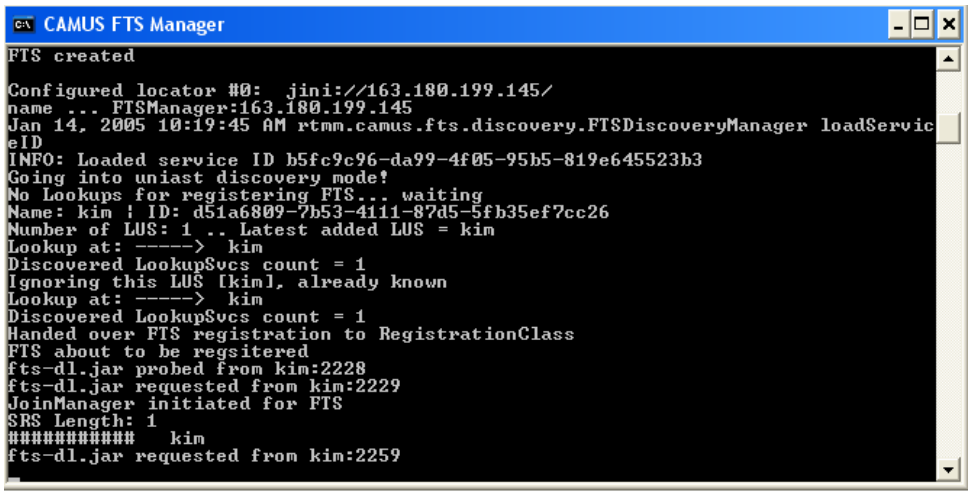


Figure 63: Feature Tuple Space console.

11.5.5.2 SAM

The GUI of SAM:

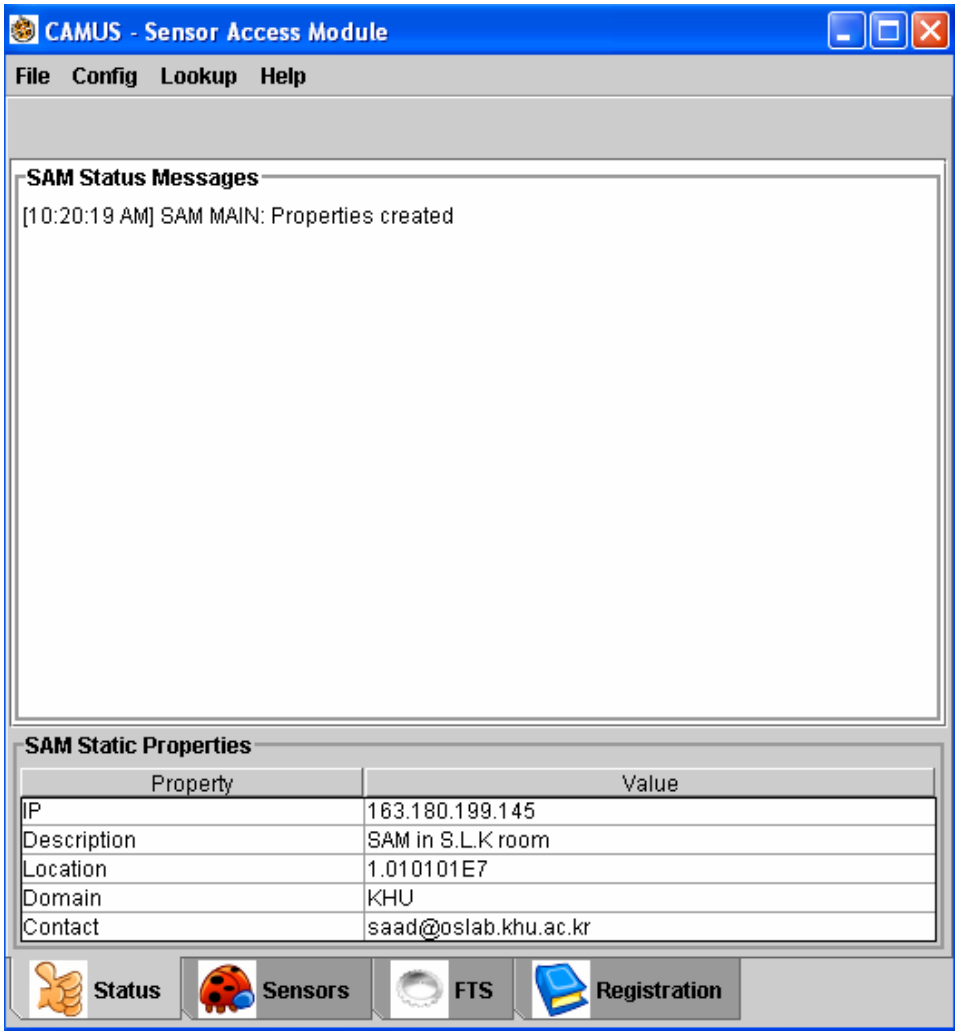


Figure 64: SAM user interface.

- First, add some sensors by loading some sensor drivers (those drivers hav e to be built as jar files)

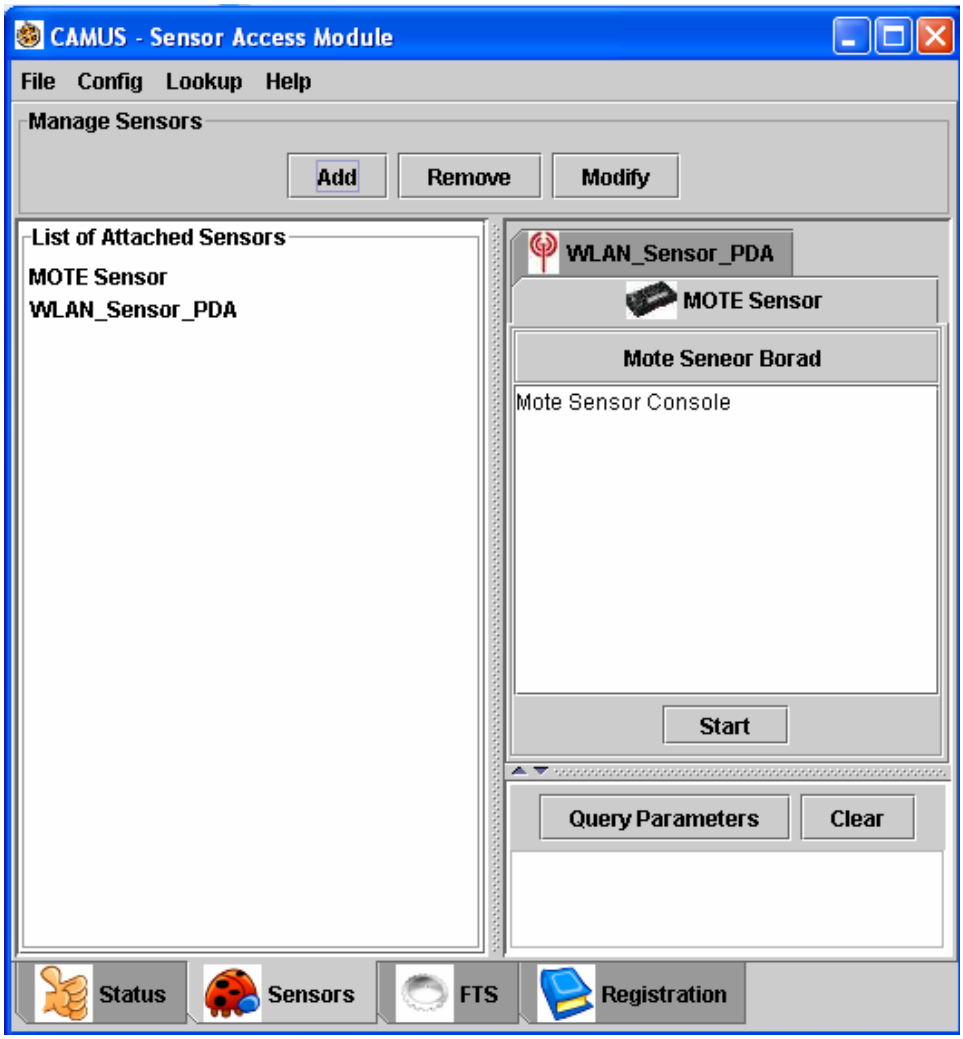


Figure 65: SAM user interface after adding MOTE and WLAN sensors.

- Second, add the feature extraction agents for those sensors (feature extrac tion agents also have to be built as jar files)

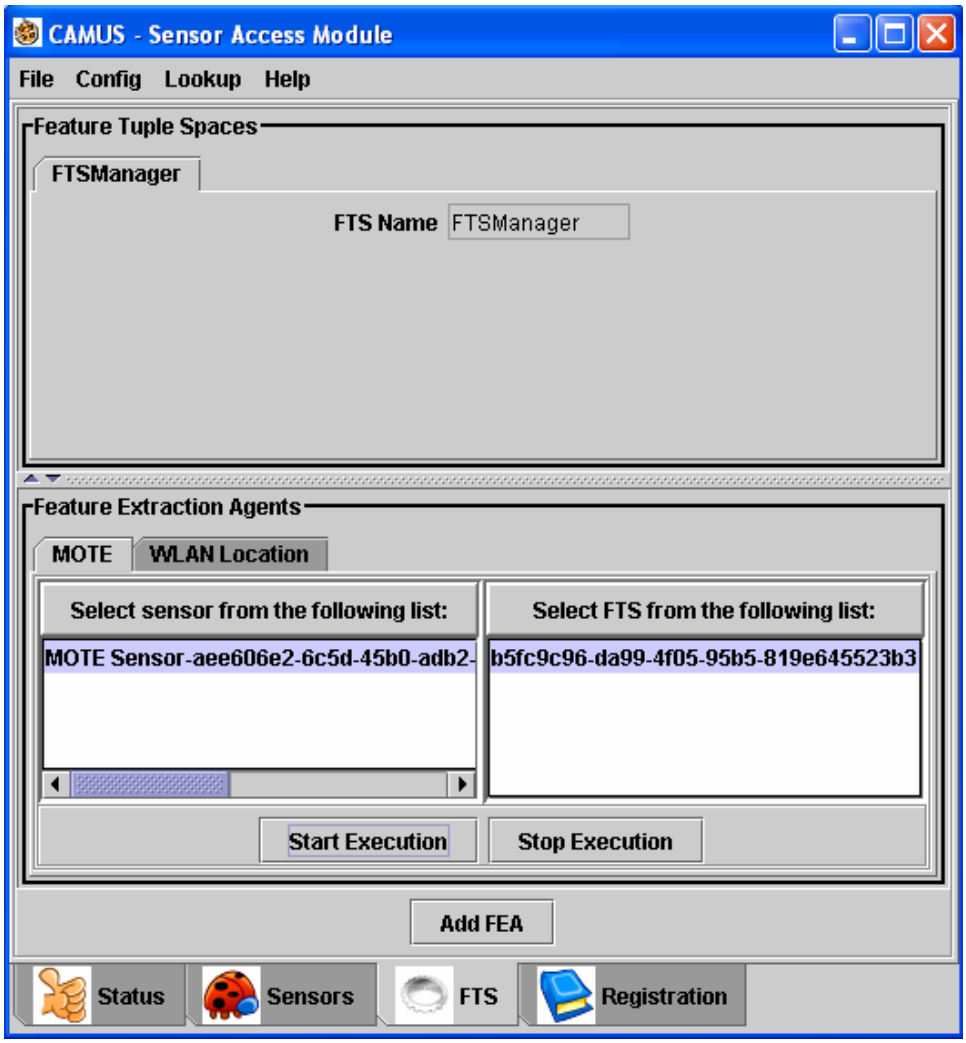


Figure 66: SAM user interface to add and execute Feature Exaction Agents.

