A Multi-agent Context-aware Service Platform in a Smart Space

Wan-rong Jih[1], Jane Yung-jen Hsu[1]*, Tsu-Chang Lee[2], and Li-lu Chen[1]

[1] Department of Computer Science and Information Engineering
National Taiwan University
Taiwan
jih@agents.csie.ntu.edu.tw

[2] ViVoDa Communications, Inc.
USA
tclee@vvdcomm.com

Received 12 April 2007; Revised 26 April 2007; Accepted 26 April 2007

Abstract. Ubiquitous computing technology plays a key role for providing context information and makes context-aware services can be delivered in a smart space. As the contexts changing rapidly, context resources can be gathered from sensors, mobile devices, and personal information softwares. Consequently, a context-aware system must be aware of such environment changes so that it can provide adaptive and proactive services to the users. In addition, all the inner computing operations have to be hidden behind the users.
We propose a Context-aware Service Platform, which is implemented in JADE agent platform and utilizes Semantic Web technologies, to analyze the ambient contexts and contrive service plans. We integrated ontology and rule-based reasoning to automatically infer high-level contexts and deduce a goal of context-aware services. An AI planner decomposes complex services and establishes the execution plan. Agents perform the specified task to accomplish the service. A scalable / distributed hardware architecture implements the Context-aware Service Platform, and supports applications with distributed multi-agents embodied in objects throughout the smart space environment. A *Smart Alarm Clock* scenario demonstrates the detail functions of each agent and shows how these agents incorporate with each others.

Keywords: Multi-agent, Context-aware, Semantic Web, Ontology, Web Service, Smart Space

1 Introduction

The world's information infrastructure continues to fragment, and various information can be collected by using tiny, battery-powered, and low-cost mobile computer devices, such as PDAs, smart phones, and wireless sensors. Utilizing the information of a physical environment, for example, temperature, humidity, monitoring light or any other environmental factor, can provide more intelligent and adaptive services to users.

In a smart space, augmented appliances, stationary computers, and mobile sensors provide raw context information, and a context-aware system must understand the meaning of a context. That is, a way to represent contexts is our first issue.

A smart space should have capability for inferring the raw context to high-level context. For instance, what is the activity in the room? Where is the user? Context-aware services require the high level description about the user's state and the environmental condition. How to infer such higher level context is the second issue.

Services are built on different platforms and run on their original computers without needing to migrate applications to personal devices. Each service must use exactly the standard communication mechanism for communicating with others. Transmission of service requests and the results is sent via wireless networks, such as WiFi, Bluetooth, and ZigBee. Furthermore, different applications are concerned with different types of knowledge and their corresponding representation models, whereas the messages between services should be understood by each other. Therefore, how to handle such knowledge sharing and maintain the consistency of knowledge is another issue.

This research explores the roles of intelligent sensing, wireless communicating, mobile and ubiquitous computing in smart home services. We introduce the Context-aware Service Platform, which provides context-aware services to the resident in an intelligent space.

Interaction between the user and services in the smart space is through a wide variety of appliances for data gathering and information presentation. Services of the environment tracks the location and specific activities of the

* Correspondence author

user through sensors, such as pressure-sensitive seats, bed sensors, infrared remote controllers, and smart gadgets. Meanwhile, the user receives multimedia messages or content through speakers and monitors, as well as smart home devices. *Context-aware computing* enables the services in the intelligent environment to respond, at the right time and in the right place, to the user's needs based on the collected sensor data. The Context-aware Service Platform is the solution and provides the implementation architecture to achieve the goal for fully supporting the demands of the user's daily life in the smart space.

The rest of this paper is organized as follows: Section 2 discusses the technologies to build the Context-aware Service Platform. The infrastructure of this platform is introduced in Section 3 and the next section concentrates on the design of each functional component. Section 5 describes a scalable / distributed server architecture for implementing the Context-aware Service Platform. Section 6 demonstrates a *Smart Alarm Clock* scenario and its detailed design. Finally, Section 7 and 8 list the related work and a conclusion, respectively.

2 Technologies Overview

An overview of the context-aware systems, service-oriented architectures, and context model are introduced in this section.

2.1 Context-Aware Systems

During the past years, a number of context-aware systems have been developed to support pervasive computing and ambient intelligent environments such as Active Badge location system[1], PARCTAB[2],and Context Toolkit[3]. These systems utilize various sensors and devices to provide location-aware services but lack knowledge sharing and context reasoning.

