Steps Towards Creating a Context-Aware Software Agent System

Harry Chen, Sovrin Tolia
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-231
September 27th , 2001*

context-aware, software agents, CoolTown, CoolAgent

The ability to reason from context is crucial to human communication. It allows a large amount of implicit information to be conveyed with a small amount of explicit description. If the same kind of ability can be provided to software agents, then these agents, even with little built-in knowledge, can adjust their behaviors according to the available implicit information in the environment.

This document describes our research in prototyping the CoolAgent Recommendation System (CoolAgent RS). This context-aware software agent system demonstrates the ability to allow contextual information to be freely distributed among agents so that the meaning of that information can be shared and understood. It provides a flexible infrastructure for agents to capture heterogeneous contextual information in the physical world, and represent that uniformly for machine-processes. It also allows agents to negotiate with other agents in the vicinity for contextual information that is not directly accessible through sensing.

* Internal Accession Date Only Approved for External Publication

Contents

Chapter 1

Introduction

The ability to reason from context is crucial to human communication. It allows a large amount of implicit information to be conveyed with a small amount of explicit description. If the same kind of ability can be provided to software agents, then agents, even with a small amount of built-in knowledge, can adjust their behaviors according to the available implicit information in the environment.

In the past many stand-alone context-aware agents and applications [1, 2] have been developed to provide ubiquitious services to users by taking advantage of the contextual information in the environment. As we foresee the increase in the complexity of future computing environments, we believe it is more suitable to move from the monolithic context-aware agent paradigm to a distributed agent architecture, in which agents can better interoperate and share contexts.

In this document, we describe our research in prototyping the CoolAgent Recommendation System (CoolAgent RS). This context-aware software agent system demonstrates the ability to allow contextual information to be freely distributed among agents so that the meaning of that information can be shared and understood. It provides an infrastructure for agents to capture heterogeneous contextual information in the physical world, and represent them uniformly for machine-processes. It also allows agents to obtain contextual information from other agents in the vicinity.

In Chapter 2 we provide background knowledge on Context-Aware software agent research. In Chapter

3 and Chapter 4 we describe the architecture and the implementation of CoolAgent RS that we have implemented. In Chapter 5 we give a brief overview of the demo scenario.In Chapter 6 we discuss our experiences in the course of implementing CoolAgent RS, and propose future research work.

Chapter 2

Problem Domain

In this chapter, we provide a brief introduction to context and context-awareness (or context-aware computing). In particular, we emphasize our discussion on context-aware software agent system.

2.1 Context

According to the *Oxford English Dictionary* the term *context* has two primary meanings: 1) the words around a word, phrase, or statement which are often used to help explain (fix) the meaning; 2) the general conditions (circumstances) in which an event, or action takes place. Clearly, the first meaning is closely related to linguistic meaning and linguists' use of the term, whereas the second meaning is closer to the definition of context in Computer Science [3].

Based on the second meaning, we define context as any information that can be used to characterize the situation of an entity or an action. For example, while you are reading this document, the entity context includes the document in your hands and the chair you are sitting on. On the other hand, the same document and the same chair are also parts of your reading action – the context of an action.

It is important to note that the meaning of the contextual information does not change, while our interpretation of them can change. For example, the document that you are holding right now is the same

document in your reading context as the document in the room context (assuming that you are reading this document in a room). Nevertheless, we can have totally different interpretations of the same document: we can think of this document being a stack of papers, or we can think of this document being a written artifact.

This is an important property of context because, as we will discuss it more in details in the later chapters, the persistent meaning of contextual information allows the same piece of information to be used in distinct domain reasoning processes.

2.2 Context-Awareness

We humans are inherently context-aware agents. When we talk with each other, we are able to use implicit situational information, or *context*, to increase the conversational bandwidth [4]. For example, when a person in the room tells you, "close the door please," you are able to reason that the person is asking you to close the door in this room, not the one in the next room, even though the person did not explicitly specify which door he/she wants you to close.

The fact that you are able to choose the correct door to close is not a simple process. First of all, you and the person in room have to share a common communication language and share the meaning of vocabularies in the conversation. This requires a common ontology. Secondly, the person needs to be able to physically communicate with you, in this case by using sound. Thirdly, you need to be able make sense out of what you have heard, that is the ability to reason. Only after you have understood the semantics of the request, based on your interpretation, can you take the action to choose to close the door in the room.

We believe the following are the three necessary capabilities that enable humans to become context-aware: 1) the capability to share a common ontology, 2) the capability to sense context and 3) the capability to reason over context.

2.3 Context-Aware Software Agents

From the design point of view, a context-aware software agent is simply a way to model a context-aware application based on the agent theories. Similar to the traditional context-aware application, the context-aware agent is designed to make use of context to provide task-relevant information and services to users.

Perhaps because of the primitive programming paradigm that traditional context-aware systems are built from, often these systems do not emphasize the capability to reason and share ontologies. Most of these systems provide the capability to sense a variety of contextual information in the environment through a collection of sophisticated sensors.

Unfortunately, these systems are like retarded humans. They have sophisticated sensing capabilities, but lack the ability to understand semantics and reason. Naturally, the functionality of these systems is limited by the amount of information that can be directly conveyed from the sensors.

On the other hand, the design of our context-aware agents explores the potential to increase context-awareness by taking advantage of shared ontologies and reasoning.

Chapter 3

CoolAgent RS: A Context-Aware Extension to CoolAgent

The CoolAgent RS project aims to build a Context-Aware Agent System which serves as an extension to the existing CoolAgent Meeting Management Prototype. The goal of this project is to develop a proof-of-concept system to

- Validate the importance of introducing context-awareness in an agent system
- Demonstrate improvements in the overall system interoperatibility and flexibility
- Show enhanced reasoning capabilities

3.1 Functional Requirements

The functional requirements of the system are the following:

- It should be able to reason about the type of event occurring at a particular venue
- During a meeting event, it should recommend relevant documents to the participants

- During a lunch event, it should recommend cuisines that match the personal profiles of the people present in the venue
- People presence in the venue should be automatically detected by the system

In order to fulfill the above requirements, we require contextual 2.1 information and to this end, the following pieces of information are used:

- Time Information: What time is it? Are we expecting any scheduled event at this moment?
- Content of the Event: Is this a Meeting Event or a Lunch Event? What is the meeting about? Is this meeting about Project X? Is this meeting related to any other projects or topics? Is this meeting related to any other previous event?
- Presences of Participants: Who is currently attending an event? Who is expected to attend this meeting? Are the key people of this meeting present?
- Personal and Professional Background information of participants: Who are the participants? How do they relate to the subject of the meeting? What are their job titles? What are their nationalities?
- Relationships among present participants: Is participant X a co-worker of participant Y? Is participant X working on the same project as participant Y? Is participant X the manager of participant Y?

Answers to some of the questions listed above may seem trivial as to how they could be used to enrich the recommendation reasoning, but some of them are not. The nontrivial ones include Who is currently attending the meeting?, Is participant X a Co-worker of participant Y?, Is this meeting related to any other project/meeting?

It is quite true that, if we consider the answers to these questions individually, they have little or nothing to do with recommendation reasoning. Nevertheless, if we consider all the answers together as a piece of integrated knowledge then it allows sophisticated reasoning to be made in a way that would be impossible otherwise.

For example, The Answers to the above questions:

- Harry,Sovrin and Reed are currently attending a HP Cooltown related meeting
- Harry is a Co-worker of Sovrin
- Reed is the manager of Sovrin
- Sovrin is working on a project that is related to HP CoolTown
- Sovrin is an intern
- Harry likes Chinese food
- Sovrin likes Indian food

By default, the document recommendation system has the following basic reasoning rules built-in

i. Recommend CoolTown White Papers (CWP) to any person who is working on CoolTown related project

ii.Recommend CoolTown SDK Documents (CSD) to any person who is a software developer, including interns and whoever is working on the CoolTown related project.

iii.Recommend CoolTown Call For Proposal (CFP) to any person who has a manager position and is working on a related project

If we consider the contextual information that is provided individually, then the following recommendations are made to the participants:

- Sovrin: CWP (by rule i) and CSD (by rule ii)
- Harry: No Recommendation
- Reed: No Recommendation

If we consider all of the contextual information as a piece of integrated knowledge, then the following recommendations are made to the participants:

- Sovrin: CWP and CSD
- Harry: CWP and CSD

- Reed: CWP, CSD and CFP

The above example shows how effective and integrated use of contextual information can significantly enhance the reasoning capability of the document recommendation system.

On the other hand, the Food Recommendation System will be able to provide cuisine recommendations if the following contextual information is available:

- Harry prefers Chinese food
- Sovrin prefers Indian food
- The occurring event is Lunch Event
- Harry and Sovrin are both present at a venue
- The current time is within the boundaries of the lunch event interval at that venue

In the above example, the people presence information is used for both document and food recommendations. This demonstrates the ability for one piece of contextual information, having persistent meaning, to be used in two distinct domain reasoning processes.

Chapter 4

Implementing CoolAgent RS

The core of the CoolAgent RS is a collection of software agents that are realized using JADE (Java Agent Development Framework). These agents share common ontologies that are encoded in RDF/RDFS. The context sensing capability of these agents is provided by the CoolTown Web Presence Manager (WPM). Agent reasoning is supported by the logical programming facilities of the SICStus Prolog system and the Prolog Forward Chaining (P_{fc}) package.

In this chapter, we will describe the implementation details of the CoolAgent RS. In the next section, we will introduce the necessary background knowledge that is important to understand the CoolAgent RS implementation. In the rest of the chapter, we will describe the ontology engineering, the sensing infrastructure, and the agents in detail.

4.1 Background Knowledge

4.1.1 RDF Data Model

This section we briefly describe the Resource Description Framework (RDF). Most of the materials in this section are adapted from the RDF Model and Synatax Specification [10].

The Resource Description Framework (RDF) is a foundation for processing metadata. It provides in-

teroperatibility between applications that exchange machine-understandable information on the Web [10]. RDF itself is not a language; it is a data model for representing metadata. In the current RDF specification, RDF metadata is encoded in XML to make it possible to specify semantics for data based on XML in a standardized and interoperable manner.

The foundation of RDF is a model for representing named properties and property values. RDF properties may be thought of as attributes of resources, and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources, and an RDF model can therefore resemble an entity-relationship diagram.

The RDF data model is a syntax-neutral way of representing RDF expressions. The data model representation is used to evaluate equivalence in meaning. Two RDF expressions are equivalent if, and only if, their data model representations are the same.

The RDF data model consists of three object types:

1. *Resources*: All things being described by RDF expressions are called resources. Resources are always named by URIs plus optional anchor ids.

2. *Properties*: Properties are specific aspects, characteristics, attributes, or relations used to describe resources. Each property has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties.

3. *Statements*: Each statement consists of a specific resource, its named property, and the value of that property for that resource. These three individual parts of a statement are called, respectively, the subject, the predicate, and the object. The object of a statement (i.e., the property value) can be another resource (specified by a URI), or it can be a literal (a simple string or other primitive datatype defined by XML). The subject, predicate, and object together constitute the triple representation of the RDF statement.

It is worthwhile to note that the RDF data model has the property that, for any given resource, it can be the subject of one triple statement, and the object of another triple statement, at the same time. For

```
<Document rdf:about="docInst"/>
<Person rdf:about="personInst">
    <holding rdf:resource="docInst"/>
<Person>
<InRoom rdf:about="inroomInst">
    <person rdf:resource="personInst"/>
</InRoom>
```

Figure 4.1: A sample RDF statement

example, Figure 4.1 shows a sample RDF statement. Two of the corresponding triple statements are the following:

```
T1: Subject(inroomInst), Predicate(person), Object(personInst)
T2: Subject(personInst), Predicate(holding), Object(docInst)
```

The Person instance, `personInst` is the object of the triple statement T1. It is also the subject of the triple statement T2.