- Select one sensor, one FTS and start execution

11.5.5.3 Context data manager service

This console will be displayed when starting the runCRM.bat:

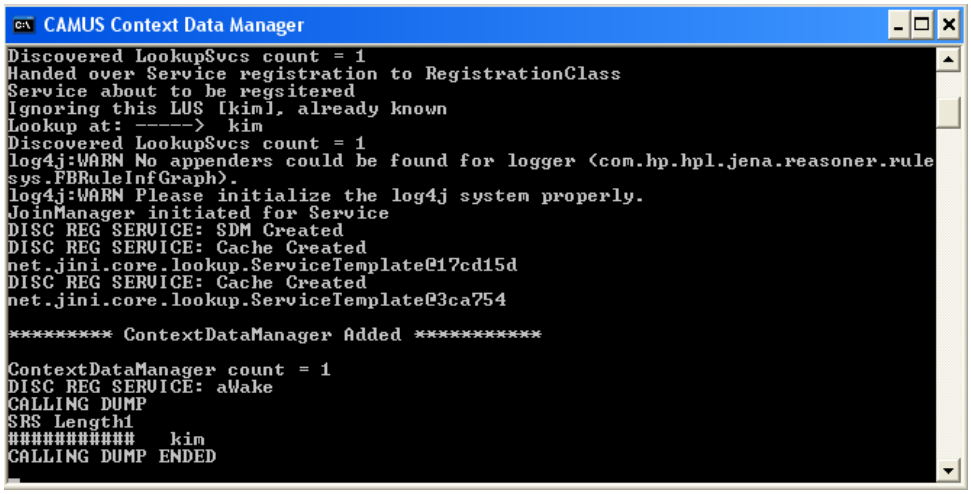


Figure 67: Context Data Manager console

11.5.5.4 Feature to context mapping manager service

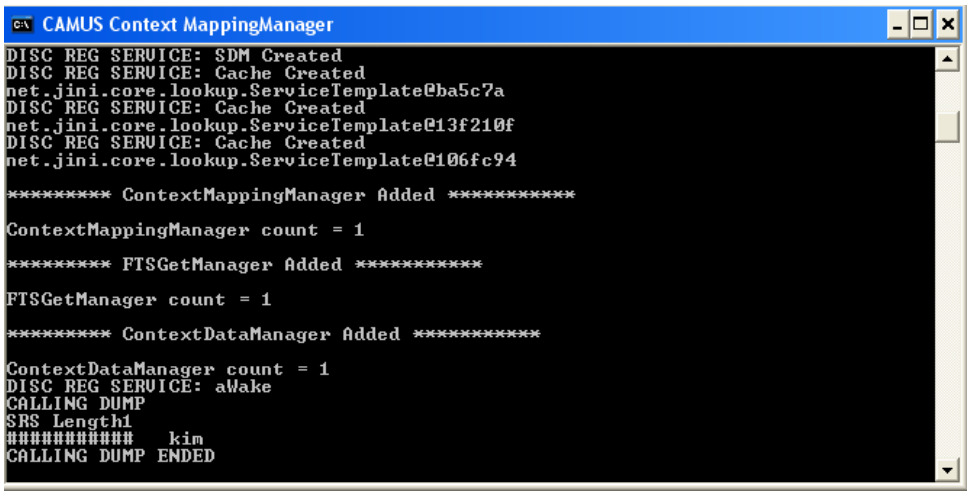


Figure 68: Context Mapping Manager console

11.5.5.5 Location Mapping service

Run the file runLocationMapping.bat. The console should close after running.

11.5.5.6 Agent Aggregator

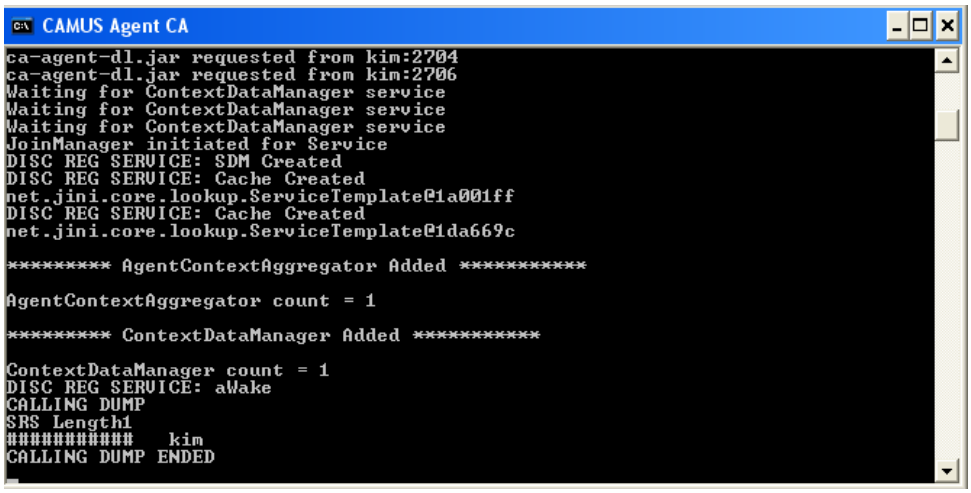


Figure 69: Agent Aggregator console

11.5.5.7 Environment Aggregator

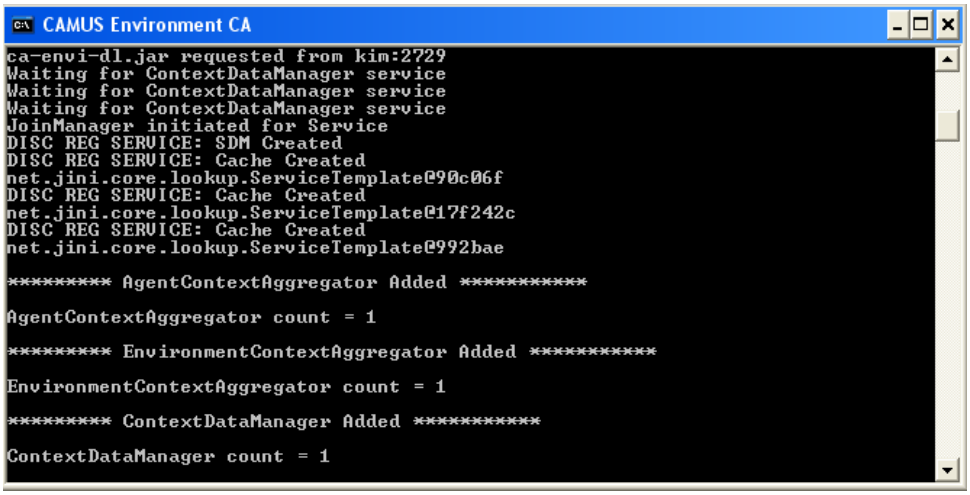


Figure 70: Environment Aggregator console

11.5.6 Running the prototype application:

11.5.6.1 Introduce the application user interface

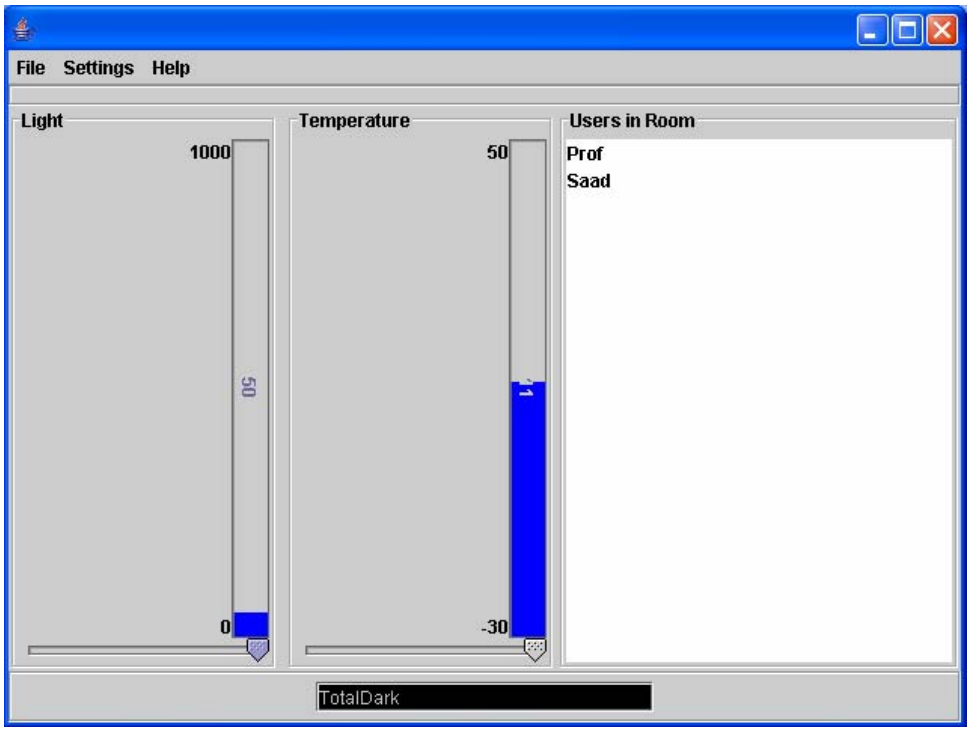


Figure 71: The user interface of Meeting Room application.

The user inteface of this prototype application contains a Light meter, a Temparature meter, a User in Room list and a Light level indicating text box.

