# Creating Context-Aware Software Agents

Harry Chen[1], Sovrin Tolia[1], Craig Sayers[2], Tim Finin[1], and Anupam Joshi[1]

[1] eBiquity Research Group
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250, USA,
{hchen4,stolia1,finin,joshi}@cs.umbc.edu
[2] Software Technology Laboratory
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94340, USA,
craig_sayers@hpl.hp.com

**Abstract.** Sharing ontology, sensing context and reasoning are crucial to the realization of context-aware software agents. This document describes our effort on using Resource Description Framework (RDF) and the Prolog Forward Chaining ($P_{fc}$) system to provide support for ontology sharing and reasoning in the CoolAgent Recommendation System (CoolAgent RS), a context-aware multi-agent system. This document also describes the implementation of the CoolAgent RS document and cuisine recommendation services that provide tailored services by exploiting user's context.

## 1 Introduction

We humans are context-aware. We are able to use implicit situational information, or context, to increase our conversational bandwidth [?]. This ability allows us to act in advance and anticipate other's needs.

For example, when two people are in the same room, Person A asks Person B, "close the door please." Naturally, Person B would reason that Person A is requesting the door in the same room to be closed, not the door in any other room.

This simple example demonstrates context-awareness in human beings. The fact that Person B is able to take the right action is due to the following three valuable capabilities of humans:

1. Ontology sharing – humans are able to share communication languages and vocabularies
2. Sensing – humans are able to perceive their environment through sensory organs
3. Reasoning – humans are able to make sense out of what they have perceived based on what they already know

If Person B is unable to share ontology with Person A or unable to sense from Person A or unable to make sense out of what he/she has perceived, then

Person B would not be able to close the door that Person A desires. In other words, Person B becomes context-aware only when he/she possesses all of the three capabilities described above.

We believe ontology sharing, sensing and reasoning are not only crucial to human context-awareness, but also significant to the realization of context-aware applications.

### 1.1 Context-aware applications

The construction of context-aware applications is cumbersome and challenging [?]. Exploiting context in applications requires sensing infrastructures to capture contextual information in the physical environment, and reasoning infrastructures to support inferencing in distributed applications.

After reviewing existing context-aware systems [?, ?, ?], we find that much of the current effort has been focused on the creation of reusable infrastructures for sensing contextual information. Futhermore, from a survey of context-aware research [?], we learn that context modeling and reasoning have not been seriously considered in the existing context-aware systems.

The Mobisaic Web Browser [?] is one of the early context-aware application that exploits location-related, and time-related, contextual information. Mobisaic extends standard client browsers to allow authors to reference dynamic contextual information in dynamic URLs containing environment variables. The dynamic URL is interpreted using current values of the environment variables, and an appropriate page is returned. Mobisaic neither includes any context modeling framework nor does it provide a domain-independent sensing infrastructure.

The Conference Assistant [?] assists attendees by exploiting context from their location, the current time, and the schedules of presentations. After a mobile user enters a conference room, it automatically displays the name of the presenter, the title of the presentation and related information.

Available audio and video equipment automatically record the current presentation, comments, and questions for later retrieval. The core of the Conference Assistant is the Context Toolkit [?], which has limited expressive power to support context modelling and reasoning.

Other systems [?, ?] that we have reviewed also possess similar weaknesses in providing support for ontology sharing and reasoning. In particular, generic contextual information is often directly programmed as part each domain-specific implementation. Such design forces the overall systems to be tightly coupled, and the system behavior becomes highly dependent on the structural representation of the contextual information.

In this document, we describe our efforts on using RDF and $P_{fc}$ to provide support for sharing ontology and reasoning in CoolAgent RS, a context-aware multi-agent system. We also describe the implementation of the CoolAgent RS document and cuisine recommendation services that provide tailored services by exploiting user's context.

Section 2 discusses the use of contextual information in the CoolAgent RS recommendation services. Section 3 provides a design overview of the CoolAgent

RS and reviews some of the related work. Section 4 describes the implementation details of the CoolAgent ontology, reasoning and agents. Conclusion and acknowledgement are given in Sect. 5 and Sect. 6, respectively.

## 2 Context-Aware Services in CoolAgent RS

Our context-aware multi-agent system is designed around the three important factors that contribute to human context-awareness: ontology sharing, sensing, and reasoning. In this section, we describe two context-aware recommendation services that exploit those three factors.