This property allows flexible information aggregration. For example, one information source might make the statement that "a person is holding a document" (see Figure 4.2). Another information source might make the statement that "a person is in a room" (see Figure 4.3). By aggregating these two statements, one can draw the conclusion that the person who is in the room is also holding a document. Furthermore, one can reason that the document is also in the room, since the person who is holding it is in the room.

```
<Document rdf:about="docInst"/>
<Person rdf:about="personInst">
    <holding rdf:resource="docInst"/>
<Person>
```

Figure 4.2: A RDF statement: a person is holding a document

```
<InRoom rdf:about="inroomInst">
    <person rdf:resource="personInst"/>
</InRoom>
```

Figure 4.3: A RDF statement: a person is in a room.

4.1.2 Prolog Forward Chaining

The P_{fc} system [9] is a package that provides a forward reasoning capability to be used together with conventional Prolog programs. The P_{fc} inference rules are Prolog terms which are asserted as facts into the regular Prolog database.

4.1.3 Reasoning Over RDF Statements Using P_{fc}

To reason over RDF statements using Prolog rules requires the following steps:

1. Convert RDF statements that are encoded in XML into triple statements in Prolog terms using RDF parsers.

2. Construct P_{fc}rules that map related triple statements into desired Prolog terms, called instance terms, that are to be used in reasoning rules. If the reasoning rules are backward chaining Prolog rules, then the instance terms appear in the body of the rules. If the reasoning rules are forward chaining P_{fc}rules, then the instance terms appear in the left-hand side of the rules.

3. Construct reasoning rules based on the defined instance terms. Reasoning rules can be defined as either forward chaining P_{fc}rules or backward chaining Prolog rules.

4.1.4 Rule Shipping Technique (RST)

Communication is the key for information to be conveyed among agents. With the dynamic nature of a multi-agent system, an agent simply cannot anticipate all of the other agents that it might communicate with in its life span.

This is particularly true in a context-aware software agent system because interpretations of contextual information can vary, therefore, it is difficult for an agent to anticipate all possible means to provide contextual information to other agents in the system.

For example, a venue agent may have the same contextual information that is described in Figure 4.1, "a person is in a room and that person is holding a document." It is quite possible that some agent in the

system may ask the venue agent about people present in the room, and some other agent may want to know if there is any person holding anything.

It clear that these are not the only two possible contextual interpretations that can be drawn from the venue agent's knowledge. Therefore, it is not feasible for the venue agent to be built with all possible means to provide various contextual information that may be required by other agents in the system.

To overcome this problem, we introduce the Rule Shipping Technique, a rule-based content message implementation technique that allows information requesting agents to define the means to generate desired information, and send these means to other agents as a part of their request messages.

For this technique to function requires the communicating agents to share a common ontology that is defined based on the RDF data model. Agents are required to have access to both the Prolog system and the P_{fc} system. In addition, these agents have a set of utility rules built-in. These rules allows the information provider agent to discover which information needs to be send out and whom this information should be sent to, such rules are executed everytime the information provider agent's knowledge base is updated.

We will illustrate this technique from the following example:

Assume that there is a venue agent who can provide contextual information of a particular room, and there is a people-tracking agent who is interested in people presence in the room.

Both agents share a common ontology that is defined based on the RDF data model. Both agents have full access to the Prolog System and P_{fc} system.

Assume the venue agent is born with the following knowledge:

```
<Document rdf:about="docInst"/>
<Person rdf:about="personInst">
    <holding rdf:resource="docInst"/>
<Person>
<InRoom rdf:about="inroomInst">
    <person rdf:resource="personInst"/>
</InRoom>
```

The corresponding triple statements of the above data model are the following:

```
T1: triple(docInst, rdf_type, Document)
```

```
T2: triple(personInst, rdf_type, Person)
T3: triple(inroomInst, rdf_type, InRoom)
T4: triple(personInst, holding, docInst)
T5: triple(inroomInst, person, personInst)
```

Note that the above triple statements are constructed as Prolog rules. The first argument is the *subject*, the second argument is the *predicate* and the third argument is *object* of the RDF triple statement.

The people-tracking agent subscribes to the Venue Agent using the following ACL message:

```
(subscribe
  :sender (agent-identifier :name people-tracking-agent)
  :receiver (set (agent-identifier :name venue-agent))
  :content
    ([prolog-rules]))

[prolog-rules] (based on the \pfc syntax):

triple(PersonInst, rdf_type, Person) => person(PersonInst).   [Rule 1]

triple(InRoomInst, rdf_type, InRoom),
triple(InRoomInst, person, PersonInst)
=> inRoom(PersonInst).                                  [Rule 2]

inRoom(PersonInst)
=> {add(shouldInform('people-tracking-agent',
                     'venue-agent', 'id-001',
                     msg(inRoom(PersonInst))))}.      [Rule 3]
```

After the venue agent has received the above message, it simply asserts these rules in the message content, without any preprocessing, into its Prolog/P_{fc} system.

Then the venue agent uses the utility rules to discover possible queued information that need to be sent out after its knowledge based has been modified. In specific, when the utility rules are executed, it checks to see if there are any facts in the form of `shouldInform(Sender, Receiver, RuleID, Msg)`. In this case, the only returned fact that has such form is the following:

```
shouldInform('people-tracking-agent','venue-agent', 'id-001',
     msg(inRoom(personInst)).
```

The above statement is added to the knowledge base when Rule 2 is true. Rules 2 is true because triple T3 and T5 are true.

In the above example, we have shown how the people tracking agent constructs the means to generate desired information from the available knowledge in the venue agent.

4.1.5 CoolTown Web Presence Manager Architecture

The basic philosophy underlying cooltown [13] is that People, Places and Things have web representations. Each web representation is an aggregation of information and services on the web, describing a real world entity. Their architecture facilitates generation of dynamic content based on user context, security policy, and relationships with other web presences. The access model of cooltown is fairly simple. Each of the components mentioned below provide methods [6] which can be accessed by HTTP interfaces. The following are the key components associated with each web entity:

Relationship Directory Component: This component handles the automatic storage and maintenance of the relationships between different web presences.

Description Component: This component stores all the relevant information about the physical world entities represented by their Web presences. The information is expressed in XML, and can be used by any client application, or another web presence, to get raw information about the entity.

Security Component: This component allows the cooltown system administrator to establish user groups, and associate different permissions with each of these groups.

Presenter Component: This component generates contents on the fly based on HTML, XML and/or WML templates, and information coming from the Directory Component, Description Component, or Security Component.

Web Presence Manager: This [7] is Cooltown's Software infrastructure that enables easy creation, maintenance, and hosting of web presences. It serves as a repository of web presences, handling client requests to access information. It serves as a focal point for two different kinds of access. On one side, sensing devices can sense physical information, and update the web presence manager with the information present in the physical world. On the other side, clients can access the web presence manager to get basic contextual information.

An Example of CoolTown's Architectural Model

Consider Harry is in Location Eureka.

- Both Harry and Eureka will have a web presence (considered as distinct entities) with the web presence manager.
- Each of these entities will have a description component giving an XML description of the entity.
- Each of these entities will have a relationship component giving details about other entities to which it is related. For example, in this case, Eureka's relationship component will say that it is related to Harry.
- Each of these entities will have a presenter component which will display details of this entity on a web page for clients.

4.1.6 JADE

JADE [8] is an agent development framework built in Java. It facilitates agent development through the use of fipa-compliant middleware and also provides a set of tools for debugging and deployment. It Supports remote configuration of the agent systems and allows agents to be moved dynamically from one machine to the other during runtime. The Agent Communication Middleware is realized primarily through Agent Management System (AMS), Agent Communication Channel(ACC) and Directory Facilitator(DF). Each Java Virtual Machine acts as a container of Agents providing the runtime environment for their execution.

The following are some of the key features that make JADE attractive to Agent Developers

- Distributed Agent Platform: The Agent Platform can be split among several hosts, and only one java virtual machine is executed on each host.
- Library of FIPA Interaction Protocols ready to be used, for example FIPA Contract-Net Protocol.
- Graphical User Interfaces to remotely manage the lifecycle of agents and their containers.

- Complete Implementation of the FIPA Communication infrastructure, ontologies, content encodings, messaging and transport mechanisms.

4.1.7 RFIDReader

The RFID Reader used in our system is a product of RFIdeas (`http://www.rfideas.com`).There are two kinds of badges that can be detected. The Passive Badges never transmit (they don't have batteries) and the Base Unit detects such badges when they are extremly close to the reader (1-8 inches). The Active Badges can transmit their identification information, thereby overcoming the proximity constraints(3-20 feet) and the base unit will detect these badges based on transmitted information.

Apart from the traditional desktop security usage, the card reader can be used for other applications, such as our own system. It can be used to detect any physical world entities identified by a proximity badge ,(mainly people), in an environment. The Key Component of usage here would be the software which interfaces with the reader.

- Using Java Communication API [11], software can be written to interface with the Serial Port on which the reader is attached.
- The Base unit is platform independant, with the only requirement being the presence of a standard RS232 interface
- The reader has the capability to communicate in both synchronous and asynchronous modes.
- The Reader supports any existing HID Proximity Card Technology [5]

4.2 Ontology Engineering

The CoolAgent RS ontology is a formal explicit description of concepts in a domain of discourse (classes or concepts), properties of each concept describing various features and attributes of the concept (slots, roles or properties), and restrictions on slots (facets or role restrictions).

This ontology plays a very important role in the CoolAgent RS. It does not only provide the means for agents to share vocabularies and semantics, but also increases the interoperatibility among agents and external systems.

Currently, the ontology of CoolAgent RS consists of 231 classes with 179 slots. Classes are modulated into 12 seperate ontology files, all of which are contained within separate RDF namespaces. The CoolAgent RS ontologies are constructed using Protege-2000, a frame-based ontology authoring tool.

The classes in the CoolAgent RS ontology capture a wide range of domain concepts, including software agents, documents, meetings, organizations, people, places, times, FIPA device ontology, HP CoolTown ontology and HP CoolAgent meeting ontology. These concepts are arranged into 11 namespaces. Concepts in the individual namespaces are stored in distinct ontology files.

The design of the ontology is based on the development process suggested in [12]. We defined the more salient concepts first, and then generalize and specialize them appropriately.

In following subsections we will breifly describe some of the key ontology files of the CoolAgent RS ontology. We will focus our discussion on the classes that are used in the current CoolAgent RS implementation.

Basic

This ontology file contains top-level general classes that are common to the lower-level ontologies. For example, `Event`, `Tangible`, `Process`, `Relationship`, etc.

Some of the key classes that are used in the existing CoolAgent RS implementation are `Agent`, `Event`, `Relationship` and `URI_String`. The class `Agent` is defined here as "one that acts". It serves as the super class for lower-level classes like `SoftwareAgent`, `Person`, etc.

The class `URI_String` represents the concepts of a string representation of the Universal Resource Indicator specified in RFC 2396.

The class *Relationship* is an abstract class that serves as the super class for all classes that define relationships between two or more classes.

Times

This ontology file contains time related classes. It provides vocabularies for describing date, time and time measuring units. It also contains temporal relationship classes that specialize classes from the basic ontology.

People

This ontology file contains people related classes and relationships. In particular it defines classes to describe person profiles and biographical information.

Some of the key classes that are used in the existing CoolAgent RS implementation are `Person`, `Role` and `Name`. The `Person` class has slots to describe various biographical information, such as age, nationality, etc. The `Role` class is an abstract concept that indicates that any person can play a role in a given context.

CoolTown

This ontology file contains classes and relationships that capture the domain ontology that is a part of the CoolTown Web Presence Manager (WPM).

The class `Web_Entity` is defined as any entity that has web presence.

The class `Web_Description` is defined as the properties and attributes of a Web Entity.

The relationship class `Is_Associated_With` defines the association between a Web Entity and its corresponding Web Description.