- The Light meter indicates Light (by absolute value) in room
- The Temperature meter indicates Temperature in room
- User in Room list shows all the users in the room
- The small box at the bottom show the Light level in room

11.5.6.2 Changing the application seting

User can go to Setting to change the Configuration of the application and also select the presenation file for the coming Presentation

When select Setting, the following panel will be displayed :

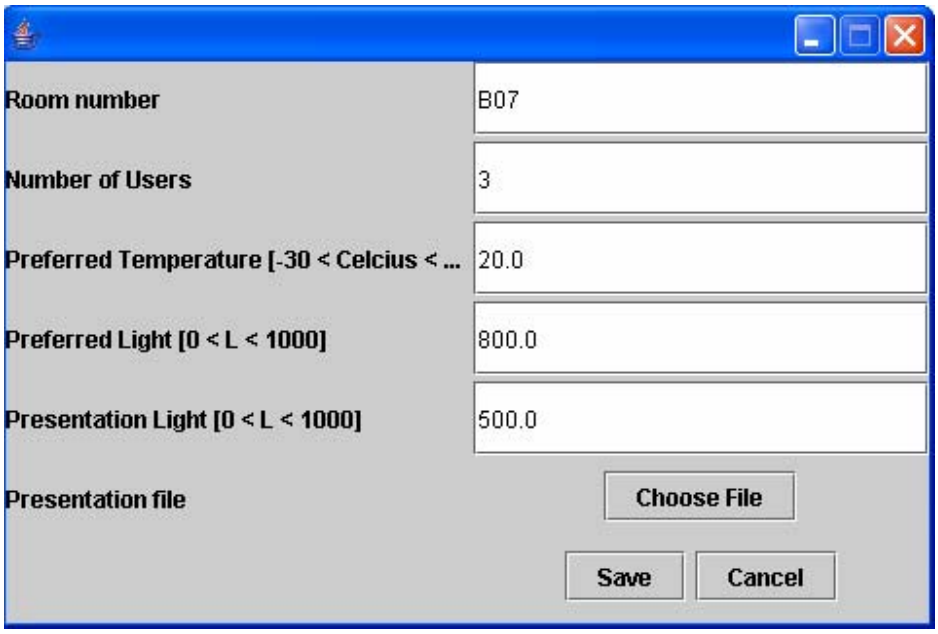


Figure 72: Configuration panel of the application.

- Room number : select the current room
- Number of user : the minimum number of users needed to start the meet ing
- Preferred temperature : if the temperature is higher or lower than this Pre ferred temperature, the air conditioner control will adjust the temperature (this function hasn't been implemented due to lackness of device)
- Same to Preferred Light
- Presentation Light : when starting the Presentation, the light will be adjust to this level

To select the presentation material, in Presentation file: click Choose File to select a file. Then Save.

11.5.6.3 Application function : detect the approriate time to start the prsentation based on users' presences

When all expected users are in the room (including the students and professor), the program will ask if you want to start the presentation.

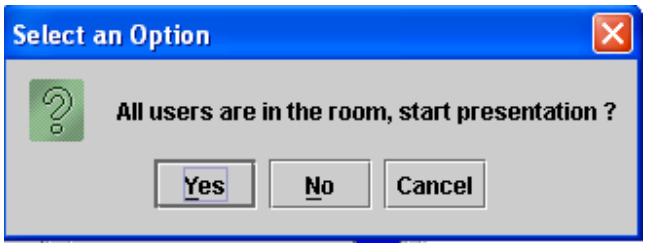


Figure 73.: Pop-up dialog to get user's confirmation about starting the presentation.

If select Yes, the presentation file will be opened, and the Light level will be adjust according to the Presentation Light setting.

11.6 Prototype Evaluation

Currently, the performance of the prototype is not good. The system run a bit slowly compare to other systems, due to the delay time for remote calls between the modules.

However, the prototype proves the feasibility and all the advantages of the middleware, which are describe in above parts.

11.7 System APIs to interact with the applications and u-Gateway

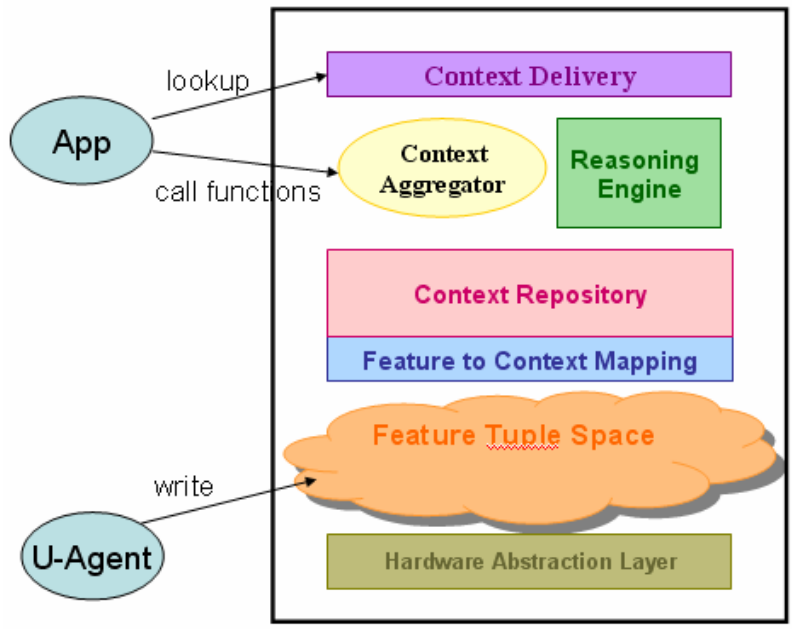


Figure 74: How CAMUS interacts with applications and u-Agent modules of u-Gateway.

11.7.1 Input API – Feature Tuple Space API

rtmm.camus.fts.put.FTSPutManager

```
  public FTSWriter getFTSWriter () throws java.rmi.RemoteException ;
  public FTSWriter getFTSWriter (long lease) throws java.rmi.RemoteException ;
  public void returnFTSWriter (FTSWriter w) throws java.rmi.RemoteException ;
```

rtmm.camus.fts.put.FTSWriter

```
  long MAX_LEASE = Long.MAX_VALUE;
  public FTSTupleID write ( FeatureTuple tuple )  throws java.rmi.RemoteException ;
  public FTSTupleID write ( FeatureTuple tuple, long lease )   throws
java.rmi.RemoteException;
  public FTSTupleID write ( FeatureTuple tuple, String index ) throws
java.rmi.RemoteException;
  public FTSTupleID write ( FeatureTuple tuple, String index, long lease )    throws
java.rmi.RemoteException;
  public FTSTupleID [] writeN ( FeatureTuple [] tuples ) throws
java.rmi.RemoteException  ;
  public FTSTupleID [] writeN ( FeatureTuple [] tuples , long lease )   throws
java.rmi.RemoteException ;
  public FTSTupleID [] writeN ( FeatureTuple [] tuples , String index )  throws
java.rmi.RemoteException;
  public FTSTupleID [] writeN ( FeatureTuple [] tuples, String index, long lease )  throws
java.rmi.RemoteException ;

  public int delete (FeatureTuple template) throws java.rmi.RemoteException;
  public int deleteN (FeatureTuple [] template) throws java.rmi.RemoteException;

  public FTSTupleID update ( FeatureTuple tuple )  throws java.rmi.RemoteException;
  public FTSTupleID [] updateN ( FeatureTuple [] tuples ) throws
java.rmi.RemoteException;
```

11.7.2 Output API – Context Delivery and Context Aggregator

rtmm.camus.cdm.match.CDMatchmakingManager

```
public Object[] getAggregator (String [] params) throws java.rmi.RemoteException;
```

The API for Context Aggregator is Service-specific

Example:

```
/*
 * this is the general context aggregator
/*
```

rtmm.camus.ca.ContextAggregator

```
  public ContextRecordset queryContext(String query) ;
  public ContextObject[] searchContext(String contextLabel);
  public ContextObject[] searchContext(ContextObject contextObject);
/*
 * this is the context aggregator to provide agent's information
/*
```

rtmm.camus.ca.AgentContextAggregator

```
  public ArrayList listAgent () throws java.rmi.RemoteException;
  public AgentModel getAgent (int agentID) throws java.rmi.RemoteException;
  public void addAgent (AgentModel ag) throws java.rmi.RemoteException;
  public void deleteAgent (String agentID) throws java.rmi.RemoteException;
  public String getUserLocationByAgentID(String agentID) throws
java.rmi.RemoteException;
  public String setAgentActivity(int agentID, String activity) throws
java.rmi.RemoteException;
  public String setEnvironmentPreference(int agentID, String activity, String
environmentType, String value) throws java.rmi.RemoteException;

/*
 * this is the context aggregator to provide environment's information
/*
```

rtmm.camus.ca.EnvironmentContextAggregator

```
  public double getCurrentTemperature(String placeName);
  public String getCurrentLightIntensity(String placeName);
  public double getCurrentTemperature(int userID);
  public String getCurrentLightIntensity(int userID);
        //light value : "TotalDark", "Dark", "Normal", "Bright"
```

rtmm.camus.ca.datamodel.AgentModel

```
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getId() {
```

```
  return id;
}
public void setId(int id) {
  this.id = id;
}
public String getRFID() {
  return RFID;
}
public void setRFID(String RFID) {
  this.RFID = RFID;
}
public String getPDAIP() {
  return PDAIP;
}
public void setPDAIP(String PDAIP) {
  this.PDAIP = PDAIP;
}
public String getMoteID() {
  return MoteID;
}
public void setMoteID(String MoteID) {
  this.MoteID = MoteID;
}
```

Chapter 12

CONCLUSION

12. Conclusion

CAMUS envisions a comprehensive middleware solution that not only focuses on providing context composition at the software level but also facilitates dynamic features retrieval at the hardware level by masking the inherent heterogeneity of environment sensors. Complexity is handled by providing 'separation of concerns' between environment features extraction, contextual data composition and context interpretation. The use of Feature Tuple Space and distributed communication protocol promise a possibility of extending CAMUS largely in both scale and scope. Different reasoning mechanisms are incorporated in CAMUS as pluggable services, ranging from rules written in different types of logic to machine-learning mechanisms.

One of the fundamental characteristics of context-aware systems is formalization of context models, expressing entities independent of any specific application. Although complete formalization is impossible but it can be applied to the degree allowed by the domain for relatively stable or invariant entities. Using ontologies to describe the entities formally support also knowledge sharing, reuse, and logical reasoning. We believe that formalizing domains should be seen as emergent phenomenon constructed incrementally, leading to the sharing of contextual information among heterogeneous context-aware systems. Here, also we discussed how formal modeling is useful for heterogeneous ubiquitous computing environment and presented our ontology for the home domain. We also discussed reasoning capabilities provided once context models are formalized. Moreover, CAMUS is backed by a OWL format context data repository which offers the best support for uncertainty to increase the accuracy of context information, as well as a distributed querying mechanism which help building a large system over multi domains.

With a systematic approach, CAMUS is proved to be a flexible and reusable novel middleware framework.