### 2.1 CoolAgent RS Document Recommendation Service

Similar to many of the existing document recommendation services, this recommendation service is designed to recommend relevant documents to meeting participants. Nevertheless, unlike most of the existing services that often make recommendations based on the relevance rankings between the document contents and user requests, the CoolAgent RS Document Recommendation Service makes recommendations based on the dynamic contextual information of the users.

The recommendation service exploits the following contextual information:

- *the presence of a meeting participant*: a person is a meeting participant if and only if that person is present in a location during the time when that location is anticipated to have a scheduled meeting
- *the profile of a meeting participant*: this includes the personal and professional background information of the participant (e.g. what is job title of the participant? what are the research interests of the participant? what is the placement of the participant in the company's organization chart?)
- *the planned meeting context*: this includes the subject of the meeting, research work and projects that are related to the meeting, the schedule of the meeting and the anticipated participants etc.
- *organizational information*: this includes the employment relationships among the participants (e.g. is person A the supervior of person B? is person A working with person B on the same project?) and the company internal policies (e.g. what kind of documents would a department manager be interested in?)

It is easy to see how some of the described contextual information can be used in a document recommendation process. For example, the presence of a participant can be used to pinpoint the recommenation candidates, and the research interests of the participant can be used to limit the search space of finding relevant documents.

On the other hand, some others (e.g. the schedule of a meeting) cannot be used independently to support recommendations. Nevertheless, this information, as a piece of integrated knowledge, can be used to deduce new contextual information to allow more sophisticated reasoning that would otherwise not be possible. For example, by knowning Person A is supervising Person B, and Person B is working on project CoolTown, then it is appropriate to recommend CoolTown documents to Person A.

### 2.2 CoolAgent RS Food Recommendation Service

Similar to the CoolAgent RS Document Recommendation Service, this recommendation service recommends customized lunch specials to the people in a cafeteria during lunch hours. This service exploits the following contextual information:

– *the presence of a diner*: a person is a diner if and only if that person is in a dining venue during the lunch hours
– *the profile of a diner*: similar to the profile information that is used in the document recommendation service
– *the daily lunch special menu*: the description of the lunch specials, including the ingredients and cuisine information

### 2.3 The Need to Share Ontology

One interesting observation from the services described above is that the same contextual information is exploited by two different services in very distinct application domains. For example, the presence of people, the profile of people etc..

This observation leads us to conclude that ontology sharing is important to our context-aware system. If services can share a common representation and context semantics, then overlapping information can be easily shared and exchanged, thereby increasing the interoperability and flexiblity of the overall system.

## 3 Designing CoolAgent RS

The CoolAgent RS was started in the summer of 2001 as a Summer Intern project at Hewlett-Packard Laboratories. The goal was to develop a context-aware extension to a prototype multi-agent Meeting Management system termed CoolAgent[**?**].

CoolAgent RS is a multi-agent system that can automatically recommend different types of tailored information to users by reasoning from their context without any explicit manual input. The goal is to develop a proof-of-concept agent system that demonstrates the significance of logical reasoning and ontology sharing in a context-aware distributed computing environment.

### 3.1 Design Goals

To support the context-aware services that we have described in Sect. 2, the design goals of our system are the following:

– separate the representation and interpretation of the contextual information from the system's operating implementation
– provide a knowledge representation infrastructure to allow contextual information to be represented, shared and manipulated
– provide a reasoning mechanism that supports cooperative reasoning

## 4 Implementing CoolAgent RS

The core of CoolAgent RS is a collection of agents, all of which are FIPA-compliant JADE agents [?]. The following is an overview of the agents in CoolAgent RS (also see Figure 1):