Typical researches dealing with context reasoning are Easy Meeting[4][5] and MyCampus e-Wallet[6][7]. Easy Meeting is a prototype of an intelligent meeting room that built on the Context Broker Architecture (CoBrA). An agent-based broker maintains all the context knowledge represented in RDF-triple and utilizes Jena and Jess to support context reasoning. Besides, the policy of SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications)[8][9] is used to control the sharing of users' contextual information. The MyCampus at Carnegie Mellon University has been developed to provide mobile context-aware services for the community in the university and the e-Wallet is its key element that supports access control and obfuscation rules for user's privacy preferences. Both Easy Meeting and MyCampus deployed context reasoning on their systems, but they did not consider the needs of service integration.

2.2 Web Services

The concept of service integration can be constructed in a service-oriented architecture (SOA). It is a loosely-coupled architecture and services in distributed systems communicate through message passing. Services communicate and collaborate with each other to solve the assigned tasks. Web services technology enables the connection of services.

World Wide Web Consortium (W3C) defines several Web server related specifications. Simple Object Access Protocol (SOAP)[3] is for exchanging structured information in a decentralized, distributed environment. Web Service Definition Language (WSDL)[4] is an XML format standard to describe network services and the way to access them. OWL-based Web Service Ontology, OWL-S[5] is an ontology of services that makes software can discover, invoke, compose, and monitor Web resources. OASIS Technical Committees define the Universal Description, Discovery and Integration (UDDI)[6] which forms the necessary technical foundation for publication and discovery of Web services implementations both within and between enterprises.

To make service integration, a framework needs to describe a standard ontology for declaring and sharing services. A reasoning application can be built into the framework, so that it can automatically determine the logical consequences of ontologies.

[3] http://www.w3.org/TR/soap/
[4] http://www.w3.org/TR/wsdl
[5] http://www.w3.org/Submission/OWL-S/
[6] http://www.uddi.org/specification.html

2.3 Context Models

In order to know the changes of environment, researchers usually define context models to represent the context information. Typically, time and location are the most well-known context models. Temporal reasoning plays an essential role in context-aware systems. DAML time ontology[10] and ISO 8601 date and time formats[11] are the popular structures and standards. Two important temporal models are point-based and interval-based time models. Traditional time structure is based on a set of points, Bry *et al.*[12] introduce CaTTs and treat the cultural calendars as interval-based time. Ma and Hayes[13] analyze the temporal interval-based models in a recent literature report.

Many context-aware systems concentrate on location aware services. Maryland Information and Network Dynamics Lab Semantic Web Agents Project (mindswap) Group[14] develops Semantic geoStuff to express basic geographic features such as countries, cities, and relationships between these spatial descriptors. The Open Geospatial Consortium, Inc. (OGC)[15] prescribes OpenGIS specifications for GIS data exchange and process, and OpenCyc Spatial Relations[16] specify the vocabularies of spatial objects and relations.

2.4 Personal Services

Many people see that the future of Internet access is through the small personal devices, such as PDAs, smart phones, iPods, and Palms. Compared to desktop computers, personal devices are more compact in size, but their CPU and memory are much less powerful. Consequently, the available tools for personal devices are fewer than for the desktop, and most of the desktop applications can not scale well to mobile devices.

In order to circumvent the limitation of personal devices, we deploy a service integration architecture to provide context-aware services. There is a growing collection of context-aware services that users can subscribe to on their personal devices and where they can access personal information (*e.g.* current location, health status, and habits). These are subject to privacy preferences. How to let users control the sharing and the use of their contextual information is the problem with which we must deal.

3 Smart Space Infrastructure

Fig. 1 shows the infrastructure of a Context-aware Service Platform in a smart space. Context resources can be obtained from the computing softwares, such as personal calendar, weather forecasts, location tracking system, personal friend list, and shopping list, as well as raw sensor data. Context Collection Agents obtain raw contexts from softwares and hardware sensors, and raw data are converted into a semantic representation. A Context-aware Service Platform is continuous collecting these contexts and infers appropriate Service Applications, and then it automatically and proactively delivers the services to users.

A *Context-aware Service Platform* contains the following components:

Message Transportation provides a well-defined protocol for maintaining a set of communicative acts. Moreover, a common message structure is defined to exchange messages over the Context-aware Service Platform. Common message structure contains sender, receiver, the type of the communicative act, message content, and description of content. In this platform, each component communicates with each other through message passing.

Life Cycle Management maintains a White Pages and states of services to control over access to and use of the services. A service can be in one of the following states: initiated, active, waiting, suspended, and deleted state. Life Cycle Management reflects the state changes and controls the state transitions. Consequently, every component is controlled by life cycle management.

Rule-based Engine uses IF-THEN rule statements, which are simply patterns and the inference engine performs the process of matching the new or existing facts against rules. Similar to DL reasoners, rule engines can also deduce high-level contexts from low-level contexts; the major difference is, rule engines can handle complex reasoning (*e.g.* combining several contexts to deduce higher-level contexts) while the DL reasoners can not. The derived high-level contexts are asserted into **Context Knowledge Base** which serves as a persistent storage for context information. Rules for which and when the appropriate service can be invoked are defined as the knowledge of service invocation rules for the Rule-based Engine.