REFERENCES

References

[1] M. Weiser: Scientifc America. The Computer for the 21st Century. (Sept. 1991) 94-104; reprinted in IEEE Pervasive Computing. (Jan.-Mar. 2002) 19-25
[2] http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html
[3] Schilit, B. N., N.I. Adams, and R. Want: Context -Aware System Architecture for Mobile Distributed Computing. PhD thesis, 1995
[4] Dey, A.K., et al.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Anchor article of a special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, Vol. 16. (2001)
[5] Pascoe J., Ryan, N. S. & Morse, D. R. (1998). Human-Computer-Giraffe Interaction – HCI in the field. Proceedings of the Workshop on Human Computer Interaction with Mobile Devices, Glasgow, Scotland.
[6] Context Toolkit project http://www.cs.berkeley.edu/~dey/context.html
[7] W. Emmerich. Engineering Distributed Objects. John Wiley and Sons, Ltd., 2000.
[8] K. Edwards. Core JINI. Prentice Hall, 1999.
[9] Sun. Javaspaces. http://www.sun.com/jini/specs/jini1.1html/js-title.html, 2001.
[10] IBM. T Spaces. http://www.almaden.ibm.com/cs/TSpaces/, 2001.
[11] Xiaodong Jiang, James A. Landay: Modeling Privacy Control in Context – Aware Systems. IEEE Pervasive Computing, Vol. 1, No. 3 July-September 2002.
[12] Philip Gray and Daniel Salber: Modeling and Using Sensed Information in the Design of Interactive Applications. 8th IFIP International Conference, EHCI 2001, Toronto, Canada, 2001
[13] Guanling Chen and David Kotz: Context Aggregation and Dissemination in Ubiquitous Computing Systems. In Proceedings of the Fourth 1EEE Workshop on Mobile Computing Systems and Applications, June 2002.oi
[14] Jeffrey Heer, Alan Newberger, Chris Beckmann, and Jason I. Hong: liquid: Context -Aware Distributed Queries. Ubiquitous Computing, 5th International Conference, Seattle (UbiComp 2003), October 12-15, 2003.
[15] Anand Ranganathan, Roy H. Campbell: A Middleware for Context -Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference, Brazil, June, 2003.

[16] S. K. Das, D. J. Cook, A. Bhattacharya, E. O. Heierman, III, and T.-Y. Lin: The Role of Prediction Algorithms in the MavHome Smart Home Architecture, IEEE Wireless Communications Special Issue on Smart Homes, 9(6), pages 77-84, 2002.
[17] G. Thomson, S. Terzis, and P. A. Nixon: Towards Dynamic Context Discovery and Composition. The First UK-UbiNet Workshop, London, UK, 2003.
[18] Haarslev, V., Moller, R.: Racer: A Core Inference Engine for the Semantic Web. In: EON2003, Sanibel Island, Florida. (Oct. 2003)
[19] Guanling Chen and David Kotz.: Solar: An Open Platform for Context – Aware Mobile Applications. In Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), Switzerland, June, 2002.
[20] Hong, J. I., et al.: An Infrastructure Approach to Context -Aware Computing. HCI Journal, 2001, Vol. 16.
[21] Zadeh, L.: Fuzzy Sets. Information and Control 8, (1965) 338-353
[22] Korpipaa, P., Koskinen, M., Peltola, J., Makela, S. M., Seppanen, T.: Bayesian approach to sensor-based context awareness. In: Personal and Ubiquitous Computing, Vol. 7, Issue 2. (July 2003) 113-124
[23] Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Revised second printing. Morgan Kaufmann, San Francisco. (1988)
[24] S. S. Yau, F. Karim, Y. Wang, B. Wang, S.Gupta: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. (Jul.-Sep. 2002) 33-40
[25] J. Davies, D. Fensel, F. V. Harmelen: Towards the Semantic Web, Ontology-Driven Knowledge Management, John Wiley & Sons. (Nov. 2002)
[26] W3C Web Ontology Working Group: The Web Ontology language: OWL. http://www.w3.org/2001/sw/WebOnt/
[27] Klyne, G., Caroll, J. J.: Resource Description Framework Abstract Concept and Syntax. W3C Recommendation. (10 Feb. 2004)
[28] FIPA Device Ontology Specification. http://www.fipa.org/specs/fipa00091/SI00091E.pdf
[29] Hobbs, J. R.: A Daml ontology of time. http://www.cs.rochester.edu/~ferguson/daml/damltime-nov2002.txt. (2002)
[30] Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/
[31] Andy Seaborne: RDQL – A Query language for RDF. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
[32] Baadar, F., Horrocks, I., Sattler, U.: Description Logics. Handbook on Ontologies. (2004)
[33] D. A. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In: Principles of Knowledge Representation and Reasoning (KR92), San Mateo, California, Aug. (1992)

[34] Allen, J. F., Ferguson, G.: Actions and Events in Interval Temporal Logic. Technical Report 521, The University of Rochester, Computer Science Department, Rochester, New York. (Jul. 1994)
[35] Protégé Project. http://protege.stanford.edu
[36] Hung, N.Q., Shehzad, A., Kiani, S. L., Riaz, M., Lee, S.: A Unified Middleware Framework for Context Aware Ubiquitous Computing. In: EUC2004, Japan. (Aug. 2004)
[37] M.P. Papazoglou and D. Georgakopoulos: Service oriented computing. Communications of the ACM, Vol. 46, No. 10. (Oct 2003)
[38] Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A Semantic Web Approach to Service Description for Matchmaking of Services. HP Labs Bristol. HPL-001-183. (2001)
[39] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the Web Services Web – An Introduction to SOAP,WSDL, and UDDI. In: IEEE Internet Computing, vol. 6, no. 2,(2002) 86–93
[40] I. Horrocks. DAML+OIL: a reason-able web ontology language. In Proc. of EDBT 2002, number 2287 in Lecture Notes in Computer Science, pages 2–13. Springer, Mar. 2002
[41] Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: WWW 2003, ACM. (2003) 331-339
[42] Hengartner, U., Steenkiste, P.: Acccess Control to Information in Pervasive Computing Environments. In: 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003
[43] Ranganathan, A., McGrath, R. E., Campbell, R. H., Mickunas, M. D.: Use of Ontologies in a Pervasive Computing Environment. In: The Knowledge Engineering Review, vol. 18, no.3, (2004) 209-220, Cambridge University Press
[44] Koshutanski, H., Massacci, F.: Deduction, Abduction and Induction, the Reasoning Services for Access Control in Autonomic Communication. In: proceedings of the 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC 2004), October 2004, Berlin, Germany. Springer, 2004
[45] Kagal, T., Finin, L., Josh, A. Trust-Based Security in Pervasive Computing Environments. In: IEEE Computer, pages 154–157, Dec. 2001
[46] Sheshagirim, M., Sadeh, N. M., Gandon, F.:Using Semantic Web Services for Context-Aware Mobile Applications. In: MobiSys 2004 Workshop on Context Awareness, Massachusetts, USA (Jun. 2004)
[47] Pokraev, S., Koolwaaij, J., Wibbels, M: Extending UDDI with context-aware features based on semantic service descriptions. In: International Conference on Web Services (ICWS), 2003
[48] Broens, T., Pokraev, S., Sinderen, M., Koolwaaij, J., Costa, P. D.: Context-aware, ontology-based, service discovery. In: EUSAI 2004, pp. 72–83,. Springer, 2004
[49] http://xml.coverpages.org/xacl.html

[50] S. Jang, Woo, W.: Ubi-UCAM: A Unified Context-Aware Application Model. In: Context 2003, Stanford, CA, USA. (Jun. 2003)
[51] Gellersen, H.W., Schmidt, A., Beigl, M.: Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts. In: Mobile Networks and Applications, Vol. 7. (Oct. 2002) 341-351
[52] Burrell, J. and Gay, G.K. (2002). E-Graffiti: evaluating real-world use of a context-aware system. In: Interacting with Computers 14 (2002), 301-312.
[53] Kidd, C.D., et. al.: The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In: Cooperative Buildings, http://citeseer.ist.psu.edu/kidd99aware.html (1999) 191-198
[54] Waldo, J., Wyant, G., Wollrath, A., Kendall, S.: A Note on Distributed Computing. In: Mobile Object Systems: Towards the Programmable Internet. Springer-Verlag, Heidelberg, Germany, April 1997, pp. 49-64
[55] Yang, K., Galis, A.: Policy-Driven Mobile Agents for Context-Aware Service in Next Generation Networks. In: Mobile Agents for Telecommunication Applications (MATA), 5th International Workshop, Morocco, (Oct. 2003)
[56] Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A Semantic Web Approach to Service Description for Matchmaking of Services. HP Labs Bristol. HPL-001-183. (2001)
[57] Furmento, N., Hau, J., Lee, W., Newhouse, S.: Darlington, J. Implementations of a Service-Oriented Architecture on top of Jini, JXTA and OGSA. In: Proceedings of the UK e-Science Program All Hands Meeting 2003, Nottingham, UK. Sept., 2003
[58] Sun Microsystems, Inc.: JiniTM *Technology Core Platform Specification* http://java.sun.com/products/jini/2_0index.html
[59] IncaX Service Browser: http://www.incax.com
[60] Mamei, M., Zambonelli, F., Leonardi, L.: Tuples on the Air: a Middleware for Context-Aware Multi Agent Systems. In: 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), Providence, Rhode Island, USA, May 2003
[61] Román, M., et al.: Gaia: A Middleware Infrastructure to Enable Active Spaces. In: IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002
[62] Yau, S. S., et al.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing," IEEE Pervasive Computing, joint special issue with IEEE Personal Communications, 1(3), July-September 2002, pp.33-40
[63] Keidl, M., Kemper, A.: Towards Context Aware Adaptable Web Services. In: Proceedings of the 13th World Wide Web Conference (WWW), New York, USA, May 17-22, 2004, pp. 55-65
[64] William Grosso: Activation Framework. In: Java RMI, O'Reilly & Associates, Inc. CA 95472, USA, October 2001 Ch. 17
[65] Cooltown Project, http://www.cooltown.com/cooltown/index.asp