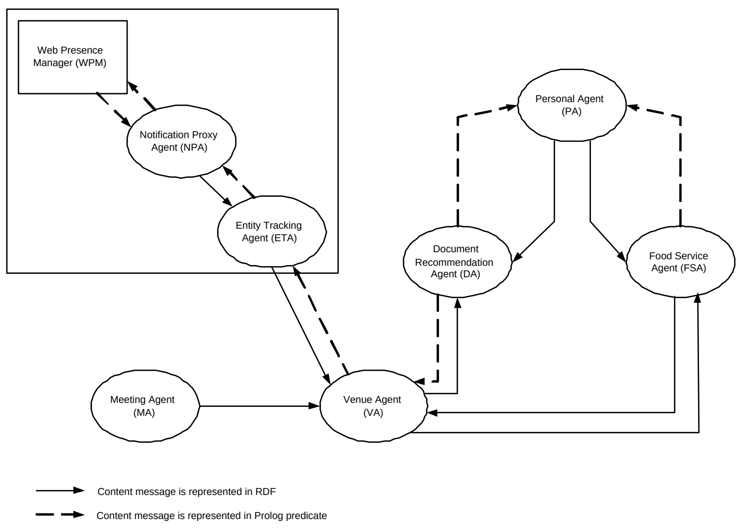
Web Presence Manager (WPM)
Notification Proxy Agent (NPA)
Entity Tracking Agent (ETA)
Personal Agent (PA)
Document Recommendation Agent (DA)
Food Service Agent (FSA)
Meeting Agent (MA)
Venue Agent (VA)
Content message is represented in RDF
Content message is represented in Prolog predicate

**Fig. 1.** An overview of the CoolAgent RS architecture

**Notification Agent Proxy (NAP)** This agent is responsible for polling the CoolTown Web Presence Manager (WPM) [?] to determine the presences of physical objects in a particular location. NAP acts a proxy to WPM for agents that do not have direct access to WPM. In particular, NAP provides an ontology mapping service, translating contextual information from the CoolTown WPM ontology to the CoolAgent RS ontology, for agents subscribing to it.

**Entity Tracking Agent (ETA)** This agent is responsible for tracking entity presences in a particular location. ETA subscribes to NAP. ETA requests to be notified when any physical objects are present in a particular location.

**Venue Agent (VA)** This agent is responsible for mediating contextual information among the agents. VA collects contextual information of a particular location, including people presences, anticipated meeting information and relationship among various types of entities, by receiving entity presence notification from ETA and communicating with the Meeting Agent.

**Personal Agent (PA)** This agent is responsible for providing personal and professional profiles of human users. PA shares common ontology with other agents, and it does not have any built-in reasoning capability.

**Meeting Agent (MA)** This agent is part of the existing CoolAgent Meeting Management prototype. It creates an RDF-encoded web page to describe each new meeting. The VA extracts details for upcoming meetings by examining those published pages.

**Document Recommendation Agent (DA)** This agent is responsible for providing document recommendations to the meeting participants based on their context.

**Food Service Agent (FA)** This agent is responsible for providing cuisine recommendations to the diners in a cafeteria based on their context.

In the following subsections we describe the ontology sharing and the logical reasoning aspects of the CoolAgent RS implementation.

### 4.1 CoolAgent RS Ontology

The CoolAgent RS ontology consists of 231 classes and 179 properties. Classes describe the concepts in the CoolAgent RS application domains. Properties describe various features and attributes of the classes.

The CoolAgent RS ontology is constructed using Protege-2000, a frame-based ontology authoring tool [?]. Using Protege-2000, classes are modulated into 12 RDF Schema files constituting different namespaces.

These RDF Schema files contains ontologies that capture a wide range of domain concepts that describe software agents, documents, meetings, organizations, people, places, times, FIPA device ontology, HP CoolTown ontology and HP CoolAgent meeting ontology

### 4.2 Reasoning

Reasoning takes place when an agent needs to make sense out of the contextual information that is captured from the sensing infrastructure, and when an agent needs to determine its subsequent behavior.

When a piece of contextual information is captured from the physical environment, the raw contextual information is mapped into the corresponding RDF data model based on the CoolAgent RS ontology. For example, when the presence of a RFID (Radio Frequency Identification) badge is detected in the

HPL Cafeteria, the corresponding RDF data model can be constructed as the following:

```
<dev:RDF_Badge rdf:about='urn:badge_001'
    dev:badge_id='000565319'
    badge_label='Harry Chen'/>
<plc:Cafeteria rdf:about='urn:cafeteria_001'
    plc:name='HPL Cafeteria'/>
<plc:Is_In rdf:about='urn:is_in_001'>
    <plc:entity rdf:resource='urn:badge_001'/>
    <plc:location rdf:resource='urn:cafeteria_001'/>
</plc:Is_In>
```

To make sense out of the contextual information in RDF, the agents rely on the support from $P_{fc}$ and reasoning rules. The same approach applies when an agent needs to determine its subsequent behavior.