Ontologies are loaded into the **DL reasoner** to deduce high-level context from low-level context. DL reasoner provides the inference services to ensure the ontology does not contain contradictory facts. The class hierarchy of an ontology can be used to answer queries by checking the subclass relations between classes. Besides, computes the direct types of every ontology instance can support to find the specific class that an instance belongs to.

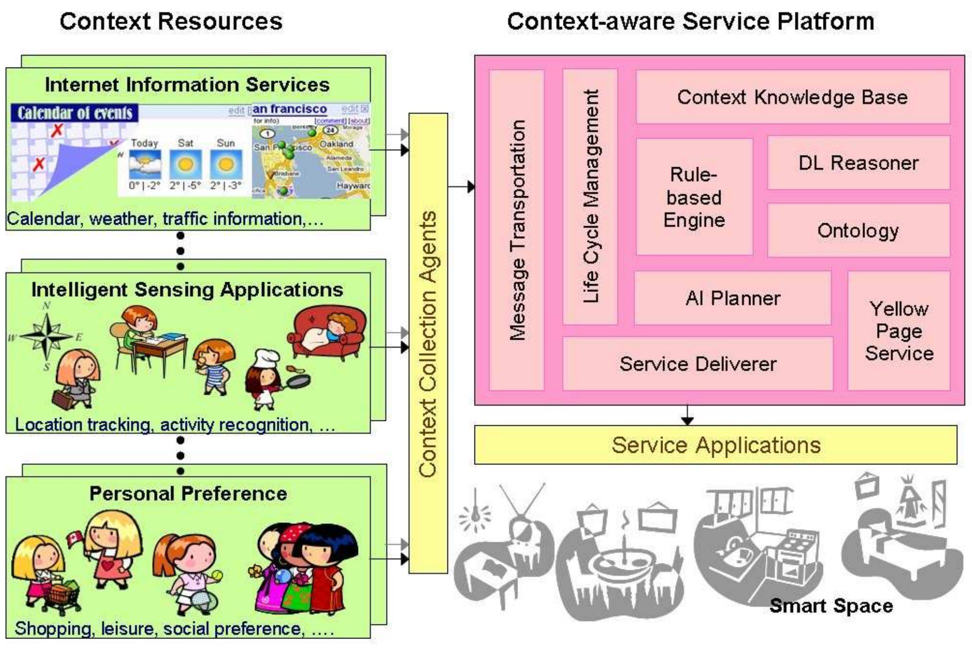

Fig. 1. A Smart Space Infrastructure

Yellow Pages Service provides the functions for service registration and discovery. New services register their services to Yellow Pages Service. Service Deliverer and other services can search the desired services and get the results.

AI Planner generates a service composition plan sequence which satisfies a given goal. **Service Deliverer** chooses a service to execution from the service candidate list which returns from Yellow Pages Service.

4 Context-aware Service Platform

The Foundation for Intelligent Physical Agents (FIPA[7]) develops computer software standards to promote the interoperation of heterogeneous agents and the services that they can represent. The Java Agent DEvelopment Framework (JADE[8]) is a FIPA-compliant software framework for multi-agent systems, implemented in Java and comprised serval agents. An Agent Management System (AMS) controls agents' life cycle and plays the role of white pages service, Directory Facilitator (DF) provides yellow pages service to other agents, an Agent Communication Channel (ACC) is the agent which can provide the path for basic contact between agents, and an Agent Communication Language (ACL) has been specified for setting out the message formats, consists of encoding, semantics, and parameters.

Design of the Context-aware Service Platform shows in Fig. 2. The top block depicts a smart space environment. *Context Resource* can be collected by Context Collection Agents that receive sensors or software data and will be delivered to *Context-aware Reasoning* model. Context Collection Agents are device dependent agents; each agent will be associated with different types of devices for providing raw sensor data. Ontology Agent and Context Reasoner can infer high-level context to provide goals for *Service Planning*. Context information and service description are stored in *Context Knowledge Base*. After perform service composition, discovery, and delivery, a context-aware service will be delivered to the specified Service Applications.