Place

This ontology file contains classes and relationships that are related to places. In particular, it defines vocabularies to describe places and relationships for an indoor environment.

Some of the key classes that are used in the existing CoolAgent RS implementation are `Location`, `Cafeteria` and `Is_In`.

The class `Location` is an abstract concept that defines location. It is the super class for concrete location

classes like `Indoor_Location` and `Cafeteria`.

The class `Cafeteria` is an concrete class that defines a dining area in office or school.

The relationship class `Is_In` defines the association between an entity in a particular indoor location at a given time interval.

4.3 Sensing and Capturing Contextual Information

Contextual Information by nature takes on many different forms; therefore, to capture this information requires heterogeneous sensing techniques (sensing from physical sensors and performing virtual instrumentation). It would be unwise to require all agents to be capable of performing numerous sensing techniques or integrate with numerous sensing infrastructures, when the implementations of these techniques would overweigh the implementation of their core functionalities. To this end, we apply the Proxy technique to mediate contextual information between the low-level sensing infrastructures and the high level community of agents.

The Proxy in the CoolAgent RS system acts as a gateway for contextual information (coming from the sensing infrastructure) to enter the agent subsystem. It is at this point that contextual data gets transformed into an ontology specific format, understandable by the agents. The idea is that, with the availability of more sensing infrastructures, the proxy can be made to evolve to fetch information from these diverse stores. This design makes the agent subsystem, utlizing context information coming from these sensing sources, independant of the sensing infrastructures and their associated low level details. It achieves interoperatibility by having these proxies act as a single point of contact to such sensing systems. This proxy technique, when coupled with a good knowledge representation language, allows heteregeneous contextual information to be represented uniformly for machines processes.

The sensing infrastructures are designed to gather contextual data from the physical environment using diverse sensors. The hardware sensing devices envisioned are Badge Readers, biometric readers, temperature, light and sound sensors. HP's Cooltown and Georgia Tech's Context Toolkit provide fairly robust

frameworks, achieving a good diversity in sensing capabilities.

4.4 CoolAgent RS Architecture

4.4.1 Sensing SubSystem

The sensing infrastructure in our system is realized using the RFID Badge Reader and Cooltown's Web Presence Manager.

RFID Reader: This is a software wrapper over the RFID Reader hardware device, written in Java, that redirects the information generated by the reader(badge presence) to the web presence manager. It uses the Java Communications API to read from the serial port on which the device is attached, and interacts with the web presence manager using pre-specified interfaces. The Wrapper is configured to serve a particular location, and provides information about badge presence in that location. The interaction with the WPM involves building and destroying relationships between badges and locations.

Web Presence Manager: The WPM used for prototyping conforms to the one specified in Cooltown manual [7]. It provides entity description and relationship information in XML. The Web Presence Manager is accessed through HTTP interfaces specific to the information that is required.

4.4.2 Agents Core

Design of CoolAgent RS System advocates the following basic tenets:

- It requires an ontology infrastructure that allows contextual information to be distributed so that it's meaning is also shared and understood.
- It requires a knowledge representation language that provides good support for semantic interoperability.
- It requires a communication infrastructure for contextual information sharing.
- It requires the flexibility to allow more agents to be provisioned in the system.

Implementation Features

The Salient features driving the implementation of CoolAgent RS system are:

- All Agents are aware of the common ontology leading to a uniform representation of contextual information semantics. This principle allows the same piece of contextual information to be used in two distinct domain reasoning processes by the virtue of it's meaning being shared and understood.
- The interaction between agents in the system is governed using the semantic capability and expressiveness of RDF. This principle, when coupled with the common ontology, enables the technique of rule shipping and allows reasoning over integrated contextual information (as opposed to pieces of it).
- All Agents are aware of their communicating parties. This implies that agents requiring information will be able to anticipate which other agents in the system might be able to provide partial or complete information.
- All Agents adhere to the FIPA Subscription Protocol for information exchange.
- Any Request for information will include both the means and the templates to generate it. This is the rule shipping technique described in 4.1.4. Any Agent which wants information from another agent, will subscribe to it, providing both the rules to be executed to generate data, and templates to generate information from this data.
- Each Agent in the system (other than the Proxy) has it's own inference engine. This ensures that these agents maintain their own view of the environment and colloborate with each other to build contextual information.

Implementation Overview

The Figure 4.4 shows the implementation of the CoolAgent RS system. The Web Presence Manager, Notification Agent Proxy, and the Entity Tracking Agent consititute the Proxy Architecture in the system. The

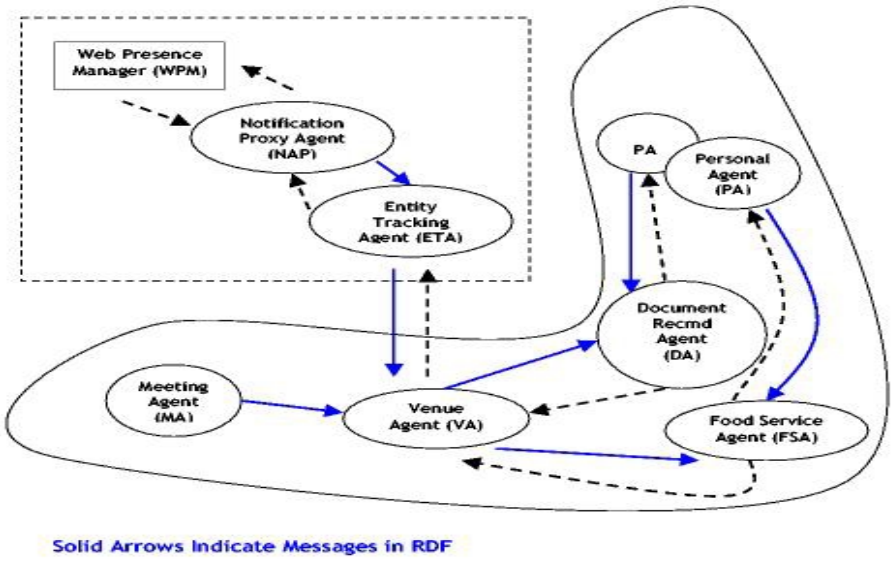

Figure 4.4: Design of Context Aware Software Agent System

Venue Agent, Meeting Agent, Food Service Agent, Document Agent, and the Personal Agent constitute the agent subsystem. The unidirectional arrows between entities represent messages that are sent, and their labels follow the convention AgentFrom 2 AgentTo (for example, a message from the Notification Agent Proxy to the Web Presence Manager will be labelled in the description as NAP2WPM.

The Notification proxy regularly polls the web presence manager for web present entities that are related to each other. The Entity Tracking Agent subscribes to the proxy for information that the proxy has gathered. It is responsible for inferring that if a badge is in the room, it implies the person is also in the room. The Venue Agent subscribes to the entity tracking agent for people presence information. The meeting agent informs the venue agent about the details of a meeting whenever it is scheduled. Both the food service agent, and document agent, serve as domain specific service agents providing recommendations to end users through their personal agents. The Personal agents inform the service agents about personal biography of their bosses.

The sections that follow give complete details about each of the agents in the system, outlining its

functional goal, reasoning capability, default knowledge, communication messages, and a brief explanation of each message and it's content.

Notification Agent Proxy

Functional Goal: The Notification proxy is responsble for:

- Polling the Web Presence Manager for finding out the entities that are related (eg. Badge in a location)
- Informing the Entity Tracking Agent about these related entities

Reasoning Capability: The notification proxy possesses no reasoning capabilities.

Default Knowledge: There is no default knowledge associated with the proxy.

Agent Communication Messages:

Message ID: NAP2ETA

From: Notification Proxy

To: Entity Tracking Agent

Message Trigger: The Message is sent when it figures out that new relations exists between entities

Message Description: This message gives details about new entities that are related with each other giving complete information about the entitites as present in the web presence manager

Protocol: It follows the protocol outlined in Appendix A

Important Message Content:

```
<cto:Is_Related_To rdf:about="urn:is_related_to_001">
       <cto:entity_a rdf:resource="urn:we_001"/>
       <cto:entity_b rdf:resource="urn:we_002"/>
</cto:Is_Related_To>
```

The above message snippet says that there are two web entities that are related to each other.

```
<plc:Is_In rdf:about="urn:is_in_001">
        <tms:start_time rdf:resource="urn:utc_001"/>
        <plc:entity rdf:resource="urn:badge_001"/>
        <plc:indoor_location rdf:resource="urn:cafeteria_001"/>
</plc:Is_In>
```

The above message snippet says that there is a badge entity present in an indoor location and this entity came in at a particular time.

Implementation Features: The Functionality of the proxy is enabled by two modules

- Transformation Routine: This routine performs conversions from XML data originating from the WPM to RDF statements required by the Entity Tracking Agent.
- Comparison Algorithm: This algorithm acts on a single level of cache maintained by the proxy and the data currently fetched. The purpose of this algorithm is to figure out the change in relationships between entities. This is based on two facts, namely, new relationships that are created and old relationships that no longer exist.

Entity Tracking Agent

Functional Goal: Entity Tracking Agent is responsible for

- Subscribing to the proxy for information about entities that are related
- Informing the Venue Agent about people present in the location

Reasoning Capability: It has got the capability to infer that a person is in the location if a badge is in the location.

```
isIn(IsInInst, BadgeInst, LocationInst, UTCInst),
badge(BadgeInst,BadgeID,_), badgeOwner(PersonInst, BdgInst),
badge(BdgInst,BadgeID,_) =>
personIsIn(IsInInst, PersonInst, LocationInst, UTCInst).
```

Default Knowledge: The entity tracking agent knows about who owns badges with particular badge ids.

Agent Communication Messages:

Message ID: NAP2ETA

Reference: Refer to Notification Proxy Agent

Message ID:ETA2VA

From:Entity Tracking Agent

To:Venue Agent

Message Trigger: This message will be sent when the rules shipped by the venue agent are executed and this execution returned some information to be conveyed.

Message Description: This message tells the venue agent that a person is present in the location and his presence was detected at a particular time

Protocol: It follows the protocol outlined in Appendix A

Important Message Content:

```
<plc:Is_In rdf:about="urn:is_in_001">
     <tms:start_time rdf:resource="urn:utc_001"/>
     <plc:entity rdf:resource="urn:person_001"/>
     <plc:indoor_location rdf:resource="urn:cafeteria_001"/>
</plc:Is_In>
```

This message snippet above says that there is a person entity present in an indoor location and this entity came in at a particular time

Venue Agent

Functional Goal: Venue Agent is responsible for

- Subscribing to the Entity Tracking Agent for finding out about people present in the location
- Executing the rules shipped by the Document Agent and based on this execution informing it about people present in a location participating in a meeting
- Executing the rules shipped by the Food Service Agent and based on this execution informing it about people present in the location during a lunch event

Reasoning Capability: The venue agent does not reason about any relationship or facts defined for itself by default. However, it does execute the rules that are shipped to it by the document agent and food service agent based on the information present in its knowledge base.

Default Knowledge: When the venue agent starts, it loads all the meeting information which the meeting agent has generated.

Agent Communication Messages:

Message ID: ETA2VA

Reference: Refer to Refer to Entity Tracking Agent

Message ID: VA2ETA

From: Venue Agent

To: Entity Tracking Agent

Message Trigger: This subscribe message is sent to the entity tracking agent when the venue agent starts

Message Description: This message ships certain rules for execution against the information present in the entity tracking agents inference engine. Along with this message, the venue agent also ships the template to be used for generating the response to be sent to it by the ETA.