[66] Asim Smailagic, Daniel P. Siewiorek, Joshua Anhalt, David Kogan, and Yang Wang, : Location Sensing and Privacy in a Context Aware Computing Environment. Pervasive Computing, 2001
[67] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the Web Services Web – An Introduction to SOAP,WSDL, and UDDI. In: IEEE Internet Computing, vol. 6, no. 2,(2002) 86–93
[68] Antonis Kalis: Communication Paradigms, Life, the Universe and Everything
[69] Rebecca Allen: The Return of the Physical World
[70] Thomas von der Gruen: Wireless Body Area Networks in Healthcare Applications
[71] Ulf Keijer: Smarthomes
[72] Duska Rosenberg: Understanding Mediated Communication
[73] Mikhail Smirnov: Managing Internet complexity in Autonomic Communication
[74] Michel Riguidel: Communications and Computer Science in 2020
[75] Giuseppe Riva: Immersive Virtual Telepresence and Virtual Residence in Communication
[76] Javier Fonollosa: Capacity limits and trade-off analysis advanced transmission configurations
[77] Sergio Barbarossa: Connectivity of self-organizing networks: Resource optimization based on Coop. Transmis.
[78] Robert Plana: Advanced micro and nanotechnologies for future communications
[79] Alexander Dmitriev: UWB communications with chaotic carriers
[80] Ioannis Stavrakakis: NETWORK as an info ANYTHING Engine
[81] Franco Zambonelli: Spray Computers: frontiers of pervasive computing - the need for novel
[82] Roger Whitaker: Device Intelligence for 2020
[83] Alois Ferscha: Ideas on Pervasive Computing
[84] Fabrizio Davide: Strategic direction towards Autonomic Communitation
[85] Simon Dobson: Why communicate
[86] Lidia Yamamoto: Self-Management in Autonomic Communication
[87] Ulrich Lang: Modelling security for complex, heterogeneous, distributed IT systems
[88] Fabio Massacci: Quality of Protection for Communications

Appendix A

CONTEXT MODELING SCHEMES

A.1. Introduction

In the past a variety of context models were subject of research, because a well designed model is a key access to the context in any context-aware system. While early models mainly addressed the modeling of context with respect to one application or an application class, generic context models are of interest since many applications can benefit from these. While some models take the users current situation, e.g. "in a meeting" into account, others model the physical environment, i.e. locations. First steps towards a common understanding of context have been published, mostly with respect to location, identity, and time. The objective of most current research is to develop uniform context models, representation and query languages as well as reasoning algorithms that facilitate context sharing and interoperability of applications.

In this appendix we want to make a survey of the most relevant current approaches to modeling context for ubiquitous computing.

A.2. Typical Context Modeling & Integration Requirements for UbiComp

In the literature several definitions of the term *context* can be found [36, 35, 32, 37, 10, 16, 41]. A detailed discussion of the differences within these definitions is out of the scope of this paper but has some impact on the models introduced in the next section. A selection of some context-aware mobile computing research is for instance provided by [10, 39].

Ubiquitous computing systems make high demands on any context modeling approach in terms of:

1. *High level of formality* *(for):* It is always a challenge to describe contextual facts and interrelationships in a *precise* and *traceable* manner. For instance, to perform the task "print document on printer near to me", it is required to have a precise definition of terms used in the task, for instance what "near" means to "me". It is highly desirable, that each participating party in an ubiquitous computing interaction shares the same interpretation of the data exchanged and the meaning "behind" it (so called *shared understanding*).

2. *Distributed composition* *(dc):* Any ubiquitous computing system is a derivative of a distributed computing system (cf. figure 1) which lacks of a central instance being responsible for the creation, deployment and maintenance of data

and services, in particular context descriptions. Instead, composition and administration of a context model and its data varies with notably high dynamics in terms of time, network topology and source.

3. *Partial validation* *(pv):* It is highly desirable to be able to partially validate contextual knowledge on structure as well as on instance level against a context model in use even if there is no single place or point in time where the contextual knowledge is available on one node as a result of distributed composition. This is particularly important because of the complexity of contextual interrelationships, which make any modeling intention error-prone.

4. *Quality of information* *(qua):* The quality of a information delivered by a sensor varies over time, as well as the richness of information provided by different kinds of sensors characterizing an entity in an ubiquitous computing environment may differ. Thus a context model appropriate for usage in ubiquitous computing should inherently support quality and richness indication.

5. *Incompleteness and Uncertainty* *(inc):* The set of contextual information available at any point in time characterizing relevant entities in ubiquitous computing environments is usually incomplete and/or ambiguous, in particular if this information is gathered from sensor net works. This should be covered by the model, for instance by interpolation of incomplete data on the instance level.

6. *Applicability* *to existing environments (app):* From the implementation perspective it is important that a context model must be applicable within the existing infrastructure of ubiquitous computing environments, e.g. a service framework such as Web Services.

The mentioned requirements are in particular important for any context modeling approach applied to a ubiquitous computing environment. Some of the requirements are addressed within a certain approach's context model, some are addressed within the associated reasoning system, and some are not addressed at all within a certain approach.

A.3. Modeling Approaches

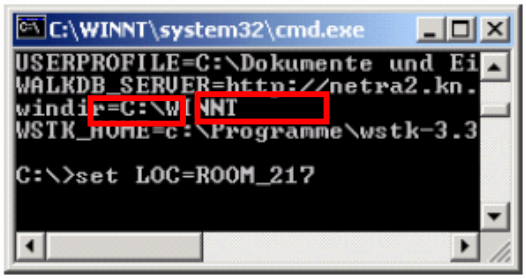
Throughout this section we will survey the most relevant context modeling approaches. These are classified by the scheme of data structures which are used to exchange contextual information in the respective system. (Obviously some of them may be classified in more than one category. In these cases they are listed in the most representative one.)

A.3.1 Key-Value Models

The model of key-value pairs is *the most simple data structure* for modeling contextual information.

Already Schilit et al. [35] used key-value pairs to model the context by providing the value of a context information (e.g. location information) to an application as an environment variable. The key-value modeling approach is frequently used in distributed service frameworks (e.g. Capeus [34]). In such frameworks, the services itself are usually described with a list of simple attributes in a key-value manner, and the employed service discovery procedure (e.g. SLP, Jini… see [39]) operates an *exact matching* algorithm on these attributes.

In particular, key-value pairs are easy to manage, but *lack capabilities for sophisticated structuring* for enabling efficient context retrieval algorithms.



Environment Variables: Key-Value-Pairs

A.3.2 Markup Scheme Models

Common to all markup scheme modeling approaches is a hierarchical data structure consisting of markup tags with attributes and content. In particular, the content of the markup tags is usually recursively defined by other markup tags.

Typical representatives of this kind of context modeling approach are *profiles*. They usually base upon a serialization of a derivative of *Standard Generic Markup Language (SGML)*, the superclass of all markup languages such as the popular XML. Some of them are defined as extension to the *Composite Capabilities / Preferences Profile (CC/PP)* [44] and *User Agent Profile (UAProf)* [46] standards, which have the expressiveness reachable by RDF/S and a XML serialization. These kinds of context modeling approaches usually extend and complete the basic CC/PP and UAProf vocabulary and procedures to try to cover the higher dynamics and complexity of contextual information compared to static profiles.

An example of this approach is the *Comprehensive Structured Context Profiles (CSCP)* by Held et al. [23]. Unlike CC/PP, CSCP does not define any fixed hierarchy. It rather supports the full flexibility of RDF/S to express natural structures of profile information as required for contextual information. Attribute names are interpreted context sensitively according to their position in the profile structure.