4.1 Context-aware Reasoning

We deploy Jess[9] in our Context-aware Service Platform. Jess is a forward-chaining rule engine used Rete algorithm[17] to process rules, which is a very efficient algorithm for solving the difficult many-to-many matching problem. Moreover, an open-source OWL-DL reasoner Pellet[10] is developed by Mindswap Lab at University of

[7] http://www.fipa.org/
[8] http://jade.tilab.com/
[9] http://herzberg.ca.sandia.gov/
[10] http://pellet.owldl.com/

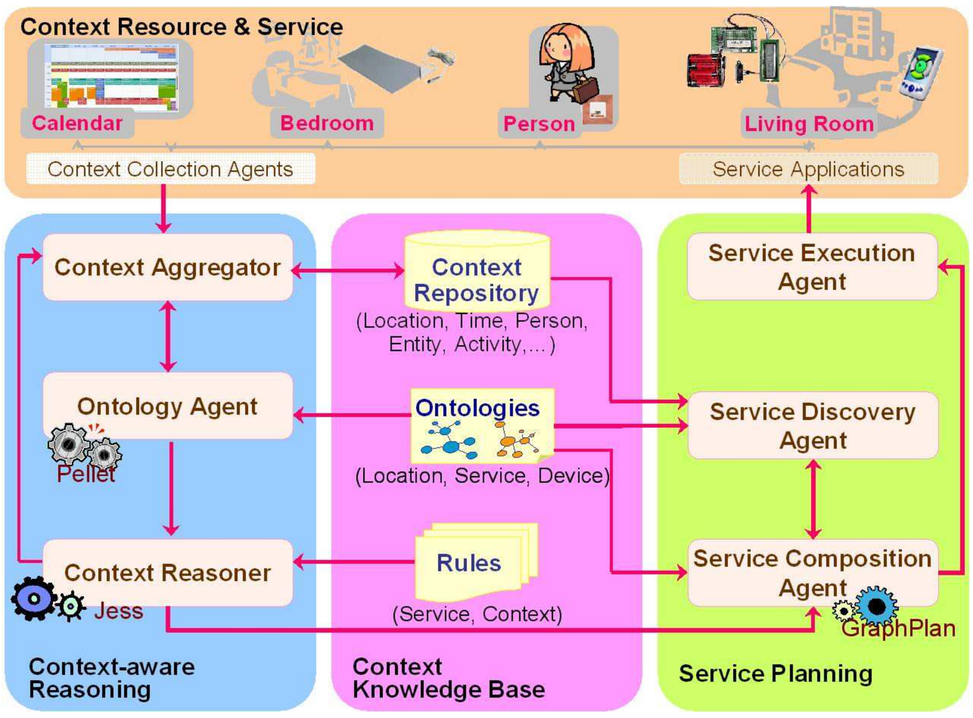

Fig. 2. Functional Flow of Context-aware Service Platform

Maryland. Finally, Jena[11] is a Java framework for building Semantic Web applications, is used for providing a programmatic environment for RDF, RDFS, and OWL.

Context Aggregator

Initialization : A configuration file declares the type of context that the Context Aggregator will received and subscribe the context to its corresponding Context Collection Agent (refer to Fig. 1).
Input message : There are two types of input contexts: (1) Raw context refers to data obtained directly from context sensors or software, such as bed sensor data and forecast data, can be delivered from a bed sensor and weather API respectively. Senders of the these low-level contexts are called Context Collection Agents and the data will be wrapped as RDF-triple in message content. (2) High-level context is the information inferred from raw context, such as "location of a furniture" and "activity who currently participate in", can be inferred from ontology reasoner and rule-based reasoner respectively.
Process : Value of a context can be changed at anytime and anywhere. Consequently, Context Aggregator must collect contexts and maintain the consistency of current context. Either raw or high-level context has an unique type identity and value. The associated value will be replaced when new context is arrived.
Output message : While raw contexts received from Context Collection Agents, the new context is immediately stored in Context Repository and a set of current context is delivered to Ontology Agent.

Ontology Agent

Initialization : An OWL context ontology describes structure and relation between contexts that will be loaded and parsed into RDF triples by Jena API. It also subscribes to Context Aggregator for the contexts that has been declared in the context ontology. In addition to ontology loading, it has to start a DL reasoner for supporting ontology query.
Input message : Context Aggregator sends the current state of contexts when any subscribed context has been changed.
Process : There are two types of ontology reasoning that perform in Ontology Agent, so they can provide high-level context. The first is inferred by Jena API that deduces high-level context from the object property of context, such as "bed sensor is attached to a bed" and "bed is placed in a bedroom". Relationships between

[11] http://jena.sourceforge.net/

the instances of context object are defined in context ontology. A DL reasoner, *i.e.* Pellet, deduces the inferred superclasses of a class and decides whether or not one class is subsumed by another, for example, "living room is an indoor location".
Output message : If there is no high-level context to be derived, the input message of current contexts will be redirected to Context Reasoner. Otherwise, the new high-level contexts must be delivered to Context Aggregator.

Context Reasoner

Initialization : The Jess API provides packages to load a rule-based engine when Context Reasoner is started up.
Input message : A set of current context, which is the same as input message of Ontology Agent.
Process : The input context must be wrapped as Jess rule format and assert into the rule-based engine, and may trigger the execution of rules that can infer new contexts or derive a goal of service.
Output message : New high-level contexts are sent to Context Aggregator, whereas the service goal is delivered to Service Composition Agent.