Protocol: It follows the protocol outlined in Appendix A

Important Message Content:

```
isIn(IsInInst, PersonInst, LocationInst, UTCInst), utc(UTCInst, Time),
cafeteria(LocationInst, LocName),
person(PersonInst, NameInst), name(NameInst, FName, LName),
represents(RepInst,AgentInst, PersonInst), agent(AgentInst, AgentID)
=>
{add(shouldInform('com.hp.agent.palo-alto.va',
'com.hp.agent.palo-alto.eta',
'va.ruleid.001',
msg(IsInInst, LocationInst, LocName, UTCInst, Time, RepInst,
PersonInst, NameInst, FName, LName, AgentInst, AgentID)))}.
```

The message snippet above outlines the rule which the venue agent wants the entity tracking agent to execute. It implies venue agents intention to get informed when there is a person present in a location and the information should contain all the details of the person and the location in which he is present.

Message ID: VA2DA

From: Venue Agent

To: Document Agent

Message Trigger: When it gets informed by the entity tracking agent about people present in a location, it fires the rules that were shipped to by the document agent. If this execution returned some results, these results are conveyed to the document agent

Message Description: The message tells the document agent that there is a scheduled meeting, giving all the details of the meeting and there are certain participants of the meeting who are already present at the location

Protocol: It follows the protocol outlined in Appendix A

Important Message Content:

```
<mtg:Scheduled_Meeting rdf:about="urn:scheduled_meeting_001"
   mtg:subject="X-CoolTown Presentation" mtg:keyword="cooltown">
<mtg:participant rdf:resource="urn:person_001"/>
<mtg:participant rdf:resource="urn:person_002"/>
<mtg:participant rdf:resource="urn:person_003"/>
<mtg:participant rdf:resource="urn:person_004"/>
<mtg:participant rdf:resource="urn:person_005"/>
<mtg:location rdf:resource="urn:cafeteria_001"/>
<tms:start_time rdf:resource="urn:utc_001"/>
<tms:end_time rdf:resource="urn:utc_002"/>
</mtg:Scheduled_Meeting>
```

The Message snippet above says that there is a scheduled meeting in a location and participants identified by entities(1 to 5) are already present in the location for the meeting

Meeting Agent

The Meeting Agent is a part of the existing CoolAgent project and generating a RDF description of the meeting it schedules is an extension for the CoolAgent RS project.

Functional Goal: The Meeting Agent is responsble for

- Generating an RDF description of the meeting

Reasoning Capability: It does not possess any context-awareness based inferencing capabilities neither does it have its own prolog inference engine

Default Knowledge: It does not possess any default knowledge related to the CoolAgent RS system.

Agent Communication Messages:

Message ID: MA2VA

From: Meeting Agent

To:Venue Agent

Message Trigger: This message is triggered when a meeting is successfully scheduled Message Description: The message tells the venue agent about all the details of a scheduled meeting including participants, start time, endtime , location etc.

Protocol: It follows the protocols outlined in Appendix A

Important Message Content:

```
<d:Cool_Event rdf:about="&f;162"
      d:id="7db00d-e885b2d44c-b05b040f "
      c:subject="Test Meeting">
      <d:invitees rdf:resource="&f;169"/>
      <d:instances rdf:resource="&f;177"/>
</d:Cool_Event>

<d:Cool_Instance rdf:about="&f;177"
 a:duration="3600">
<c:location rdf:resource="&f;179"/>
<c:start rdf:resource="&f;184"/>
<c:attendee-probabilities rdf:resource="&f;190"/>
</d:Cool_Instance>
```

The above message snippet tells the venue agent that there is a coolevent happening at a particular location, having a certain set of invitees,starts at a particular time and the coolevent is associated with a subject and id along with certain other information

Implementation Features: The Meeting Agent does not send a FIPA ACL message informing the venue agent about the meeting. It just generates an RDF description of the meeting and stores it persistently and this description is picked up by the venue agent when it starts.

Document Agent

Functional Goal: Document Agent is responsible for

- Subscribing to the Venue Agent, shipping a bunch of rules to be executed for finding out people present in a location participating in a meeting event
- Sending out Document recommendations to the personal agents of individuals for whom the recommendations are meant

Reasoning Capability: It can recommend documents to people present in a location attending a meeting based on their biography. The recommendation is based on the persons role in the organization, his co-workers, his managers and the project all of them work for. For example

```
hasRole(PersonInst, Role), intern(RoleInst),
coolTownParticipant(PersonInst) =>
recommend(PersonInst, wpmspec),
recommend(PersonInst, whitePaper),
recommend(PersonInst, techReport).

worksWith(P1, P2, OrgInst), recommend(P1, Doc) => recommend(P2, Doc).

manages(P1, P2), recommend(P2, Doc) => recommend(P1, Doc).
```

The above rules say that if a person is an intern, recommend certain documents to him, if a person has a co-worker, recommend all the documents to the co-worker also and if a person has a manager, recommend all the documents to the manager also.

Default Knowledge: It contains information about people profiles as regards, who works for which project and what is the hierarchical relationship between people and what role does each individual have.

Agent Communication Messages:

Message ID: DA2VA

From: Document Agent

To: Venue Agent

Message Trigger: This subscribe message is sent when the document agent starts

Message Description: This message ships certain rules for execution against the venue agents inference engine. Along with these rules, it also ships the template for generating the response sent to it by the venue agent

Protocol: It follows the protocol outlined in Appendix A

Important Message Content:

```
attendingMeeting(PersonInst, SchMeetingInst),
scheduled_meeting(SchMeetingInst, PersonInst, LocationInst,
STInst, ETInst,Keyword, Subject),
cafeteria(LocationInst, LocName), utc(STInst, STime), utc(ETInst, ETime),
person(PersonInst, NameInst), name(NameInst,FName, LName),
represents(RepInst, AgentInst, PersonInst),agent(AgentInst,AgentID)
```

```
 =>
{add(shouldInform('com.hp.agent.palo-alto.da',
'com.hp.agent.palo-alto.va', 'da.ruleid.001',
msg(SchMeetingInst, PersonInst, NameInst, FName, LName,
LocationInst, LocName, STInst, STime, ETInst, ETime,
Keyword, Subject, RepInst, AgentInst, AgentID)))}.
```

Food Service Agent

Functional Goal: The Food Service Agent is responsible for

- Subscribing to the Venue Agent, shipping a bunch of rules to be executed for finding out people present in a location during a lunch event
- Sending out Cuisine recommendations to the personal agents of individuals for whom the recommendations are meant

Reasoning Capability: The food service agent is able to make cuisine recommendations based on the nationality of people who are present in a particular location during a lunch event.

```
inLunch(PersonInst, EventInst), person(PersonInst,NameInst),
name(NameInst,FName,LName),
person(SecPersonInst,SecNameInst), name(SecNameInst,FName,LName),
hasNationality(SecPersonInst,SecNameInst,Nationality)
 => lookingFor(SecPersonInst,Nationality).

lookingFor(PersonInst,Nationality), todaysmenu(Dish,Nationality)
=> recmd1(PersonInst,Dish).
```

The above rule says that if there is a person in a location during a lunch event, then recommend todays special dishes to him based on his nationality.

Default Knowledge: It has default knowledge about the nationality of the people who might be present in a location.

Agent Communication Messages:

Message ID: FSA2VA

From: Food Service Agent

To: Venue Agent

Message Trigger: This subscribe message is sent to the venue agent when the food service agent starts

Message Description: This message ships certain rules for execution against the information present in the venue agents inference engine. Along with this message, the food service agent also ships the template to be used for generating the response to be sent to it by the VA.

Protocol: It follows the protocol outlined in Appendix A

Important Message Content:

```
havingLunch(PersonInst,LunchInst),
scheduled_event(LunchInst,PersonInst,LocationInst,STInst,ETInst) ,
cafeteria(LocationInst,LocName), utc(STInst,STime),
utc(ETInst,ETime), person(PersonInst,NameInst),
name(NameInst,FName,LName), represents(RepInst,AgentInst,PersonInst),
agent(AgentInst,AgentID)
=> {add(shouldInform('com.hp.agent.palo-alto.fsa',
        'com.hp.agent.palo-alto.va',
                'fsa.ruleid.001',
msg(LunchInst, PersonInst, NameInst, FName, LName,
    LocationInst, LocName, STInst, STime, ETInst, ETime,
    RepInst, AgentInst, AgentID)))}.
```

The above message snippet is the rule which the food service agent wants the venue agent to execute. It implies its intention of getting information when there is a person present in a location during a lunch event.

Message ID: FSA2PA

From: Food Service Agent

To: Personal Agent

Message Trigger: This message is triggered when the food service agent gets information from the venue agent about some people present in a location during a lunch event.

Message Description: The message tells the personal agent about certain cuisines which his boss might be interested in.

Protocol: It follows the protocol outlined in Appendix A

Personal Agent

Functional Goal: The Personal Agent is responsible for

- Informing the Document Agent and Food Service Agent about the personal biography of it's boss

Reasoning Capability: It has got no reasoning capability

Default Knowledge: The Default knowledge present with the personal agent is the one which its boss has configured, giving details about him like employee role, nationality etc.

Implementation Features: Right now the personal agent does not send a FIPA ACL Message to the document agent and food service agent. It is assumed that it generates a file giving the personal biography of its boss.

Chapter 5

Demo Scenario

Location: HPL Cafeteria.

People Involved: Reed, Craig, Harry and Sovrin.

Project: CoolAgent.

Food is served int the cafeteria from 3:00 pm to 6:00 pm.

There is a Meeting Scheduled at HPL Cafeteria.

The Role Relationship between people are the following

- Sovrin works on the CoolAgent project.
- Harry is a coworker of Sovrin
- Reed Manages Harry
- Craig is coworker of Harry

The following events happen

- Sovrin enters the cafeteria and cuisine and document recommendations are made to him
- Harry, Reed, and Craig enter the cafeteria and the recommendations are made to them also

The recommendations were made on the following basis:

- Since Sovrin works on the CoolAgent project, he gets a certain set of documents recommended
- Since Harry is the co-worker of Sovrin and sovrin is already present, he also gets the same documents
- Since Craig is the co-worker of Harry, he also gets the same documents
- Since Reed is the manager of Harry, he gets all those documents which harry received, and also high level project related documents.
- All people present were recommended cuisines based on their food preferences.

The Demo Scenario illustrates the following

- The same piece of contextual information is used in two distinct domain reasoning processes and this is made possible by sharing common ontologies so that the meaning of information is also shared and understood.
- Different pieces of contextual information are aggregated together to perform high level reasoning. This aggregation is made possible by the standard triple representation provided by RDF
- The coupling of the ontology and the knowledge representation language leads to the creation of systems with enhanced reasoning capabilities

Chapter 6

Discussion and Future Work

In the course of developing the CoolAgent RS system we have demonstrated the following significant features:

- Construction of Prolog programs that can reason over facts described in RDF
- The interoperatibility that is provided by sharing a common ontology among agents, and the portability of representing ontology and knowledge in RDF. The ontology gives the system consistent semantics of information, and RDF lends portability by representing facts in the standard Triple model.
- A Prolog-based approach using RDF triples, and common ontology, extends the knowledge base of contextual information by aggregating accessible facts that are related.
- The benefits of sharing an ontology among agents

    1. Sharing ontologies allows domain knowledge to be separated from operational knowledge
        - In our system, the information that is involved in the recommendation process is separated from the algorithm that is designed for recommendation
        - Due to the above separation, it is easy to deploy new algorithms to operate on the same information to provide different recommendation results. The concept of rule shipping make this task much easier.

2. Sharing ontologies allows knowledge and agents to be reused for different domain applications
   - The contextual information about people present in a venue location, and their personal information, can be used in distinct domain reasoning processes.
   - The Venue agent can provide recommendations to both document agent and food service agent even though they are two distinct domain applications.

- Effective use of contextual information can potentially lead to the realization of intelligent systems.
- The effective use of agent communication language and ontology can provide a foundation for creating a middleware solution for making heteregeneous sensing infrastructures uniformly accessible by the agents. This proxy approach to gather sensing information:
  1. Shields the low level sensing details (WPM specifics) from the rest of the agent system
  2. Allows existing sensing infrastructures to be replaced, or new infrastructures to be added, without making any implementation changes to any agents other than the proxy agent.