Hence, unambiguous attribute naming across the whole profile as necessary with CC/PP is not required. Another drawback of CC/PP, the restricted overriding mechanism of default values only, replaced by a more flexible overriding and merging mechanism, allowing for instance to override and/or merge a whole profile subtree. See figure below for a CSCP profile example.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
xmlns:cscp="context-
aware.org/CSCP/CSCPProfileSyntax#"
xmlns:dev="context-
aware.org/CSCP/DeviceProfileSyntax#"
xmlns:net="context-
aware.org/CSCP/NetworkProfileSyntax#"
xmlns="context-
aware.org/CSCP/SessionProfileSyntax#"
<SessionProfile rdf:ID="Session">
<cscp:defaults rdf:resource=
"http://localContext/CSCPProfile/previous#
Session"/>
<device><dev:DeviceProfile>
<dev:hardware><dev:Hardware>
<dev:memory>9216</dev:memory>
</dev:Hardware></dev:hardware></dev:Device
Profile>
</device>
</SessionProfile>
</rdf:RDF>
```

CSCP profile example

A similar approach to CSCP is the *CC/PP Context Extension* by Indulska et al. [27]. They extended the basic CC/PP and UAProf vocabulary by a number of component-attribute trees related to some aspects of context, e.g. concerning location, network characteristics, application requirements, session information as well as certain types of relations and dependencies.

The authors concluded that their approach is capable of enabling context-awareness to applications and other parts of ubiquitous computing infrastructure. They already realized that it is difficult and non-intuitive to capture complex contextual relationships and constraints due to the underlying CC/PP.

Another context modeling approach in the markup scheme category – which does not bear towards CC/PP – is the *Pervasive Profile Description Language (PPDL)* [14]. This XML-based language allows to account for contextual information and dependencies when defining interaction patterns on a limited scale. The number of valuable contextual aspects and the comprehensiveness of the language itself seem to be relatively limited. Due to the fact that no design criteria and only parts of the language are available to the public, the actual appropriateness of this context modeling approach remains unknown.

There are several other context modeling approaches in the markup scheme category. They are often either proprietary or limited to a small set of contextual aspects, or both.

Examples affected by these limitations are, among others, the *context configuration* of Capra et al.'s reflective middleware [9] the *Centaurus Capability Markup Language (CCML)* [28], *ConteXtML* [33] or the note-tags of the *stick-e notes* system [7].

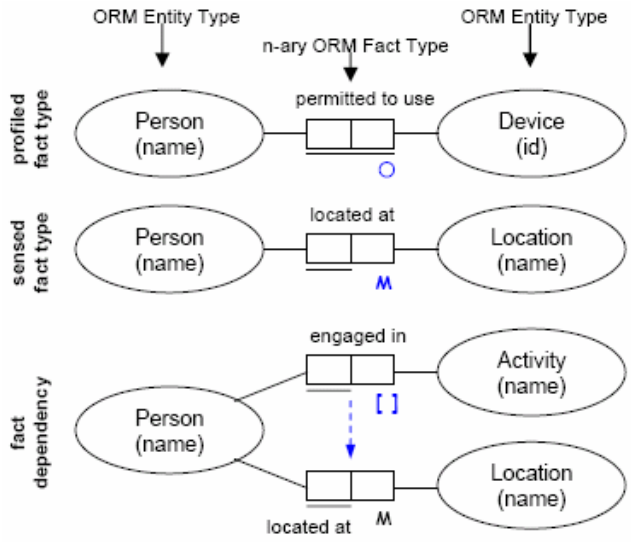
A.3.3 Graphical Models

A very well known general purpose modeling instrument is the *Unified Modeling Language (UML)* which has a strong graphical component (UML diagrams). Due to its generic structure, UML is also appropriate to model the context. This is shown for instance by Bauer in [5], where contextual aspects relevant to air traffic management are modeled as UML extensions. Another example is the nicely designed graphics oriented context model introduced in [26] by Henricksen et al., which is a context extension to the Object-Role Modeling (ORM) approach [22] according some contextual classification and description properties [25]. In ORM, the basic modeling concept is the *fact*, and the modeling of a domain using ORM involves identifying appropriate fact types and the roles that entity types play in these. Henricksen extended ORM to allow fact types to be categorised, according to their persistence and source, either as *static* (facts that remain unchanged as long as the entities they describe persist) or as *dynamic*.

The latter ones are further distinguished depending on the source of the facts as either *profiled*, *sensed* or *derived* types. Another quality indicator introduced by Henricksen is a history fact type to cover a time-aspect of the context.

The last extension to ORM made by Henricksen for context modeling purposes are fact dependencies, which represent a special type of relationship between facts, where a change in one fact leads automatically to a change in another fact: the *dependsOn* relation. See figure 3 on the right for an example of Henricksen's notation.

This kind of approach is particularly applicable to derive an ER-model [12] from it, which is very useful as structuring instrument for a relational database in information system based context management architecture such as the one described in [27].



Contextual Extended ORM

A.3.4 Logic Based Models

A logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inference) from a set of other expressions or facts. To describe these conditions in a set of rules a formal system is applied. In a logic based context model, the context is consequently defined as facts, expressions and rules. Usually contextual information is added to, updated in and deleted from a logic based system in terms of facts or inferred from the rules in the system respectively. Common to all logic based models is a high degree of formality.

One of the first logic based context modeling approaches has been researched and published as Formalizing Context in early 1993 by McCarthy and his group at Stanford [29, 30]. McCarthy introduced contexts as abstract mathematical entities with properties useful in artificial intelligence.

He prevented emphatically to give a definition what context is. Instead he tried to give a formalization recipe which allows for simple axioms for common sense phenomena, e.g. axioms for static blocks worlds situations, to be lifted to context involving fewer assumptions, e.g. contexts in which situations change. Thus

lifting rules, which relate the truth in one context to the truth in another context, are an important part of the model itself. The basic relation in this approach is ist(c, p), which asserts that the it proposition p is true in the context c. This allows for formulas such as c0: ist(contextof(" Sherlock Holmes stories"), "Holmes is a detective"), where c0 is considered to be an outer context. McCarthy's model already supports the concept of inheritance. The main focus of Giunchiglia's approach, sometimes referred to as Multicontext Systems, is less on context modeling than on context reasoning [18, 17]. He take a context to be that specific subset of the complete state of an individual entity that is used for reasoning about a given goal; it is seen as a (partial) theory of the world which encodes an individual's subjective perspective about it.

Another early representative of this kind of approach is the Extended Situation Theory by Akman and Surav [2]. As the name implies it extends the Situation Theory which has been proposed by Barwise and Perry [4]. Barwise and Perry tried to cover model-theoretic semantics of natural language in a formal logic system. Akman and Surav used and extended this system to model the context with situation types which are ordinary situations and thus first-class objects of situation theory. The variety of different contexts is addressed in form of rules and presuppositions related to a particular point of view. They represent the facts related to a particular context with parameter-free expressions supported by the situation type which corresponds to the context. The figure below shows a short example of how the rules of a context are represented as constraints in their approach.

$$S_1 = [\dot{s} \mid \dot{s} \models\!\!\ll bird, \dot{a}, 1 \gg]$$
$$S_2 = [\dot{s} \mid \dot{s} \models\!\!\ll flies, \dot{a}, 1 \gg]$$
$$B \models\!\!\ll present, air, 1 \gg \wedge \ll penguin, \dot{a}, 0 \gg \wedge \ldots$$
$$C = S_1 \Rightarrow S_2 \mid B$$

Context Expression from Extended Situation Theory

A similar approach is the Sensed Context Model proposed by Gray and Salber [19]. They use first-order predicate logic as a formal representation of contextual propositions and relations.

Another approach within this category is the multimedia system by Bacon et al. [3]. In this system the location as one aspect of the context is expressed as facts in a rule based system. The system itself is implemented in Prolog.

A.3.5 Object Oriented Models

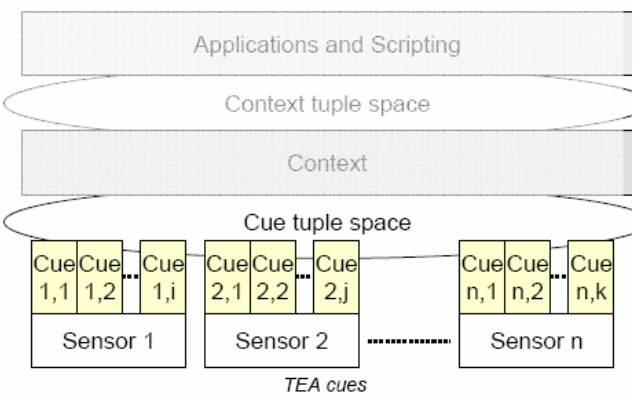
Common to object oriented context modeling approaches is the *intention to employ the main benefits of any object oriented approach - namely encapsulation and reusability* – to cover parts of the problems arising from the dynamics of the context in ubiquitous environments. The details of context processing is encapsulated on an object level and hence hidden to other components. Access to contextual information is provided through specified interfaces only.

A representative for this kind of approach is the *cues* [37] developed within the TEA project [1, 38]. The concept of cues provides an abstraction from physical and logical sensors.

A cue is regarded as a function taking the value of a single physical or logical sensor up to a certain time as input and providing a symbolic or sub-symbolic output. A finite or infinite set of possible values is defined for each cue. The output of each cue depends on a single sensor, but different cues may be based on the same sensors. The context is modeled as an abstraction level on top of the available cues.

Thus the cues are objects providing contextual information through their interfaces, hiding the details of determining the output values.
Another approach within the object category is the *Active Object Model* of the GUIDE project [13]. Again, the chosen approach has been primarily driven by the requirement of being able to manage a great variety of personal and environmental contextual information while maintaining scalability. All the details of data collection and fusing (e.g. the context adaptive composition of HTML fragments) are encapsulated within the active objects and thus hidden to other components of the system.

The approach of Bouzy and Cazenave [6] followed a similar intention: They propose to use general object oriented mechanisms to represent contextual knowledge about temporal, goal, spatial and global contexts in computer Go (a 4000 years old game that is very famous in Japan, China and Korea). They justified their object oriented context modeling approach with its inheritance and reutilization capabilities, allowing "defining the smallest number of properties, functions and rules in order to simplify knowledge representation in very complex domains and systems".

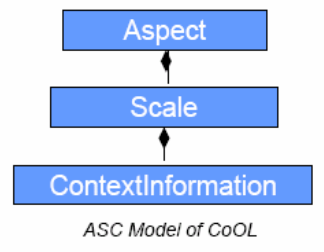


A.3.6 Ontology Based Models

Ontologies are a promising instrument to specify concepts and interrelations [43, 20]. They are particularly suitable to project parts of the information describing and being used in our daily life onto a data structure utilizable by computers.

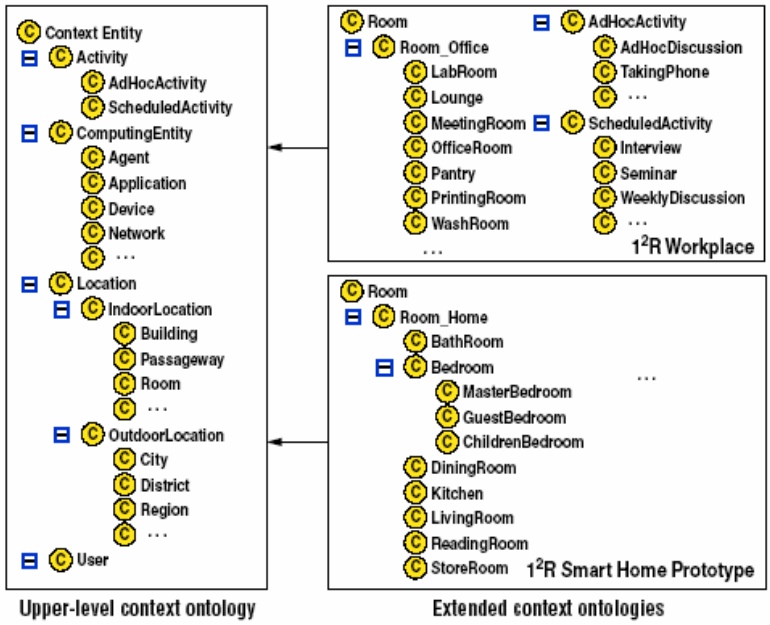
One of the first approaches of modeling the context with ontologies has been proposed by ¨ Otzt¨urk and Aamodt [31]. They analysed psychological studies on the difference between *recall* and *recognition* of several issues in combination with contextual information. From this examination they derived the necessity of normalizing and combining the knowledge from different domains. They proposed a context model based on ontologies due to their strengths in the field of normalization and formality.

Another approach within the ontology category has been proposed as the *Aspect-Scale-ContextInformation (ASC)* model [39]. Using ontologies provides an uniform way for specifying the model's core concepts as well as an arbitrary amount of subconcepts and facts, altogether enabling contextual knowledge sharing and reuse in an ubiquitous computing system [15]. This contextual knowledge is evaluated using ontology reasoners. The model has been implemented applying selected ontology languages. These implementations build up the core of a non monolithic *Context Ontology Language (CoOL)*, which is supplemented by integration elements such as scheme extensions for Web Services and others [41, 40]. Beyond determination of service interoperability in terms of contextual compatibility and substitutability, this language is used to support context-awareness in distributed service frameworks for various applications. For instance a contextual motivated *non-carrier service handover* is presented as one of the applications [42].



ASC Model of CoOL

The CONON context modeling approach by Wang et al. [21, 45] is based on the same idea of the ASC/CoOL approach, namely to develop a context model based on ontologies because of its knowledge sharing, logic inferencing and knowledge reuse capabilities. Wang et al. created an upper ontology which captures general features of basic contextual entities and a collection of domain specific ontologies and their features in each subdomain. The CONON ontologies are serialized in OWL-DL which has a semantic equivalence to well researched description logics. This allows for consistency checking and contextual reasoning using inference engines developed for description languages.



The CONON Ontology Model

A promising emerging context modeling approach based on ontologies is the CoBrA system [11]. This system provides a set of ontological concepts to characterize entities such as persons, places or several other kinds of objects within their contexts. The CoBrA system uses a broker-centric agent architecture

to provide runtime support for context-aware systems, particularly in *Intelligent Meeting Rooms*, a prevalent scenario of an ubiquitous computing environment.

A.4. Evaluation and Comparison

Beside inefficiencies in describing complex contextual information as mentioned in 3.1, it is common to all key-value models, that they are weak on the requirements 1 to 5. Distributed composition and the handling of incompleteness is possible on the instance level only. There is no scheme or at least range definitions available to check against, making partial validation a difficult task and any kind of matching algorithm error-prone at runtime. The simplicity of key-value pairs is an advance from the management and error risk perspective, but it is a drawback if quality meta-information or ambiguity shall be considered. Solely the applicability to existing ubiquitous computing environments is strength of this kind of context modeling approach.

Modeling Approach	Standard Retrieval Method	Level of formality	Distributed Composition	Partial Validation	Quality of Information	Incomplete-ness & Uncertainty	Applic-ability
Key-Value Models	Linear Search	–	–	–	–	–	+
Markup Scheme Models	Markup Query Language	+	+	++	–	–	++
Graphical Models	Transform-ation	+	–	–	+	–	+
Logic Based Models	Inference	++	++	–	–	–	–
Object Oriented Models	Algorithm	+	++	+	+	+	+
Ontology Based Models	Reasoning	++	++	++	+	+	+

Markup scheme models (section A.3.2) are strong concerning the partial validation requirement. There usually exists a scheme definition and a set of validation tools which can be used for type checking, even for complex types. Range checking is also possible to some degree for numerical values. But incompleteness and ambiguity have to be handled proprietary on the application level. If and how far the distributed composition requirement is met depends on the single approach.