4.2 Service Planning

The modern trend of software is to build a platform-independent architecture that distributes software components on the Internet. New services and functionalities can be automatically achieved by selecting and combining a set of available software components.

A service functionality contains a semantic annotation of what it does and a functional annotation of how it behaves. OWL-S(formerly DAML-S) is an ontology for services, and it provides three essential types of knowledge about a service: service profile, process model, and service grounding. An OWL-S service profile illustrates the preconditions required by the service and the expected effects that result from the execution of the service. A process model describes how services interact and how the functionalities offer that can be exploited to solve the goals. The role of service grounding is to provide concrete details of message formats and protocols.

According to these semantic annotations, AI planning has been investigated for composing services. Graphplan[18] is a general purpose graph-based planner. The state transition is defined by operators consists of preconditions and effects. Given initial states, goals, and operations, a planning system returns a service execution plan, which is a sequence of actions that starts from the initial states and accomplishes the given goals.

Service Composition Agent

Initialization : An OWL-S service ontology represents all available services and the service profile describes service goal, preconditions, and effects for AI planner, *i.e.* Graphplan. Consequently, the service ontology must be parsed and transferred into Graphplan operations.
Input message : A service goal is sent by Context Reasoner.
Process : According to the service operations, Graphplan creates a sequence of operation to achieve the goal.
Output message : When a operation represents a composite service, the corresponding service model will be delivered to Service Discovery Agent. If the execution plan has been generated, it delivers a sequence of service to Service Execution Agent.

Service Discovery Agent

Initialization : According to the description of the service model in service ontology, information of every atomic service will be kept in this agent.
Input message : A composite service model receives from Service Composition Agent.
Process : Given the composite service, Service Discovery Agent searches the atomic processes that are available and can carry out the composite service.
Output message : The atomic services, which can accomplish the given composite service, must be delivered to Service Composition Agent.

Service Execution Agent

Initialization : Service ontology consists of the description of service grounding, which specifies the details of how an agent can access a service.
Input message : A service list is sent by Service Composition Agent.
Process : Following the control sequence of services, the corresponding device agents will be invoked for providing atomic service.
Output message : The input parameters, invoking atomic service, must be passed to the device agent.

4.3 Context Knowledge Base

Context Repository Context Repository contains a consistency of context, includeing location, time, person, and activity information. A RDF-triple represents contexts of the repository, like a subject, a predicate, and an object. Subject is a resource named by a URI with an optional anchor identity. The predicate is a property of the resource, and the object is the value of the property for the resource. The following triple represents "Peter is sleeping".

```
<http://...#Peter>
<http://...#participatesIn>
<http://...#sleeping>
```

Where `Peter` represents subject, `participatesIn` is a predicate, and object `sleeping` is an activity. According to the elements of RDF-triple, we use subject and predicate as the compound key of Context Repository.

Ontologies An ontology is a data model that represents a domain and is used to reason about the objects in that domain and their relations. We defines a context ontology depicts in Fig. 3 as a representation of common concepts about the smart space environment. Context information are collected from real-world classes (Person, Location,