In the course of developing the prototype system we encountered several obstacles:

- Ontology construction is one of the most challenging tasks in the development course of the system. The challenges involved are
  1. Defining the right number of terms in a domain and the relations among them
  2. There is no one correct way or methodology for developing an ontology

  The Challenge is eased in part by
  1. Using ontology construction tools like Protege 2000
  2. Reusing existing ontologies helps accelerate ontology construction
  3. Adopting a modular approach, by decomposing a large ontology into a set of smaller ontologies, can provide better support for reuse and version control

- Manually defining prolog rules to process the contextual information that is described in RDF triples is a tedious process

  1. To create reasoning rules that involve facts expressed in triples, we require the construction of additional intermediate rules. These intermediate rules convert the triple statements, that are a part of the same instance triple model, into declarative prolog terms that are suitable for prolog processing.

  2. An example of this would be:

```
triple(SchMeetingInst, 'rdf_type', 'meeting_Scheduled_Meeting'),
triple(SchMeetingInst, 'meeting_participant', ParticipantInst),
triple(SchMeetingInst, 'meeting_location', LocationInst),
triple(SchMeetingInst, 'times_start_time', UTCInst1),
triple(SchMeetingInst, 'times_end_time', UTCInst2),
triple(SchMeetingInst, 'meeting_keyword', Keyword),
triple(SchMeetingInst, 'meeting_subject', Subject)
=> scheduled_meeting(SchMeetingInst, ParticipantInst,LocationInst,
     UTCInst1, UTCInst2, Keyword, Subject).

triple(NameInst, 'rdf_type', 'people_Name'),
triple(NameInst, 'people_first_name', FName),
triple(NameInst, 'people_last_name', LName) =>
name(NameInst, FName, LName).

triple(PersonInst, 'rdf_type', 'people_Person'),
triple(PersonInst, 'people_name', NameInst)
=> person(PersonInst, NameInst).
```

     To reason about the participants of a meeting, I need intermediate rules such as name and person. As the number of RDF statements increases, defining these intermediate rules manually becomes a tedious process. A utility is warranted which can convert a set of RDF triples into intuitive and usable prolog intermediate rules.

- The basis for comparing two sets of RDF Statements for equality is still not quite clear. While it is evident that their instance types should match, what is not clear is whether instance matches are sufficient or whether key values should be used and, in such a case, what are the key values that should be used for equality — should the key concept be introduced in the RDF model itself, or should it be a part of the Ontology governing the communication?

1. An Example: If I have two instances of a person,

```
<ppl:Person rdf:about="urn:person_001">
<ppl:name rdf:resource="urn:name_001"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_001" ppl:first_name="harry"
                      ppl:last_name="chen"/>


<ppl:Person rdf:about="urn:person_002">
<ppl:name rdf:resource="urn:name_002"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_002" ppl:first_name="harry"
                      ppl:last_name="chen"/>
```

   It is not quite clear (conforming to some standard) if the the basis for establishing equality should be all the keys found in instances, some of the keys found in instances, just the instance itself, or equivalent URI's for all instances representing the same person.

Appendix A

Communication Protocol

This appendix outlines the communication protocols followed between various agents. For each pair of agents, the ACL message format is given along with the message content.

A.1 Notification Proxy and Entity Tracking Agent

```
Entity Tracking Agent sends subscribe message to the Proxy requesting
to be notified when there are new relationships created in the CoolTown
Web Presence Manager.

(subscribe
:sender (agent-identifier :name eta)
:receiver (agent-identifier :name nap)
:content (is_related_to(web_entity_description(X),
    web_entity_description(Y)))
)

The "is_related_to" predicate represents the "is_related_to" relationship class,
which is a part of the "cooltown-ont" ontology.

The Notification Proxy Sends the complete RDF describing the entities that are
related to each other only when new relationships are created.

(inform
:sender (agent-identifier :name nap)
:receiver (agent-identifier :name eta)
:content "[content-msg-0]"
)

[content-msg-0]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY rdfs "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<!ENTITY basic 'http://agora.hpl.hp.com/basic#'>
<!ENTITY cooltown 'http://agora.hpl.hp.com/cooltown#'>
<!ENTITY devices "http://agora.hpl.hp.com/devices#">
<!ENTITY fipa_device "http://agora.hpl.hp.com/fipa_device#">
<!ENTITY place "http://agora.hpl.hp.com/place#">
<!ENTITY people "http://agora.hpl.hp.com/people#">
<!ENTITY agents "http://agora.hpl.hp.com/agents#">
<!ENTITY times "http://agora.hpl.hp.com/times#">
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:bas="&basic;"
xmlns:cto="&cooltown;" xmlns:dev="&devices;" xmlns:fdev="&fipa_device;"
xmlns:plc="&place;" xmlns:ppl="&people;" xmlns:agt="&agents;"
xmlns:tms="&times;">

<dev:RFID_Badge rdf:about="urn:badge_001" dev:badge_id="000565319"
dev:badge_label="Harry Chen"/>
<plc:Cafeteria rdf:about="urn:cafeteria_001" plc:name="HPL Cafeteria"/>
<tms:UTC rdf:about="urn:utc_001" tms:utc_value="2002-04-27T12:30:00Z"/>
<plc:Is_In rdf:about="urn:is_in_001">
<tms:start_time rdf:resource="urn:utc_001"/>
<plc:entity rdf:resource="urn:badge_001"/>
<plc:indoor_location rdf:resource="urn:cafeteria_001"/>
</plc:Is_In>
<cto:Web_Entity rdf:about="urn:we_001">
<cto:we_instance rdf:resource="urn:badge_001"/>
</cto:Web_Entity>
<cto:Web_Entity rdf:about="urn:we_002">
<cto:we_instance rdf:resource="urn:cafeteria_001"/>
</cto:Web_Entity>
<cto:Is_Related_To rdf:about="urn:is_related_to_001">
<cto:entity_a rdf:resource="urn:we_001"/>
<cto:entity_b rdf:resource="urn:we_002"/>
</cto:Is_Related_To>
<cto:Is_Associated_With rdf:about="urn:is_associated_with_001">
<cto:iaw_entity rdf:resource="urn:we_001"/>
<cto:iaw_entity_description rdf:resource="urn:we_description_001"/>
</cto:Is_Associated_With>
<cto:Is_Associated_With rdf:about="urn:is_associated_with_002">
<cto:iaw_entity rdf:resource="urn:we_002"/>
<cto:iaw_entity_description rdf:resource="urn:we_description_002"/>
</cto:Is_Associated_With>
<cto:Web_Entity_Description rdf:about="urn:we_description_001"
cto:we_type="thing" cto:we_name="Harry Chens Badge"
cto:we_description="http://research.ebiquity.org/badge1.rdf">
<cto:we_homepage rdf:resource="urn:uri_string_003"/>
<cto:we_directory rdf:resource="urn:uri_string_002"/>
<cto:we_xml rdf:resource="urn:uri_string_001"/>
</cto:Web_Entity_Description>
<cto:Web_Entity_Description rdf:about="urn:we_description_002"
cto:we_type="place" cto:we_name="HPL 1U Cafeteria"
```

```
cto:we_description="http://agora.hpl.hp.com/hpl1ucafeteria.rdf">
<cto:we_homepage rdf:resource="urn:uri_string_006"/>
<cto:we_directory rdf:resource="urn:uri_string_005"/>
<cto:we_xml rdf:resource="urn:uri_string_004"/>
</cto:Web_Entity_Description>
<bas:URI_String rdf:about="urn:uri_string_001"
bas:uri_string="http://localhost:8080/wpm/main/harryc_badge.description"/>
<bas:URI_String rdf:about="urn:uri_string_002"
bas:uri_string="http://localhost:8080/wpm/main/harryc_badge.directory"/>
<bas:URI_String rdf:about="urn:uri_string_003"
bas:uri_string="http://localhost:8080/wpm/main/harryc_badge.presenter"/>
<bas:URI_String rdf:about="urn:uri_string_004"
bas:uri_string="http://localhost:8080/wpm/main/hpl1ucafeteria.description"/>
<bas:URI_String rdf:about="urn:uri_string_005"
bas:uri_string="http://localhost:8080/wpm/main/hpl1ucafeteria.directory"/>
<bas:URI_String rdf:about="urn:uri_string_006"
bas:uri_string="http://localhost:8080/wpm/main/hpl1ucafeteria.presenter"/>
<ppl:Person rdf:about="urn:person_001">
<ppl:name rdf:resource="urn:name_001"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_001" ppl:first_name="Harry" ppl:last_name="Chen"/>
<bas:Owns rdf:about="urn:owns_001">
<bas:owner rdf:resource="urn:person_001"/>
<bas:possession rdf:resource="urn:badge_001"/>
</bas:Owns>
<agt:Software_Agent rdf:about="urn:software_agent_001" agt:agent_id="agent.id.001"/>
<agt:Represents_Person rdf:about="urn:represents_person_001">
<agt:agent rdf:resource="urn:software_agent_001"/>
<agt:person rdf:resource="urn:person_001"/>
</agt:Represents_Person>
</rdf:RDF>
```

A.2 Entity Tracking Agent and Venue Agent

```
The Venue Agent sends a subscribe request to the ETA and in this process
shipping a bunch of rules to be executed by the ETA. It also ships
the template based on which the response should be constructed.


(subscribe
:sender (agent-identifier :name va)
:receiver (agent-identifier :name eta)
:content "[content-msg-1]"
)

[content-msg-1]

%%
% VA is to be notified when a person's present in the HP Cafeteria has been
% detected.
%%
```

```
:- ensure_loaded('irules.pfc').

%%
% Instance Generation Rules
%%

%
% place:Cafeteria
%
triple(CafeteriaInst, 'rdf_type', 'place_Cafeteria'),
triple(CafeteriaInst, 'place_name', PlaceName) =>
cafeteria(CafeteriaInst, PlaceName).


%
% people:Person
triple(PersonInst, 'rdf_type', 'people_Person'),
triple(PersonInst, 'people_name', NameInst) =>
person(PersonInst, NameInst).

%
% people:Name
%
triple(NameInst, 'rdf_type', 'people_Name'),
triple(NameInst, 'people_first_name', FName),
triple(NameInst, 'people_last_name', LName) =>
name(NameInst, FName, LName).


%
% agents:Represents_Person
%
triple(RepPersonInst, 'rdf_type', 'agents_Represents_Person'),
triple(RepPersonInst, 'agents_agent', AgentInst),
triple(RepPersonInst, 'agents_person', PersonInst)
=> represents(RepPersonInst, AgentInst, PersonInst).

%
% place:Is_In
%
triple(IsInInst, 'rdf_type', 'place_Is_In'),
triple(IsInInst, 'place_entity', EntityInst),
triple(IsInInst, 'place_indoor_location', LocationInst),
triple(IsInInst, 'times_start_time', UTCInst)
=> isIn(IsInInst, EntityInst, LocationInst, UTCInst).
%
% times:UTC
%
triple(UTCInst, 'rdf_type', 'times_UTC'),
triple(UTCInst, 'times_utc_value', UTCTime)
=> utc(UTCInst, UTCTime).

%
% agents:Software_Agent
%
```

```
triple(AgentInst, 'rdf_type', 'agents_Software_Agent'),
triple(AgentInst, 'agents_agent_id', AgentID)
=> agent(AgentInst, AgentID).

%%
% Reasoning Rules.
%%

isIn(IsInInst, PersonInst, LocationInst, UTCInst), utc(UTCInst, Time),
cafeteria(LocationInst, LocName),
person(PersonInst, NameInst), name(NameInst, FName, LName),
represents(RepInst,AgentInst, PersonInst), agent(AgentInst, AgentID)
=>
{add(shouldInform('com.hp.agent.palo-alto.va',
'com.hp.agent.palo-alto.eta',
'va.ruleid.001',
msg(IsInInst, LocationInst, LocName, UTCInst, Time, RepInst,
PersonInst, NameInst, FName, LName, AgentInst, AgentID)))}.


The Entity Tracking Agent sends the following RDF message back to VA
when it detects a person in the room

{inform
:sender (agent-identifier :name eta)
:receiver (agent-identifier :name va)
:content "[content-msg-2]"
)

[content-msg-2]

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<!ENTITY basic "http://agora.hpl.hp.com/basic#">
<!ENTITY place "http://agora.hpl.hp.com/place#">
<!ENTITY people "http://agora.hpl.hp.com/people#">
<!ENTITY agents "http://agora.hpl.hp.com/agents#">
<!ENTITY times "http://agora.hpl.hp.com/times#">
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:bas="&basic;"
xmlns:plc="&place;" xmlns:ppl="&people;" xmlns:agt="&agents;"
xmlns:tms="&times;">

<plc:Cafeteria rdf:about="urn:cafeteria_001" plc:name="HPL Cafeteria"/>
<tms:UTC rdf:about="urn:utc_001" tms:utc_value="2002-04-27T12:30:00Z"/>
<ppl:Person rdf:about="urn:person_003">
<ppl:name rdf:resource="urn:name_003"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_003" ppl:first_name="Reed"
ppl:last_name="Letsinger"/>
<plc:Is_In rdf:about="urn:is_in_003">
<tms:start_time rdf:resource="urn:utc_001"/>
<plc:entity rdf:resource="urn:person_003"/>
```

```
<plc:indoor_location rdf:resource="urn:cafeteria_001"/>
</plc:Is_In>
<agt:Software_Agent rdf:about="urn:software_agent_003"
agt:agent_id="agent.id.003"/>
<agt:Represents_Person rdf:about="urn:represents_person_003">
<agt:agent rdf:resource="urn:software_agent_003"/>
<agt:person rdf:resource="urn:person_003"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_001">
<ppl:name rdf:resource="urn:name_001"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_001" ppl:first_name="Harry"
ppl:last_name="Chen"/>
<plc:Is_In rdf:about="urn:is_in_001">
<tms:start_time rdf:resource="urn:utc_001"/>
<plc:entity rdf:resource="urn:person_001"/>
<plc:indoor_location rdf:resource="urn:cafeteria_001"/>
</plc:Is_In>
<agt:Software_Agent rdf:about="urn:software_agent_001"
agt:agent_id="agent.id.001"/>
<agt:Represents_Person rdf:about="urn:represents_person_001">
<agt:agent rdf:resource="urn:software_agent_001"/>
<agt:person rdf:resource="urn:person_001"/>
</agt:Represents_Person>
</rdf:RDF>
```