Standard CC/PP and UAProf have only restricted overriding and merging mechanisms which are required for distributed composition. This weakness is addressed within the CSCP approach by providing more flexible overriding and merging mechanisms. It is worth mentioning that Indulska et al. [27] as well as Butler [8] made negative experiences with CC/PP and UAProf based context models because of the constraints imposed by the XML serialization respectively the representation in RDF. Furthermore, they identified flaws in the design of CC/PP itself, for instance pertaining to the method of updating values or regarding the absence of relational constraints.

Another drawback concerning distributed composition has to be tackled if Document Type Definitions (DTDs) are used on the markup structuring level - they do not provide overriding or merging. Quality meta-information may be added to contextual information at any level of the markup data. As far as visible, this is done to some degree in the CSCP approach, the CC/PP Context Extension approach as well as the PPDL approach. A comprehensive scheme definition is a step towards a high level formality and thus may
be used to determine interoperability. Applicability to existing markup-centric infrastructures of an ubiquitous computing environment (e.g. Web Services) is a strength of this kind of context modeling approach.

The strengths of graphical models as described in A.3.3 are definitely on the structure level. They are mainly used to describe the structure of contextual knowledge and derive some code (Bauer's approach) or an ER-model (Henricksen's approach) from the model, which is valuable in the sense of the applicability requirement. The distributed composition requirement has some constraints on the structure level, because the merging of model fragments is less efficient than the merging of instance data. Partial validation is possible. Incompleteness and ambiguity seem to be unconsidered by Bauer, but are addressed by Henricksen in a revised version of their model [24]. Most of the extensions made by Henricksen to ORM are quality labels so that quality meta-information may be considered to be intrinsic to that approach. The level of computer evaluable formality is usually relatively low for any graphical model. It is mainly used for human structuring purposes.

Object oriented context modeling approaches (section A.3.4) are strong regarding the distributed composition requirement. New types of contextual information (classes) as well as new or updated instances (objects) may be handled in the system in a distributed fashion. Partial validation is possible, typically using a compiler on the structure level and a runtime environment on the instance level. The TEA approach is safe concerning the quality of information requirement, because the concept of cues contains a parameter describing the quality of the

cue's output symbol. This is useful to handle incompleteness and ambiguity correctly. A higher level of formality is reached through the use of well-defined interfaces to access the object's content, but the invisibility as consequence of encapsulation is a little drawback concerning the formality requirement. Applicability to existing object oriented ubiquitous computing runtime environments is given, but has usually strong additional requirements on the resources of the computing devices – requirements which often cannot be fulfilled in ubiquitous computing systems.

Logic based context models (see A.3.5) may be composed distributed, but partial validation is difficult to maintain. Their level of formality is extremely high, but without partial validation the specification of contextual knowledge within a logic based context model is very error-prone. None of the logic based models within our survey seem to fulfill the quality of information requirement, even if it should be easy to add quality meta-information. Incompleteness and ambiguity seem to be addressed neither. Applicability to existing ubiquitous computing environments seems to be a major issue, because full logic reasoners are usually not available on ubiquitous computing devices.

Due to the similarities between the modeling instruments of ontologies (concepts, facts) and objects (classes, instances), ontology based context modeling approaches described in A.3.6 are also strong regarding the distributed composition requirement. Partial validation is possible, and a comprehensive set of validation tools do exist. The ASC model seems to be the only model of the survey enabling not only data type validation, but also full data content validation by specifying ranges for the contextual information called *scales*.

All ontology based context models inherit the strengths in the field of normalization and formality from ontologies. The ASC model and the CONON model inherently support quality meta-information and ambiguity, whereas the CoBrA approach seems not to do so, but could be easily extended in that way. Incompleteness is covered by all approaches in a similar way. Applicability to different existing ubiquitous computing environments is reached in the ASC model approach adopting integration elements of CoOL such as scheme extensions. The applicability of the CONON ontologies is without any further integration elements restricted to environments capable of handling OWL-DL for knowledge representation purposes. Due to its broker-centric agent architecture the CoBrA approach is particularly applicable to agent systems.

Due to our analysis we arrived at the conclusion that the most promising assets for context modeling for ubiquitous computing environments with respect to the

requirements listed in section 2 can be found in the ontology category. The representatives of this category met the requirements best.
However, this does not mean that the other approaches are unsuitable for ubiquitous computing environments.
As with all surveys, the list of context modeling approaches is comprehensive but incomplete. Further emerging approaches should be considered and evaluated in a similar way.

REFERENCES

[1] Esprit project 26900: Technology for enabled awareness (tea), 1998.
[2] AKMAN, V., AND SURAV, M. The use of situation theory in context modeling. *Computational Intelligence 13*, 3 (1997), 427–438.
[3] BACON, J., BATES, J., AND HALLS, D. Location-oriented multimedia. *IEEE Personal Communications 4*, 5 (1997).
[4] BARWISE, J., AND PERRY, J. *Situations and Attitudes*. MIT Press, 1983.
[5] BAUER, J. Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic,
Mar. 2003. Diplomarbeit.
[6] BOUZY, B., AND CAZENAVE, T. Using the Object Oriented Paradigm to Model Context in Computer Go.
In *Proceedings of Context'97* (Rio, Brazil, 1997).
[7] BROWN, P. J., BOVEY, J. D., AND CHEN, X. Context-aware Applications: from the Laboratory to
the Marketplace. *IEEE Personal Communications 4*, 5 (October 1997), 58–64.
[8] BUTLER, M. H. CC/PP and UAProf: Issues, improvements and future directions. In *Proceedings of*
W3C Delivery Context Workshop (DIWS 2002) (Sophia-Antipolis/France, March 2002).
[9] CAPRA, L., EMMERICH, W., AND MASCOLO, C. Reflective middleware solutions for context-aware applications, 2001.
[10] CHEN, G., AND KOTZ, D. A survey of context-aware mobile computing research. Tech. Rep. TR2000-381, Dartmouth, November 2000.
[11] CHEN, H., FININ, T., AND JOSHI, A. Using OWL in a Pervasive Computing Broker. In *Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003)* (2003).
[12] CHEN, P.-S. The entity-relationship model - toward a unified view of data. *ACM Transaction on Database Systems 1*, 1 (March 1976), 9–36.
[13] CHEVERST, K., MITCHELL, K., AND DAVIES, N. Design of an object model for a context sensitive tourist GUIDE. *Computers and Graphics 23*, 6 (1999), 883–891.

[14] CHTCHERBINA, E., AND FRANZ, M. Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment. In *Proceedings of International*
Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w) (L'Aquila/Italy, January 2003).
[15] DE BRUIJN, J. Using Ontologies – Enabling Knowledge Sharing and Reuse on the Semantic Web.
Tech. Rep. Technical Report DERI-2003-10-29, Digital Enterprise Research Institute (DERI), Austria,
October 2003.
[16] DEY, A. K. Understanding and using context. *Personal and Ubiquitous Computing, Special issue on*
Situated Interaction and Ubiquitous Computing 5, 1 (2001).
[17] GHIDINI, C., AND GIUNCHIGLIA, F. Local models semantics, or contextual reasoning locality compatibility. *Artificial Intelligence 127*, 2 (2001), 221–259.
[18] GIUNCHIGLIA, F. Contextual reasoning *Epistemologica - Special Issue on I Linguaggi e le*
Macchine 16 (1993), 345–364. Also IRST-Technical Report 9211-20, IRST, Trento, Italy.
[19] GRAY, P., AND SALBER, D. Modelling and Using Sensed Context Information in the design of
Interactive Applications. In *LNCS 2254: Proceedings of 8th IFIP International Conference on Engineering*
for Human-Computer Interaction (EHCI 2001) (Toronto/Canada, May 2001), M. R. Little and
L. Nigay, Eds., Lecture Notes in Computer Science (LNCS), Springer, p. 317 ff.
[20] GRUBER, T. G. A translation approach to portable ontologies. *Knowledge Acquisition 5*, 2 (1993),
199–220.
[21] GU, T., WANG, X. H., PUNG, H. K., AND ZHANG, D. Q. Ontology Based Context Modeling and
Reasoning using OWL. In *Proceedings of the 2004 Communication Networks and Distributed Systems*
Modeling and Simulation Conference (CNDS2004) (San Diego, CA, USA, January 2004).
[22] HALPIN, T. A. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufman Publishers, San Francisco, 2001.
[23] HELD, A., BUCHHOLZ, S., AND SCHILL, A. Modeling of context information for pervasive computing applications. In *Proceedings of SCI 2002/ISAS 2002* (2002).

[24] HENRICKSEN, K., AND INDULSKA, J. Modelling and Using Imperfect Context Information. In *Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004)* (Orlando, FL, USA, March 2004), pp. 33–37.

[25] HENRICKSEN, K., INDULSKA, J., AND RAKOTONIRAINY, A. Modeling context information in pervasive computing systems. In *LNCS 2414: Proceedings of 1st International Conference on Pervasive Computing* (Zurich, Switzerland, 2002), F. Mattern and M. Naghshineh, Eds., Lecture Notes in Computer Science (LNCS), Springer, pp. 167–180.

[26] HENRICKSEN, K., INDULSKA, J., AND RAKOTONIRAINY, A. Generating Context Management Infrastructure from High-Level Context Models. In *Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), pp. 1–6.

[27] INDULSKA, J., ROBINSONA, R., RAKOTONIRAINY, A., AND HENRICKSEN, K. Experiences in using cc/pp in context-aware systems. In *LNCS 2574: Proceedings of the 4th International Conference on*
Mobile Data Management (MDM2003) (Melbourne/Australia, January 2003), M.-S. Chen, P. K. Chrysanthis, M. Sloman, and A. Zaslavsky, Eds., Lecture Notes in Computer Science (LNCS), Springer,
pp. 247–261.

[28] KAGAL, L., KOROLEV, V., CHEN, H., JOSHI, A., AND FININ, T. Project centaurus: A framework for indoor mobile services.

[29] MCCARTHY, J. Notes on formalizing contexts. In *Proceedings of the Thirteenth International Joint*
Conference on Artificial Intelligence (San Mateo, California, 1993), R. Bajcsy, Ed., Morgan Kaufmann,
pp. 555–560.

[30] MCCARTHY, J., AND BUVA ˇC. Formalizing context (expanded notes). In *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language* (Menlo Park, California, 1997), S. Buvaˇc and Ł. Iwa´nska, Eds., American Association for Artificial Intelligence, American Association for Artificial Intelligence, pp. 99–135.

[31] ¨OTZT ¨U RK, P., AND AAMODT, A. Towards a model of context for case-based diagnostic problem solving. In *Context-97; Proceedings of the interdisciplinary conference on modeling and using context* (Rio de Janeiro, February 1997), pp. 198–208.

[32] PASCOE, J. Adding Generic Contextual Capabilities to Wearable Computers. In *2nd International*
Symposium on Wearable Computers (ISWC 1998) (1998), pp. 92–99.

[33] RYAN, N. ConteXtML: Exchanging Contextual Information between a Mobile Client and the
FieldNote Server, August 1999.

[34] SAMULOWITZ, M., MICHAHELLES, F., AND LINNHOFF-POPIEN, C. Capeus: An architecture for context-aware selection and execution of services. In *New developments in distributed applications and interoperable systems* (Krakow, Poland, September 17-19 2001), Kluwer Academic Publishers, pp. 23–39.

[35] SCHILIT, B. N., ADAMS, N. L., AND WANT, R. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, US, 1994).

[36] SCHILIT, W. N. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[37] SCHMIDT, A., BEIGL, M., AND GELLERSEN, H.-W. There is more to context than location. *Computers and Graphics 23*, 6 (1999), 893–901.

[38] SCHMIDT, A., AND LAERHOVEN, K. V. How to Build Smart Appliances. *IEEE Personal Communications* (August 2001).

[39] STRANG, T. *Service Interoperability in Ubiquitous Computing Environments*. PhD thesis, Ludwig-Maximilians-University Munich, Oct. 2003.

[40] STRANG, T., LINNHOFF-POPIEN, C., AND FRANK, K. Applications of a Context Ontology Language. In *Proceedings of International Conference on Software, Telecommunications and Computer Networks (SoftCom2003)* (Split/Croatia, Venice/Italy, Ancona/Italy, Dubrovnik/Croatia, October 2003), D. Begusic and N. Rozic, Eds., Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, Croatia, pp. 14–18.

[41] STRANG, T., LINNHOFF-POPIEN, C., AND FRANK, K. 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)* (Paris/France, November 2003), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 236–247.

[42] STRANG, T., LINNHOFF-POPIEN, C., AND ROECKL, M. Highlevel Service Handover through a Contextual Framework. In *Proceedings of 8th International Workshop on Mobile Multimedia Communications (MoMuC2003)* (Munich/Germany, October 2003), J. Kaefer and M. Zuendt, Eds., vol. 1, Center for Digital Technology and Management (CDTM), pp. 405–410.

[43] USCHOLD, M., AND GR ¨UNINGER, M. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review 11*, 2 (1996), 93–155.

[44] W3C. Composite Capabilities / Preferences Profile (CC/PP). http://www.w3.org/Mo- bile/CCPP.

[45] WANG, X. H., ZHANG, D. Q., GU, T., AND PUNG, H. K. Ontology Based Context Modeling and Reasoning using OWL. In *Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004)* (Orlando, FL, USA, March 2004), pp. 18–22.
[46] WAPFORUM. User Agent Profile (UAProf). http://www.wapforum.org.

Appendix B

SOME SAMPLE REASONING MECHANISMS

B.1. Context Reasoning using Description Logic (DL) [1]

Description Logic (DL) allows specifying a terminological hierarchy using a restricted set of FOL (first order logic) formulas. The equivalence of OWL and description logics allows OWL to exploit the considerable existing body of DL reasoning including class consistency and consumption, and other ontological reasoning. The formal semantics entailed by description logic can be used to automatically reason about the context knowledge to fulfill important logical requirements. These requirements include concept *satisfiability* (whether concept can exist), class *subsumption* (whether class C is a sub-class of class D), *class consistency* (whether all the definitions of classes, properties, and relations in a context ontology are satisfiable), and instance checking (whether a context instance is satisfied to the context ontology).

The following is an example for Description Logic Rules to infer user location. If we have the rules that *locatedIn* has a *transitiveProperty*, and *isPartOf* is *subPropertyOf locatedIn*,