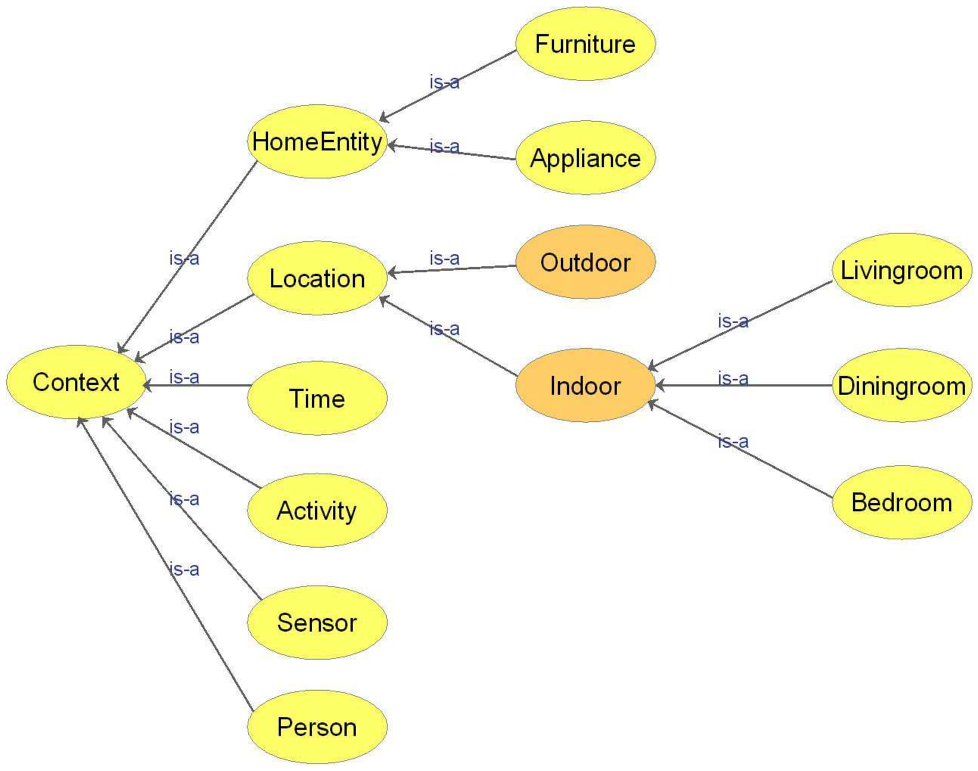

Fig. 3. A Context Ontology

Sensor, Time, HomeEntity), and a conceptual class Activity. The class hierarchy represents an `is-a` relation; an arrow points from a subclass to another superclass. A class can have subclasses that represent the concepts more specific than their superclass. For example, we can divide the classes of all locations into indoor and outdoor locations, that is, Indoor Location and Outdoor Location are two disjoint classes and both of them belong to Location class. In addition, the subclass relation is transitive, therefore, the Livingroom is a subclass of Location class because Livingroom is a subclass of Indoor and Indoor is a subclass of Location.

A service ontology defined by OWL-S is for describing available services that comprises service profile, service model, and service grounding, which has been stated in Section 4.2

Rules Rules of a rule-based system serve as IF-THEN statements. Context rules can be triggered to infer high-level context. A rule, detecting whether a user is sleeping or not, is showed as follows:

```
(defrule User_is_sleeping
  (triple
   (subject ?person)
   (predicate "http://www.w3.org/...#type")
   (object    "http://...#Person")
  )
  (triple
   (subject ?person)
   (predicate "http://...#isLocatedIn")
   (object    "http://...#bedroom")
  )
  (triple
   (subject "http://...#bed_sensor")
   (predicate "http://...#isOn")
   (object    "#true^^http://...#boolean")
  )
  =>
  (assert
    (triple
     (subject  ?person)
     (predicate "http://...#participatesIn")
     (object "#sleeping")
    )
  )
)
```

Patterns before ==> are the conditions, matched by a specific rule, called left hand side (LHS) of the rule. On the other hand, patterns after the ==> are the statements that may be fired, called right hand side (RHS) of the rule. If all the LHS conditions are matched, then the actions of RHS will be executed. The RHS statement can be either asserted new high-level contexts or delivered a service goal.

The `User_is_sleeping` rule shows that if `?person` is a person, and `?person` is in bedroom, and the bed sensor is on, then asserts the "`?person` is sleeping" as a fact.

5 Hardware Implementation

For practical implementation of the platform proposed in this paper, we need to consider system efficiency and throughout,in particular, when the context knowledge base grows in complexity and the service becomes more demanding. In this section, we propose a scalable and distributed Server architecture to accelerate the Context-aware services presented in this paper. The hardware server architecture shown below is a specific application of the more general system structure presented in a pending patent application[19].

Fig. 4 shows the overall diagram of a Context-aware Service System (CSS), in which the Context-aware Server (CaS) is an implementation of the Context-aware Service Platform presented in this paper. Here CaS is connected to certain communication network to access a bundle of context resources (shown as the Context Resource Reservoir - CRR). CaS can provide context-aware services to enable Context-aware application agents to perform certain Context-aware service applications in a Smart Space Operation Environment.

Fig. 5 shows the architecture of the Context-aware Server (CaS). Here the Context-aware Service Processor (CaSP) is special purpose hardware to process the computation demanding tasks of Context-aware Reasoning and Service Planning. The Message Transportation Channel (MTC) is a communication channel to provide message