A.3 Venue Agent and Document Agent

```
The Document Agent sends a subscribes to VA, asking VA to execute
a bunch of rules so as to get informed when people are present
in a room during a meeting event.It also ships the template
required to generate the response.

(subscribe
:sender (agent-identifier :name da)
:receiver (agent-identifier :name va)
:content "[content-msg-3]"
)

[content-msg-3]

% 1. DA is asking to be notified when there is an occuring meeting evnet.


%%
% Instance Generation Rules
%%
triple(IsInInst, 'rdf_type', 'place_Is_In'),
triple(IsInInst, 'place_entity', EntityInst),
triple(IsInInst, 'place_indoor_location', LocationInst),
triple(IsInInst, 'times_start_time', UTCInst)
```

```
=> isIn(IsInInst, EntityInst, LocationInst, UTCInst).

triple(UTCInst, 'rdf_type', 'times_UTC'),
triple(UTCInst, 'times_utc_value', UTCTime)
=> utc(UTCInst, UTCTime).

%
% people:Name
%
triple(NameInst, 'rdf_type', 'people_Name'),
triple(NameInst, 'people_first_name', FName),
triple(NameInst, 'people_last_name', LName) =>
name(NameInst, FName, LName).
%
% people:Person
%
triple(PersonInst, 'rdf_type', 'people_Person'),
triple(PersonInst, 'people_name', NameInst)
=> person(PersonInst, NameInst).

triple(AgentInst, 'rdf_type', 'agents_Software_Agent'),
triple(AgentInst, 'agents_agent_id', AgentID)
=> agent(AgentInst, AgentID).

triple(RepPersonInst, 'rdf_type', 'agents_Represents_Person'),
triple(RepPersonInst, 'agents_agent', AgentInst),
triple(RepPersonInst, 'agents_person', PersonInst)
=> represents(RepPersonInst, AgentInst, PersonInst).

triple(CafeteriaInst, 'rdf_type', 'place_Cafeteria'),
triple(CafeteriaInst, 'place_name', PlaceName)
=> cafeteria(CafeteriaInst, PlaceName).

triple(SchMeetingInst, 'rdf_type', 'meeting_Scheduled_Meeting'),
triple(SchMeetingInst, 'meeting_participant', ParticipantInst),
triple(SchMeetingInst, 'meeting_location', LocationInst),
triple(SchMeetingInst, 'times_start_time', UTCInst1),
triple(SchMeetingInst, 'times_end_time', UTCInst2),
triple(SchMeetingInst, 'meeting_keyword', Keyword),
triple(SchMeetingInst, 'meeting_subject', Subject)
=> scheduled_meeting(SchMeetingInst, ParticipantInst,LocationInst,
     UTCInst1, UTCInst2, Keyword, Subject).


%%
% These rules map the cool event to Scheduled Event basically from
  coolagent ont to meeting ont
% It is not as clean as ever, timing constraint forces me to do slight adhoc
%%

# Cool Event

triple(CoolEventInst,'rdf_type','coolmeet_Cool_Event'),
triple(CoolEventInst,'coolmeet_invitees',InviteesInst),
triple(CoolEventInst,'coolmeet_instances',EventInst),
```

```
triple(CoolEventInst,'coolmeet_id',ID),
triple(CoolEventInst,'meeting_subject',Subject)
=> coolevent(CoolEventInst,ID,Subject,InviteesInst,EventInst).

# Bag Collection should be done at schema level No time
triple(BagInst,'rdf_type','rdf_Bag'),
triple(BagInst,nothing_,ListItemInst)
=> bag(BagInst,ListItemInst).



# Cool Instance within Cool Event

triple(CoolInst,'rdf_type','coolmeet_Cool_Instance'),
triple(CoolInst,'meeting_location',LocationInst),
triple(CoolInst,'meeting_start',StartTimeInst),
triple(CoolInst,'meeting_attendee-probabilities',ATProbInst),
triple(CoolInst,'times_duration',Duration)
=>
coolinstance(CoolInst,LocationInst,StartTimeInst,ATProbInst,Duration).


triple(CoolPlaceInst,'rdf_type','coolmeet_Cool_Place'),
triple(CoolPlaceInst,'place_name',PlaceName)
=> coolplace(CoolPlaceInst,PlaceName).

coolplace(CoolPlaceInst,PlaceName)
=> cafeteria(CoolPlaceInst,PlaceName).

# Mapping Cool Event to Meeting Event

coolevent(CoolEventInst,ID,Subject,InviteesInst,EventInst),
agent(AGInst,AgentID), bag(InviteesInst,AgentInst),
agent(AgentInst,AgentID), represents(_,AGInst,PersonInst),
coolinstance(EventInst,LocationInst,StartTimeInst,_,_),
coolplace(LocationInst,PlaceName), utc(StartTimeInst,StartTime)
=> scheduled_meeting(CoolEventInst,PersonInst ,LocationInst,
StartTimeInst, StartTimeInst, Subject, Subject).


%%
% Reasoning Rules.
%%
isIn(IsInInst,_,_,UTCInst), utc(UTCInst,UTCTime)
=> cTime(UTCTime).

scheduled_meeting(SchMeetingInst,_,_,StartTimeInst,_,_,_),
utc(StartTimeInst,StartTime)
=> meetingStartTime(SchMeetingInst, StartTime).

cTime(Time),meetingStartTime(SchMeetingInst,Time)
=> occurringMeetingEvent(SchMeetingInst).

%%
% A little dirty hack to create a wrapper term that contains all of
```

```
% the information we need to generate the content of the "inform" message.
%%

occurringMeetingEvent(SchMeetingInst),
scheduled_meeting(SchMeetingInst, ParticipantInst,
LocationInst, _, _,Keyword, Subject),
cafeteria(LocationInst,PLName),
person(ParticipantInst,NInst),name(NInst,FName,LName),
isIn(_,PInst,LInst,_), person(PInst,NSecInst),name(NSecInst,FName,LName),
cafeteria(LInst,PLName)
=> attendingMeeting(ParticipantInst, SchMeetingInst).

attendingMeeting(PersonInst, SchMeetingInst),
scheduled_meeting(SchMeetingInst, PersonInst, LocationInst,
STInst, ETInst,Keyword, Subject),
cafeteria(LocationInst, LocName), utc(STInst, STime), utc(ETInst, ETime),
person(PersonInst, NameInst), name(NameInst,FName, LName),
represents(RepInst, AgentInst, PersonInst),agent(AgentInst,AgentID)
 =>
{add(shouldInform('com.hp.agent.palo-alto.da',
'com.hp.agent.palo-alto.va',
'da.ruleid.001',
msg(SchMeetingInst, PersonInst, NameInst, FName, LName,
LocationInst, LocName, STInst, STime, ETInst, ETime, Keyword,
Subject, RepInst, AgentInst, AgentID)))}.


The Venue agent executes these rules and returns an RDF expression
telling document agent about people in the room during a
meeting event.

(inform
:sender (agent-identifier :name va)
:receiver (agent-identifier :name da)
:content "[content-msg-4]"
)

[content-msg-4]

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY rdfs "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<!ENTITY basic 'http://agora.hpl.hp.com/basic#'>
<!ENTITY place "http://agora.hpl.hp.com/place#">
<!ENTITY people "http://agora.hpl.hp.com/people#">
<!ENTITY meeting "http://agora.hpl.hp.com/meeting#">
<!ENTITY agents "http://agora.hpl.hp.com/agents#">
<!ENTITY times "http://agora.hpl.hp.com/times#">
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:bas="&basic;"
xmlns:plc="&place;" xmlns:ppl="&people;" xmlns:mtg="&meeting;"
xmlns:agt="&agents;" xmlns:tms="&times;">
<mtg:Scheduled_Meeting rdf:about="urn:scheduled_meeting_001"
mtg:subject="X-CoolTown Presentation" mtg:keyword="cooltown">
```

```
<mtg:participant rdf:resource="urn:person_001"/>
<mtg:participant rdf:resource="urn:person_002"/>
<mtg:participant rdf:resource="urn:person_003"/>
<mtg:participant rdf:resource="urn:person_004"/>
<mtg:participant rdf:resource="urn:person_005"/>
<mtg:location rdf:resource="urn:cafeteria_001"/>
<tms:start_time rdf:resource="urn:utc_001"/>
<tms:end_time rdf:resource="urn:utc_002"/>
</mtg:Scheduled_Meeting>
<plc:Cafeteria rdf:about="urn:cafeteria_001"
plc:name="HPL Cafeteria"/>
<ppl:Person rdf:about="urn:person_001">
<ppl:name rdf:resource="urn:name_001"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_001" ppl:first_name="Harry"
ppl:last_name="Chen"/>
<agt:Software_Agent rdf:about="urn:software_agent_001"
agt:agent_id="agent.id.001"/>
<agt:Represents_Person rdf:about="urn:represents_person_001">
<agt:agent rdf:resource="urn:software_agent_001"/>
<agt:person rdf:resource="urn:person_001"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_002">
<ppl:name rdf:resource="urn:name_002"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_002" ppl:first_name="Sovrin"
ppl:last_name="Tolia"/>
<agt:Software_Agent rdf:about="urn:software_agent_002"
agt:agent_id="agent.id.002"/>
<agt:Represents_Person rdf:about="urn:represents_person_002">
<agt:agent rdf:resource="urn:software_agent_002"/>
<agt:person rdf:resource="urn:person_002"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_003">
<ppl:name rdf:resource="urn:name_003"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_003" ppl:first_name="Reed"
ppl:last_name="Letsinger"/>
<agt:Software_Agent rdf:about="urn:software_agent_003"
agt:agent_id="agent.id.003"/>
<agt:Represents_Person rdf:about="urn:represents_person_003">
<agt:agent rdf:resource="urn:software_agent_003"/>
<agt:person rdf:resource="urn:person_003"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_004">
<ppl:name rdf:resource="urn:name_004"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_004" ppl:first_name="Craig"
ppl:last_name="Sayers"/>
<agt:Software_Agent rdf:about="urn:software_agent_004"
agt:agent_id="agent.id.004"/>
<agt:Represents_Person rdf:about="urn:represents_person_004">
<agt:agent rdf:resource="urn:software_agent_004"/>
<agt:person rdf:resource="urn:person_004"/>
</agt:Represents_Person>
```

```
<ppl:Person rdf:about="urn:person_005">
<ppl:name rdf:resource="urn:name_005"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_005" ppl:first_name="Bernard"
ppl:last_name="Burg"/>
<agt:Software_Agent rdf:about="urn:software_agent_005"
agt:agent_id="agent.id.005"/>
<agt:Represents_Person rdf:about="urn:represents_person_005">
<agt:agent rdf:resource="urn:software_agent_005"/>
<agt:person rdf:resource="urn:person_005"/>
</agt:Represents_Person>
<tms:UTC rdf:about="urn:utc_001"
tms:utc_value="2002-04-27T12:30:00Z"/>
<tms:UTC rdf:about="urn:utc_002"
tms:utc_value="2002-04-27T13:30:00Z"/>
<tms:Occurring rdf:about="urn:occuring_001">
<tms:event rdf:resource="urn:scheduled_meeting_001"/>
</tms:Occurring>
</rdf:RDF>
```