**Reasoning Over RDF** RDF is a foundation for processing meta-data. RDF itself is not a language; it is data model for representing meta-data [?]. At present, RDF data model is commonly represented in XML statements, RDF graphs, and triple statements.

Different RDF representations are suitable for different processing needs. The RDF XML representation is suitable for machine processing (parsing in particular) and for information exchange. RDF graphs provides a diagrammatic representation of the RDF data model, which is suitable for human visualization. On the other hand, the triple representation provides the means for logical reasoning.

To support reasoning over RDF, we developed our infrastructure based on the $P_{fc}$ [?] package and the SICStus Prolog system. We first constructed a set of Prolog rules that express the standard RDF model in first-order logic [?]. We called them *the basic RDF rules*.

For example, the `rdfs:subClassOf` property, which specifies a subset/superset relation between classes, can be interpreted by the $P_{fc}$ rules as

```
triple(A, subClassOf, B) => subClassOf(A,B).
triple(A, subClassOf, B), subClassOf(B,C) => subClassOf(A,C).
```

And the `rdf:type` property, which indicates a resource is a member of a class, can be interpreted by the $P_{fc}$ rules as

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

Domain specific rules, that allow agents to provide their own interpretation of the contextual information and to determine their subsequent behaviors, are constructed from the basic RDF rules. For example, domain specific reasoning that infers the ownership of a badge can be constructed using the following rules:

```
instanceOf(P,personClass) => person(P).
instanceOf(B,badgeClass) => badge(B).
instanceOf(R,ownsClass), triple(R,owner,P),
   triple(R,thing,T) => owns(P,T).

person(P), badge(B),owns(P,B) => badgeOwner(P,B).
```

**Implementing the Rule Shipping Technique** The FIPA Agent Communicative Acts and Interaction Protocols specifications [?] have defined the communication language and policy for agent communications. There are also FIPA specifications for different content languages. However, it is up to the implementation to provide an ontology, and the interpretation of messages using that ontology.

In the CoolAgent RS, the communication is mainly based on the FIPA Subscribe Interaction Protocol [?]. For example, the Venue Agent sends a `subscribe` message to the Entity Tracking Agent requesting to be notified when a person is present in a particular location. When a person is present, the Entity Tracking Agent notifies the Venue Agent by replying with an `inform` message. In both messages, the content messages, the request for notification and the reply to subscription, are to be defined by the CoolAgent RS.

To provide a flexible implementation for the Subscribe Interaction Protocol, we have developed a rule-based content message representation technique. This Rule Shipping technique allows an agent to send Prolog rules as the content of a `subscribe` message. The content rules are evaluated by the receiver to deduce new facts. Deduced facts are replied back to the sender as the content of a `inform` message. The following example illustrates the Rule Shipping Technique:

The Entity Tracking Agent has the following contextual information expressed in RDF triples:

```
K1:  triple('urn:loc1',rdf_type,place_Cafeteria)
K2:  triple('urn:loc1',place_name,'HP Cafeteria')
K3:  triple('urn:person1',rdf_type,people_Person)
K4:  triple('urn:person1',people_fname,'Harry')
K5:  triple('urn:person1',people_lname,'Chen')
K6:  triple('urn:utc1',rdf_type,times_UTC),
K7:  triple('urn:utc1',times_utc_value,'2001-03-27:T13:26:00Z')
K8:  triple('urn:isin1',rdf_type, place_Is_In)
K9:  triple('urn:isin1',place_entity','urn:person1')
K10: triple('urn:isin1',place_location','urn:loc1')
K11: triple('urn:isin1',times_start_time','urn:utc1')
```

The Venue Agent is interested in the people presences in a particular location. It subscribes the following forward-chaining rules as the content of its `subscribe` message:

```
R1: instanceOf(CafeteriaInst,place_Cafeteria),
    triple(CafeteriaInst,place_name,PlaceName)
      => cafeteria(CafeteriaInst,PlaceName).
```

```
R2: instanceOf(PersonInst,people_Person),
    triple(PersonInst,people_name,FName,LName)
      => person(PersonInst,FName,LName).
R3: instanceOf(UTCInst,times_UTC),
    triple(UTCInst,utc_value,UTCTime)
      => utc(UTCInst, UTCTime).
R4: instanceOf(IsInInst,place_Is_In),
    triple(IsInInst,place_entity,EntityInst),
    triple(IsInInst,place_location,LocationInst),
    triple(IsInInst,times_start_time,UTCInst)
      => isIn(IsInInst, EntityInst, LocationInst, UTCInst).
R5: isIn(IsInInst,PersonInst,LocationInst,UTCInst),
    utc(UTCInst,Time), cafeteria(LocationInst,LocName),
    person(PersonInst,FName,LName)
      => {add(shouldInform('com.hp.agent.palo-alto.va',
                           'com.hp.agent.palo-alto.eta',
                           'va.ruleid.131'
           msg(IsInInst, LocationInst, LocName, UTCInst, Time,
               PersonInst, FName, LName)))}.
```