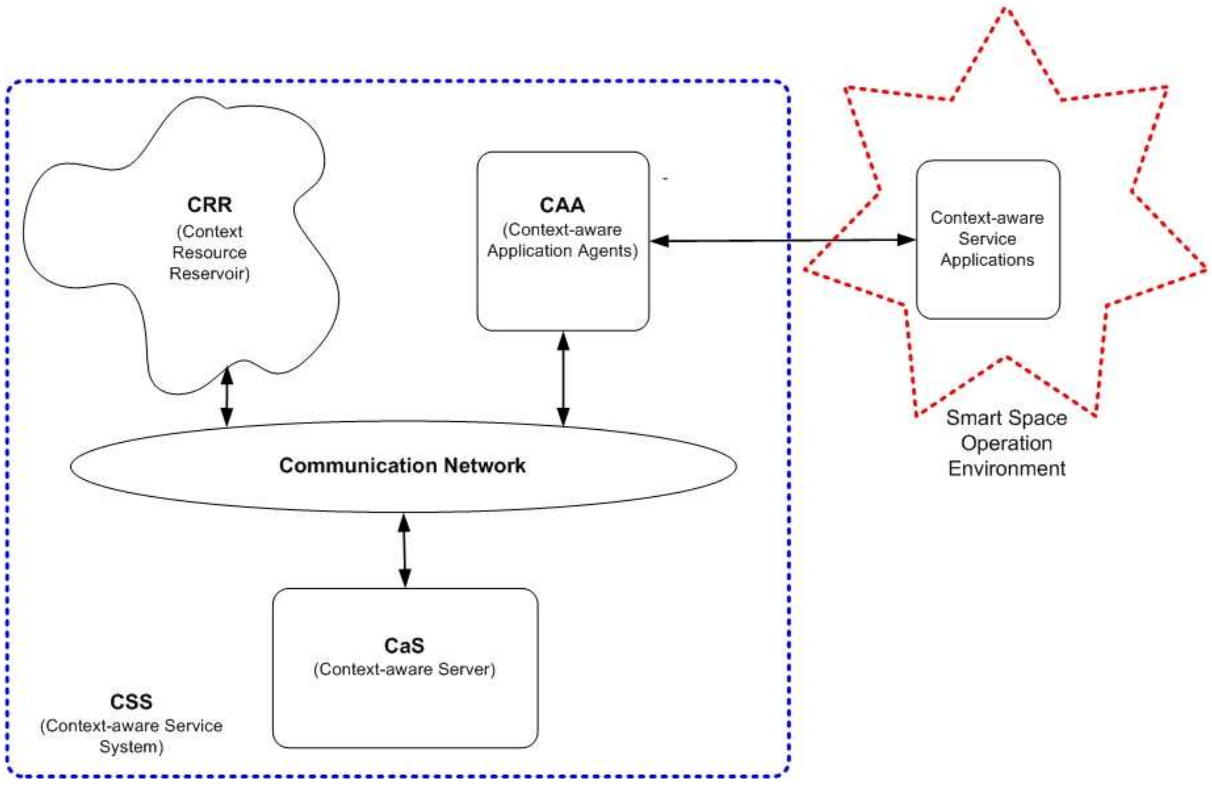
CRR
(Context Resource Reservoir)
CAA
(Context-aware Application Agents)
Context-aware Service Applications
Smart Space Operation Environment
Communication Network
CaS
(Context-aware Server)
CSS
(Context-aware Service System)

Fig. 4. Context-Aware Service System Structure.

transporting paths to the Context Resources and Application Agents on the communication network. The Reasoning Object Mapper (ROM) provides mappings between the context resource space and the objects processed by the Context-aware Service Processor (CaSP). The Service Object Mapper (SOM) provides mappings between the service space and the objects processed by the Context-aware Service Processor (CaSP). The ROM and SOM are special purpose object memory mapping circuits to access a large amount of data distributed through multiple storage hierarchies throughout the communication network.

In most of the smart space applications, the resources for service agents might be distributed throughout the communication network. The CaS are connected to form a Context-aware Server NETwork (CaSNET). In this type of configurations, the Context-aware services are provided by the cooperation of multiple Context-aware Servers (CaS) on the network. Fig. 6 shows a demonstrative example of services provided by multiple CaS on the network. Both “Push” and “Pull” operations are possible on the CasNET. Each CaS can “Push” services up stream to other CaS via its SOM. For example, CaS_3 can “Push” the service to CaS_1 to display on a TV automatically. A CaS can also pull the services via its ROM down stream from other CaS. For example, CaS_2 can ”Pull” services from CaS_3 to server human users according to their requests. When implemented in SoC (System on Chip),and integrated with memory, sensors or MEMS (Micro-Electro-Mechanical System) devices using advanced SiP (System in Package) technology, the Context-aware Server (CaS) can be embodied into objects and places throughout the smart space. As shown in this example, Ca_9 could be attached to a house for security monitoring; CaS_8 could be a monitor embodied in a house dog’s collar with built in GPS function; CaS_7 could be running on a PC to access certain household data bases, etc.

6 Demonstration Scenario

We use a simple examples to illustrate detailed picture of Context-aware Service Platform.

In a smart space, a Smart Alarm Clock can check Peter’s schedule and set the alarm one hour prior to the daily first task. If Peter does not wake up within 5-minute period after the alarm is sent, send another sound of alarm and increases its volume. If Peter wake up early then the alarm time, there will be no more alarm.
On the other hand, when the first task event is approaching and the sensors detect that Peter remains sleeping, an exceptional control should deal with such situation.