A.4 Venue Agent and Food Service Agent

```
The Food service agent subscribes to VA, shipping a bunch of rules
which it wants VA to execute and inform when there are people
in the room during lunch event. The template is also sent
along with the subscription process.

(subscribe
:sender (agent-identifier :name fsa)
:receiver (agent-identifier :name va)
:content "[content-msg-5]"
)

[content-msg-5]

:- dynamic lunchStartTime/2.
%%
% Instance Generation Rules
%%
triple(IsInInst, 'rdf_type', 'place_Is_In'),
triple(IsInInst, 'place_entity', EntityInst),
triple(IsInInst, 'place_indoor_location', LocationInst),
triple(IsInInst, 'times_start_time', UTCInst)
=> isIn(IsInInst, EntityInst, LocationInst, UTCInst).

triple(UTCInst, 'rdf_type', 'times_UTC'),
triple(UTCInst, 'times_utc_value', UTCTime)
=> utc(UTCInst, UTCTime).
```

```
%
% people:Name
%
triple(NameInst, 'rdf_type', 'people_Name'),
triple(NameInst, 'people_first_name', FName),
triple(NameInst, 'people_last_name', LName) =>
name(NameInst, FName, LName).


%
% people:Person
%
triple(PersonInst, 'rdf_type', 'people_Person'),
triple(PersonInst, 'people_name', NameInst)
=> person(PersonInst, NameInst).

triple(AgentInst, 'rdf_type', 'agents_Software_Agent'),
triple(AgentInst, 'agents_agent_id', AgentID)
=> agent(AgentInst, AgentID).

triple(RepPersonInst, 'rdf_type', 'agents_Represents_Person'),
triple(RepPersonInst, 'agents_agent', AgentInst),
triple(RepPersonInst, 'agents_person', PersonInst)
=> represents(RepPersonInst, AgentInst, PersonInst).

triple(CafeteriaInst, 'rdf_type', 'place_Cafeteria'),
triple(CafeteriaInst, 'place_name', PlaceName)
=> cafeteria(CafeteriaInst, PlaceName).

%%
% FSA defining the Lunch Event
%%

=> triple('urn:lunch_event_001', 'rdf_type', 'dining_Lunch_Event').
=> triple('urn:lunch_event_001', 'times_start_time', 'urn:utc_001').
=> triple('urn:lunch_event_001', 'times_end_time', 'urn:utc_002').
=> triple('urn:utc_001', 'rdf_type', 'times_UTC').
=> triple('urn:utc_001', 'times_utc_value', '2002-04-27T12:30:00Z').
=> triple('urn:utc_002', 'rdf_type', 'times_UTC').
=> triple('urn:utc_002', 'times_utc_value', '2002-04-27T13:30:00Z').
=> lunchStartTime('urn:lunch_event_001',2002-04-27T12:30:00Z).

triple(LunchInst, 'rdf_type', 'dining_Lunch_Event'),
triple(LunchInst, 'dining_person', PersonInst),
triple(LunchInst, 'meeting_location', LocationInst),
triple(LunchInst, 'times_start_time', UTCInst1),
triple(LunchInst, 'times_end_time', UTCInst2)
=> scheduled_event(LunchInst,PersonInst,LocationInst,
UTCInst1,UTCInst2).


%%
% Reasoning Rules.
%%
isIn(IsInInst,_,_,UTCInst), utc(UTCInst,UTCTime)
```

```
=> cTime(UTCTime).


cTime(Time),lunchStartTime(LunchInst,Time)
=> occurringLunchEvent(LunchInst,Time).

occurringLunchEvent(LunchInst,Time), utc(UTCInst,Time),
isIn(_,PersonInst,LocationInst,SecUTCInst),utc(SecUTCInst,Time) ,
person(PersonInst,_)
=> triple(LunchInst, 'dining_person', PersonInst),
   triple(LunchInst,'meeting_location', LocationInst).


occurringLunchEvent(LunchInst,_),
scheduled_event(LunchInst,PersonInst,LocationInst,_,_),
isIn(_,PersonInst,LocationInst,_)
=> havingLunch(PersonInst,LunchInst).


havingLunch(PersonInst,LunchInst),
scheduled_event(LunchInst,PersonInst,LocationInst,STInst,ETInst) ,
cafeteria(LocationInst,LocName), utc(STInst,STime),
utc(ETInst,ETime), person(PersonInst,NameInst),
name(NameInst,FName,LName), represents(RepInst,AgentInst,PersonInst),
agent(AgentInst,AgentID)
=> {add(shouldInform('com.hp.agent.palo-alto.fsa',
        'com.hp.agent.palo-alto.va',
                'fsa.ruleid.001',
msg(LunchInst, PersonInst, NameInst, FName, LName,
    LocationInst, LocName, STInst, STime, ETInst, ETime,
    RepInst, AgentInst, AgentID)))}.


The Venue Agent executes the above rules and informs food service agent
about people present in a location during a lunch event.

(inform
:sender (agent-identifier :name va)
:receiver (agent-identifier :name fsa)
:content "[content-msg-6]"
)

[content-msg-6]

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY rdfs "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<!ENTITY basic 'http://agora.hpl.hp.com/basic#'>
<!ENTITY place "http://agora.hpl.hp.com/place#">
<!ENTITY people "http://agora.hpl.hp.com/people#">
<!ENTITY agents "http://agora.hpl.hp.com/agents#">
<!ENTITY times "http://agora.hpl.hp.com/times#">
<!ENTITY dining "http://agora.hpl.hp.com/dining#">
<!ENTITY meeting "http://agora.hpl.hp.com/meeting#">
```

```
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:bas="&basic;"
xmlns:plc="&place;" xmlns:ppl="&people;" xmlns:agt="&agents;"
xmlns:tms="&times;" xmlns:din="&dining;" xmlns:mtg="&meeting;">
<din:Lunch_Event rdf:about="urn:lunch_event_001">
<din:person rdf:resource="urn:person_001"/>
<din:person rdf:resource="urn:person_002"/>
<din:person rdf:resource="urn:person_003"/>
<din:person rdf:resource="urn:person_004"/>
<din:person rdf:resource="urn:person_005"/>
<mtg:location rdf:resource="urn:cafeteria_001"/>
<tms:start_time rdf:resource="urn:utc_001"/>
<tms:end_time rdf:resource="urn:utc_002"/>
</din:Lunch_Event>
<plc:Cafeteria rdf:about="urn:cafeteria_001"
plc:name="HPL Cafeteria"/>
<ppl:Name rdf:about="urn:name_001" ppl:first_name="Harry"
ppl:last_name="Chen"/>
<agt:Software_Agent rdf:about="urn:software_agent_001"
agt:agent_id="agent.id.001"/>
<agt:Represents_Person rdf:about="urn:represents_person_001">
<agt:agent rdf:resource="urn:software_agent_001"/>
<agt:person rdf:resource="urn:person_001"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_002">
<ppl:name rdf:resource="urn:name_002"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_002" ppl:first_name="Sovrin"
ppl:last_name="Tolia"/>
<agt:Software_Agent rdf:about="urn:software_agent_002"
agt:agent_id="agent.id.002"/>
<agt:Represents_Person rdf:about="urn:represents_person_002">
<agt:agent rdf:resource="urn:software_agent_002"/>
<agt:person rdf:resource="urn:person_002"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_003">
<ppl:name rdf:resource="urn:name_003"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_003" ppl:first_name="Reed"
ppl:last_name="Letsinger"/>
<agt:Software_Agent rdf:about="urn:software_agent_003"
agt:agent_id="agent.id.003"/>
<agt:Represents_Person rdf:about="urn:represents_person_003">
<agt:agent rdf:resource="urn:software_agent_003"/>
<agt:person rdf:resource="urn:person_003"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_004">
<ppl:name rdf:resource="urn:name_004"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_004" ppl:first_name="Craig"
ppl:last_name="Sayers"/>
<agt:Software_Agent rdf:about="urn:software_agent_004"
agt:agent_id="agent.id.004"/>
<agt:Represents_Person rdf:about="urn:represents_person_004">
<agt:agent rdf:resource="urn:software_agent_004"/>
```

```
<agt:person rdf:resource="urn:person_004"/>
</agt:Represents_Person>
<ppl:Person rdf:about="urn:person_005">
<ppl:name rdf:resource="urn:name_005"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_005" ppl:first_name="Bernard"
ppl:last_name="Burg"/>
<agt:Software_Agent rdf:about="urn:software_agent_005"
agt:agent_id="agent.id.005"/>
<agt:Represents_Person rdf:about="urn:represents_person_005">
<agt:agent rdf:resource="urn:software_agent_005"/>
<agt:person rdf:resource="urn:person_005"/>
</agt:Represents_Person>
<tms:UTC rdf:about="urn:utc_001"
tms:utc_value="2002-04-27T12:30:00Z"/>
<tms:UTC rdf:about="urn:utc_002"
tms:utc_value="2002-04-27T13:30:00Z"/>
<tms:Occurring rdf:about="urn:occuring_001">
<tms:event rdf:resource="urn:scheduled_event_001"/>
</tms:Occurring>
</rdf:RDF>
```

A.5 Miscellaneous

The Meeting Agent publishes the following Meeting Descriptions

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
 <!ENTITY a 'http://agora.hpl.hp.com/times#'>
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY b 'http://agora.hpl.hp.com/agents#'>
 <!ENTITY c 'http://agora.hpl.hp.com/meeting#'>
 <!ENTITY d 'http://agora.hpl.hp.com/coolagent_meeting#'>
 <!ENTITY e 'http://agora.hpl.hp.com/place#'>
 <!ENTITY f 'urn:rdf:23d5b83cf370769e0a8464bfb807737a-'>
]>
<rdf:RDF xmlns:a="&a;"
 xmlns:rdf="&rdf;"
 xmlns:d="&d;"
 xmlns:c="&c;"
 xmlns:b="&b;"
 xmlns:e="&e;"
 xmlns:f="&f;">
<d:Cool_Event rdf:about="&f;32"
 d:id="7db00d-e885aa2287-b05b040f "
 c:subject="cooltown">
<d:invitees rdf:resource="&f;39"/>
<d:instances rdf:resource="&f;47"/>
</d:Cool_Event>
<rdf:Bag rdf:about="&f;39">
<rdf:li rdf:resource="&f;42"/>
```