Upon receiving the `subscribe` message, the Entity Tracking Agent asserts these rules into it Knowledge Base (KB). The forward-chaining system in the Entity Tracking Agent automatically adds new facts that can be deduced. Based on the triple statements `K1-K11`, the following facts can be deduced and are added to the KB:

```
N1: utc('urn:time1', '2001-03-27:T13:26:00Z')
N2: cafeteria('urn:loc1', 'HP Cafeteria')
N3: person('urn:person1', 'Harry', 'Chen')
N4: isIn('urn:isin1', 'urn:person1', 'urn:loc1', 'urn:time1')
N5: shouldInform('com.hp.agent.palo-alto.va',
                 'com.hp.agent.palo-alto.eta',
                 'va.ruleid.131',
                 msg('urn:isin1','urn:loc1','HP Cafeteria',
                     'urn:time1','2001-03-27:T13:26:00Z',
                     'urn:person1','Harry','Chen'))
```

`N1` says there is a *UTC* time instance referenced by the Unique Resource Identifier (URI) `'urn:time1'` with the UTC value `'2001-03-27:T13:26:00Z'`. `N2` says there is a *cafeteria* instance referenced by the URI `'urn:loc1'` with the venue name `'HP Cafeteria'`. `N3` says there is a *person* instance referenced by the URI `'urn:person1'` and with the first-name value `'Harry'` and the last-name value `'Chen'`. `N4` says a person referenced by URI `'urn:person1'` is in a location referenced by URI `'urn:loc1'` at UTC time referenced by URI `'urn:time1'`. `N5` says the Entity Tracking Agent should inform the Venue Agent, in reply to the message ID `va.ruleid.131`, the fact

```
msg('urn:isin1','urn:loc1','HP Cafeteria',
    'urn:time1','2001-03-27:T13:26:00Z',
    'urn:person1','Harry','Chen')
```

The Entity Tracking Agent queries for `shouldInform/4` whenever new facts are asserted to its KB. In the example above, as soon as the rules from the Venue Agent are asserted, the Entity Tracking Agent queries its KB for `shouldInform/4` and returns `N5`.

The above example shows how the Rule Shipping Technique exploits the triple representation of the RDF data model to enable reasoning. This technique allows the Venue Agent to construct customized reasoning rules to infer about people presences with the only requirement being that both the Venue Agent and Entity Tracking Agent share a common ontology.

## 5 Conclusion

Enabling context-awareness is one step closer to the realization of computing systems that can act in advance and anticipate users' needs. Ontology sharing, reasoning and sensing are the three important properties that the context-aware systems need to possess.

In this document we have described a context-aware multi-agent system, CoolAgent RS. We have demonstrate the feasibility of using RDF data model and $P_{fc}$ to support ontology sharing and reasoning in a context-aware system. We have presented the reasoning infrastructure in the CoolAgent RS, which enables distributed reasoning and knowledge sharing among agents.

### 5.1 Future Work

The Rule Shipping Technique provides the foundation for building distributed cooperative reasoning and knowledge sharing. In the current implementation, the sender is required to provide a complete set of reasoning rules to be evaluated by the receiver.

We believe such requirements impose potential scalability issues on the communication message size and the rule construction overhead. One of our immediate objectives is to develop a more effective and scalable distributed cooperative reasoning and knowledge sharing infrastructure to support context-aware systems. Moreover, it provides scope for doing research on security considerations under this scheme.

## 6 Acknowledgment

The research work described in this document was conducted by Harry Chen and Sovrin Tolia as part of their research internships at Hewlett-Packard Laboratories. The active support from all the members of the Agents for Mobility Group and Cooltown Group at HPL is gratefully acknowledged.

Harry Chen would like to thank HPL for the fellowship support that helps him to begin his initial research work on developing context-aware software agents.