```
<owl:ObjectProperty rdf:ID="locatedIn">
<rdf:type rdf:resource="&owl;TransitiveProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isPartOf">
<rdfs:subPropertyOf rdf:resource="#locatedIn" />
</owl:ObjectProperty>
```

And if we have the location of a user Bilbo is in bedRoom

```
<Room rdf:ID="bedRoom">
<isPartOf rdf:resource="#home"/>
</Room>
<Agent rdf:ID="Bilbo">
<locatedIn  rdf:resource="#bedRoom"/>
</Agent>
```

Then we can can infer bedRoom is in home:

```
<Room rdf:ID="bedRoom">
<locatedIn rdf:resource="#home"/>
</Room>
```

And we can can also infer Bilbo is at home,

<Agent rdf:ID="Bilbo">
<locatedIn rdf:resource="#home"/>
</Agent>

B.2. Context Reasoning using First-Order Logic (FOL)

The user-centric design rationale of context aware systems requires more flexible logic reasoning mechanisms than description logic reasoning. By customizing reasoning rules within the entailment of first-order logic (actually a subset of first order logic, as OWL is less expressive than FOL), a wide range of high-level, conceptual context such as "what the user is doing" can be deduced from low-level context.

The following is a sample rule that infer a user's likely situation based on context, activity, location, and computing entity.

type(?user, User), type(?event, Meeting), location(?event, ?room), locatedIn(?user, ?room),
startDateTime(?event, ?t1),
endDateTime(?event, ?t2), lessThan(?t1, currentDateTime()),
greaterThan(?t2, currentDateTime())
=>situation(?user, AtMeeting)

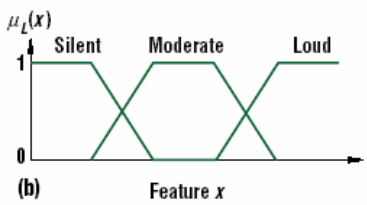
B.3. Context Reasoning using Probabilistic and Fuzzy Logic

Probabilistic logic lets us write rules that reason about events' probabilities in terms of the probabilities of other related events. An example rule (written in XSB) is

prob(X, Y, union, P) :- prob(X, Q), prob(Y, R), disjoint(X, Y), (P is Q + R).

This rule essentially says that Pr(X U Y) = Pr(X) + Pr(Y) if X and Y are disjoint events, that is, they never occur together. X and Y can be context predicates. For example, X could be *location(Bob, in, Room 2401)* and Y could be *location(Bob, in, Room 3234)*. X and Y are disjoint events, since Bob can't be in two different locations at the same time [2].

Fuzzy logic is somewhat similar to probabilistic logic. In fuzzy logic, confidence values represent degrees of membership rather than probability. Fuzzy logic is useful in capturing and representing imprecise notions such as "tall," "trustworthy," and "confidence" and reasoning about them.

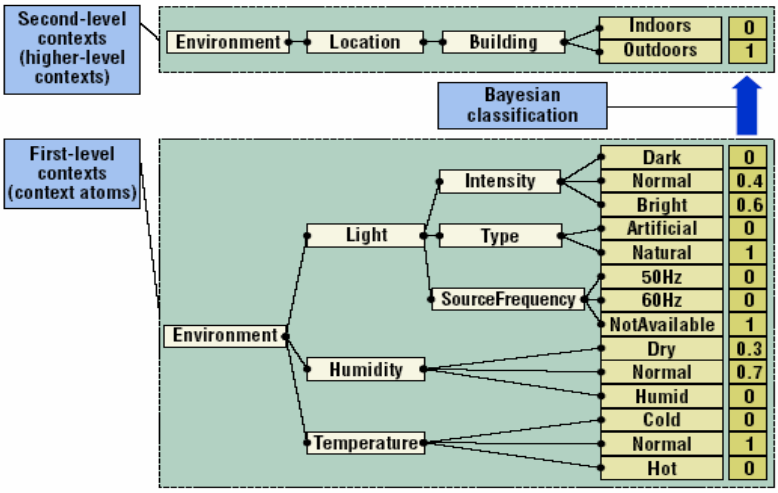


An example of Fuzzy Logic for audio feature [3]

For example, in quantizing environment sound intensity, the quantization divides the processed feature into three quantities - Silent, Moderate, and Loud - corresponding to the three membership functions denoted as $\mu_L(x)$. Apply a fuzzy set for features, resulting in continuous valued fuzzy labeling (Figure above). For example, $\mu_L(x) = 0.7$ / Silent + 0.3 / Moderate + 0 / Loud [3].

B.4. Context Reasoning using Bayesian Networks

Bayesian networks are *directed acyclic graphs*, where the nodes are random variables representing various events and the arcs between nodes represent causal relationships. Bayesian networks are a powerful way of handling uncertainty in context data, especially when there are causal relationships between various events. They are useful for performing probabilistic sensor fusion and higher-level context derivation. In general, root nodes in the Bayesian network represent the information to be deduced, while the leaves are sensed information. The intermediate nodes are important sub-goals that are helpful to the deduction process.



An example of Bayesian Network to deduce the location (indoor/outdoor)

Figure above introduces the formation of the higher level context *Outdoors* from the *context atoms* (or low level sensed contexts) using a naïve Bayesian network [3]. White rectangular boxes represent types and light tan boxes represent context values. Dark tan boxes contain the corresponding confidence instance values for the current situation. The naïve Bayesian network classifies the confidence instance values into one of the previously defined output classes, Indoors and Outdoors in this network.

The naïve Bayes classifier works well for online context inference and sensor fusion mainly because 1) It has proven robust even with missing, uncertain, and incomplete information.5. 2) It is computationally efficient. Training and inference both have a linear complexity in input data size. And 3) It requires no background information modeling except for choosing the relevant network inputs.

REFERENCES

[1] Baadar, F., Horrocks, I., Sattler, U.: Description Logics. Handbook on Ontologies. (2004) 3-28

[2] Anand Ranganathan, Roy H. Campbell: A Middleware for Context -Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference, Brazil, June, 2003.

[3] Korpipaa, P.; Mantyjarvi, J.; Kela, J.; Keranen, H.; Malm, E.J. Managing context information in mobile devices.; Pervasive Computing, IEEE , Volume: 2, Issue: 3 , July-Sept. 2003 Pages:42 – 51