```
<rdf:li rdf:resource="&f;43"/>
<rdf:li rdf:resource="&f;44"/>
<rdf:li rdf:resource="&f;45"/>
<rdf:li rdf:resource="&f;46"/>
</rdf:Bag>
<b:Software_Agent rdf:about="&f;42"
 b:agent_id="agentid002"/>
<b:Software_Agent rdf:about="&f;43"
 b:agent_id="agentid001"/>
<b:Software_Agent rdf:about="&f;44"
 b:agent_id="agentid003"/>
<b:Software_Agent rdf:about="&f;45"
 b:agent_id="agentid004"/>
<b:Software_Agent rdf:about="&f;46"
 b:agent_id="agentid005"/>
<d:Cool_Instance rdf:about="&f;47"
 a:duration="3600">
<c:location rdf:resource="&f;49"/>
<c:start rdf:resource="&f;54"/>
<c:attendee-probabilities rdf:resource="&f;60"/>
</d:Cool_Instance>
<d:Cool_Place rdf:about="&f;49"
 e:name="hplcafeteria"/>
<a:UTC rdf:about="&f;54"
 a:utc_value="2001-08-27T00:00:00Z"/>
<rdf:Bag rdf:about="&f;60">
<rdf:li rdf:resource="&f;62"/>
<rdf:li rdf:resource="&f;81"/>
<rdf:li rdf:resource="&f;82"/>
<rdf:li rdf:resource="&f;83"/>
<rdf:li rdf:resource="&f;84"/>
</rdf:Bag>
<d:Cool_Agent_Float_Pair rdf:about="&f;62"
 e:value="1.0">
<e:agent rdf:resource="&f;64"/>
</d:Cool_Agent_Float_Pair>
<b:Software_Agent rdf:about="&f;64"
 b:agent_id="agentid002"/>
<d:Cool_Agent_Float_Pair rdf:about="&f;81"
 e:value="1.0">
<e:agent rdf:resource="&f;91"/>
</d:Cool_Agent_Float_Pair>
<b:Software_Agent rdf:about="&f;91"
 b:agent_id="agentid001"/>
<d:Cool_Agent_Float_Pair rdf:about="&f;82"
 e:value="1.0">
<e:agent rdf:resource="&f;92"/>
</d:Cool_Agent_Float_Pair>
<b:Software_Agent rdf:about="&f;92"
 b:agent_id="agentid002"/>
<d:Cool_Agent_Float_Pair rdf:about="&f;83"
 e:value="1.0">
<e:agent rdf:resource="&f;93"/>
</d:Cool_Agent_Float_Pair>
<b:Software_Agent rdf:about="&f;93"
```

```
 b:agent_id="agentid002"/>
<d:Cool_Agent_Float_Pair rdf:about="&f;84"
 e:value="1.0">
<e:agent rdf:resource="&f;94"/>
</d:Cool_Agent_Float_Pair>
<b:Software_Agent rdf:about="&f;94"
 b:agent_id="agentid002"/>
</rdf:RDF>
```

The Personal Agent Publishes the following profiles

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY rdfs "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<!ENTITY basic 'http://agora.hpl.hp.com/basic#'>
<!ENTITY organization "http://agora.hpl.hp.com/organization#">
<!ENTITY people "http://agora.hpl.hp.com/people#">
<!ENTITY meeting "http://agora.hpl.hp.com/meeting#">
<!ENTITY agents "http://agora.hpl.hp.com/agents#">
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:bas="&basic;"
xmlns:org="&organization;" xmlns:ppl="&people;" xmlns:mtg="&meeting;"
xmlns:agt="&agents;">
<ppl:Person rdf:about="urn:person_001" ppl:nationality="hongkong">
<ppl:name rdf:resource="urn:name_001"/>
</ppl:Person>
<ppl:Contact_Info rdf:about="urn:contact_info_001"
ppl:email="harryc@exch.hpl.hp.com">
<ppl:contact_person rdf:resource="urn:person_001"/>
</ppl:Contact_Info>
<ppl:Name rdf:about="urn:name_001" ppl:first_name="harry"
ppl:last_name="chen"/>
<org:Intern rdf:about="urn:intern_001"/>
<org:Has_Employment_Role rdf:about="urn:has_employment_role_001">
<org:employee_role rdf:resource="urn:intern_001"/>
<org:person rdf:resource="urn:person_001"/>
</org:Has_Employment_Role>
<ppl:Person rdf:about="urn:person_002" ppl:nationality="india">
<ppl:name rdf:resource="urn:name_002"/>
</ppl:Person>
<ppl:Contact_Info rdf:about="urn:contact_info_002"
ppl:email="sovrin@exch.hpl.hp.com">
<ppl:contact_person rdf:resource="urn:person_002"/>
</ppl:Contact_Info>
<ppl:Name rdf:about="urn:name_002" ppl:first_name="sovrin"
ppl:last_name="tolia"/>
<org:Has_Employment_Role rdf:about="urn:has_employment_role_002">
<org:employee_role rdf:resource="urn:intern_001"/>
<org:person rdf:resource="urn:person_002"/>
</org:Has_Employment_Role>
<ppl:Person rdf:about="urn:person_003" ppl:nationality="america">
<ppl:name rdf:resource="urn:name_003"/>
```

```
</ppl:Person>
<ppl:Contact_Info rdf:about="urn:contact_info_003"
ppl:email="letsinge@exch.hpl.hp.com">
<ppl:contact_person rdf:resource="urn:person_003"/>
</ppl:Contact_Info>
<ppl:Name rdf:about="urn:name_003" ppl:first_name="reed"
ppl:last_name="letsinger"/>
<org:Project_Manager rdf:about="urn:project_manager_001"/>
<org:Has_Employment_Role rdf:about="urn:has_employment_role_003">
<org:employee_role rdf:resource="urn:project_manager_001"/>
<org:person rdf:resource="urn:person_003"/>
</org:Has_Employment_Role>
<org:Has_Employment_Role rdf:about="urn:has_employment_role_003_001">
<org:employee_role rdf:resource="urn:research_scientist_001"/>
<org:person rdf:resource="urn:person_003"/>
</org:Has_Employment_Role>
<ppl:Person rdf:about="urn:person_004" ppl:nationality="newzealand">
<ppl:name rdf:resource="urn:name_004"/>
</ppl:Person>
<ppl:Name rdf:about="urn:name_004" ppl:first_name="craig"
ppl:last_name="sayers"/>
<ppl:Contact_Info rdf:about="urn:contact_info_004"
ppl:email="csayers@exch.hpl.hp.com">
<ppl:contact_person rdf:resource="urn:person_004"/>
</ppl:Contact_Info>
<org:Has_Employment_Role rdf:about="urn:has_employment_role_004">
<org:employee_role rdf:resource="urn:research_scientist_001"/>
<org:person rdf:resource="urn:person_004"/>
</org:Has_Employment_Role>
<org:Research_Scientist rdf:about="urn:research_scientist_001"/>
<ppl:Person rdf:about="urn:person_005" ppl:nationality="france">
<ppl:name rdf:resource="urn:name_005"/>
</ppl:Person>
<ppl:Contact_Info rdf:about="urn:contact_info_005"
ppl:email="bernardb@exch.hpl.hp.com">
<ppl:contact_person rdf:resource="urn:person_005"/>
</ppl:Contact_Info>
<ppl:Name rdf:about="urn:name_005" ppl:first_name="bernard"
ppl:last_name="burg"/>
<org:Department_Manager rdf:about="urn:department_manager_001"/>
<org:Has_Employment_Role rdf:about="urn:has_employment_role_005">
<org:employee_role rdf:resource="urn:department_manager_001"/>
<org:person rdf:resource="urn:person_005"/>
</org:Has_Employment_Role>
<org:Works_With rdf:about="urn:works_with_001">
<!--sorvin says, "i work  with harry"-->
<org:organization rdf:resource="urn:company_001"/>
<org:person_a rdf:resource="urn:person_002"/>
<org:person_b rdf:resource="urn:person_001"/>
</org:Works_With>
<org:Works_With rdf:about="urn:works_with_002">
<!--craig says, "i work with harry"-->
<org:organization rdf:resource="urn:company_001"/>
<org:person_a rdf:resource="urn:person_004"/>
<org:person_b rdf:resource="urn:person_001"/>
```

```
</org:Works_With>
<org:Manages rdf:about="urn:manages_001">
<org:manager rdf:resource="urn:person_003"/>
<org:agent rdf:resource="urn:person_001"/>
</org:Manages>
<org:Project rdf:about="urn:project_001"
org:project_name="coolagent" org:keywords="cooltown"/>
<org:Participates rdf:about="urn:participates_002">
<org:person rdf:resource="urn:person_002"/>
<org:project rdf:resource="urn:project_001"/>
</org:Participates>
</rdf:RDF>

The Rules for RDF/RDFS logical Intepretation are

:- dynamic instanceOf/2.

triple(S,P,O) => res(S), uri(P), obj(O).

uri(R) => res(R).

obj(O) , ~res(O) => lit(O).

triple(A,
'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#subClassOf',B)
=> subClassOf(A,B).

triple(A,
'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#subClassOf',B),
subClassOf(B,C) => subClassOf(A,C).

triple(A,
'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#subPropertyOf',B)
=> subPropertyOf(A,B).

triple(A,
'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#subPropertyOf',B),
subPropertyOf(B,C) => subPropertyOf(A,C).

triple(I,
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',C)
=> instanceOf(I,C).

subClassOf(B,C), instanceOf(I,B) => instanceOf(I,C).
```

Bibliography

[1] Gregory Abowd A. Dey, D. The conference assistant: Combining context -awareness with wearable computing. *3rd International Symposium on Wearable Computers*, pages 21–28, October 1999.

[2] Gregory D. Abowd, Jason A. Brotherton, and Janak Bhalodia. Classroom 2000: A system for capturing and accessing multimedia classroom experiences. *CHI*, May 1998.

[3] Varol Akman and Mehmet Surav. Steps toward formalizing context. *AI Magazine*, 17(3):55–72, 1996.

[4] P. J. Brown, N. Davies, M. Smith, and P. Steggles. Panel: towards a better understanding of context and context-awareness. In *Hand-held and ubiquitous computing: HUC'99 proceedings*, 1999.

[5] HID Corporation. Access control cards and readers. Available online from `http://www.hidcorp.com`.

[6] Philippe Debaty. Web presence manager api documentation, 2001. Available online from `http://debaty2.hpl.hp.com/wpmdocs/javadoc/`.

[7] Philippe Debaty. Web presence manager documentation, 2001. Available online from `http://debaty2.hpl.hp.com/wpmdocs/wpm_user_guide.html`.

[8] CSELT Centro Studi e Laboratori Telecomunicazioni S.p.A. Java agent development environment. More Information from `http://sharon.cselt.it/projects/jade/`.

[9] Tim Finin. *P_{fc}User Manual*. Dept. of CSEE, U. of Maryland Baltimore County, August 1999.

[10] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification.* W3C, February 1999.

[11] Sun Microsystem. Java communications api. Available online from `http://java.sun.com/products/javacomm/index.html`.

[12] N. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Medical Informatics, 2001.

[13] Debaty Philippe and Deborah Caswell. Uniform web presence architecture for people, places and things. *IEEE Personal Communications*, 2001. Available Online from `http://www.hpl.hp.com/techreports/2000/HPL-2000-67.html`.

$Revision: 1.2 $, $Date: 2001/09/25 20:58:00 $