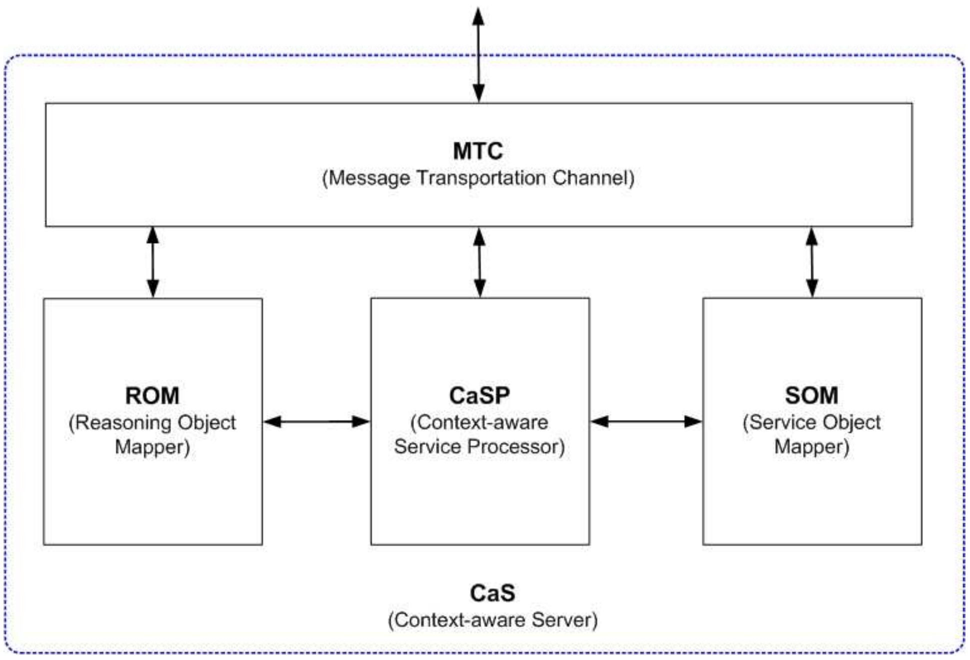
MTC
(Message Transportation Channel)
ROM
(Reasoning Object Mapper)
CaSP
(Context-aware Service Processor)
SOM
(Service Object Mapper)
CaS
(Context-aware Server)

Fig. 5. Context-Aware Server Hardware Architecture.

In addition to send the alarm through traditional alarm clock, the alarm service can deliver to the devices that in Peter's bedroom, such as radio, stereo, speaker, or personal mobile device.

6.1 Context-aware Reasoning

In order to archive *Smart Alarm Clock*, we have to collect Peter's schedule to decide the alarm time and should reasoning whether Peter is awake or not. Google Calendar Data API supports on-line schedule information and position-aware sensors, bed pressure sensors, etc., can detect whether user on the bed or not. RFID technologies[20] can be used to recognize and identify the activities of Peter while a wireless-based indoor location tracking system can determine Peter's location with room-level precision[21].

In order to know whether Peter is sleeping or not, all the related instances form an ontology instance network, shows in Fig. 7. Dashed line indicates the connection of a class and its instance. Word in the box depicts a instance of its corresponding class, for example, bed is an instance of Furniture class. Each solid arrow reflects the direction of `owl:ObjectProperty` relationship, from domain to range and a inverse property can be declared while specifies the domain and range classes. For example, a sensor bed_sensor is attached to (`isAttachedTo`) furniture bed, and the inverse property is `hasSensor`. A boolean data type property isOn is associated with Sensor class for detecting whether the instances of class is on or off.

If someone is on the bed and the sensor bed_sensor is on, then the value of isOn is `true`. On the other hand, when nobody touches the bed, the value of isOn has to be `false` and it infers that Peter is awake. Therefore, the system is not necessary to deliver the alarm_service.

Suppose that calendar agent reports the first event of the day will be held at 8:00am, therefore, the alarm is set to 7:00am. When the time is up, given the location of Peter and the status of bed sensor, the rule `User_is_sleeping` in Section 4.3 can deduce that whether Peter is sleeping or not. Assume that there is another rule reflects that "if Peter is sleeping, then deliver smart alarm service". Consequently, the service goal of *Smart Alarm Clock* can be derived and deliver to Service Composition Agent.

If Peter does not wake up for the task, according to the owner and importance of this task, an exceptional handling rules will be triggered for deciding whether to postpone or cancel this coming task. For instance, Peter has to host a meeting at 8:00am and hence the task is a high priority event. Consequently, a rule will be triggered to postpone the meeting, and an emergency message will be sent to the corresponding participants for informing the situation. On the contrary, if this task is "watch TV show at 8:00am" with low priority, then the context-aware reasoning infers that this scheduled task should be canceled and a video recording event will be invoked.

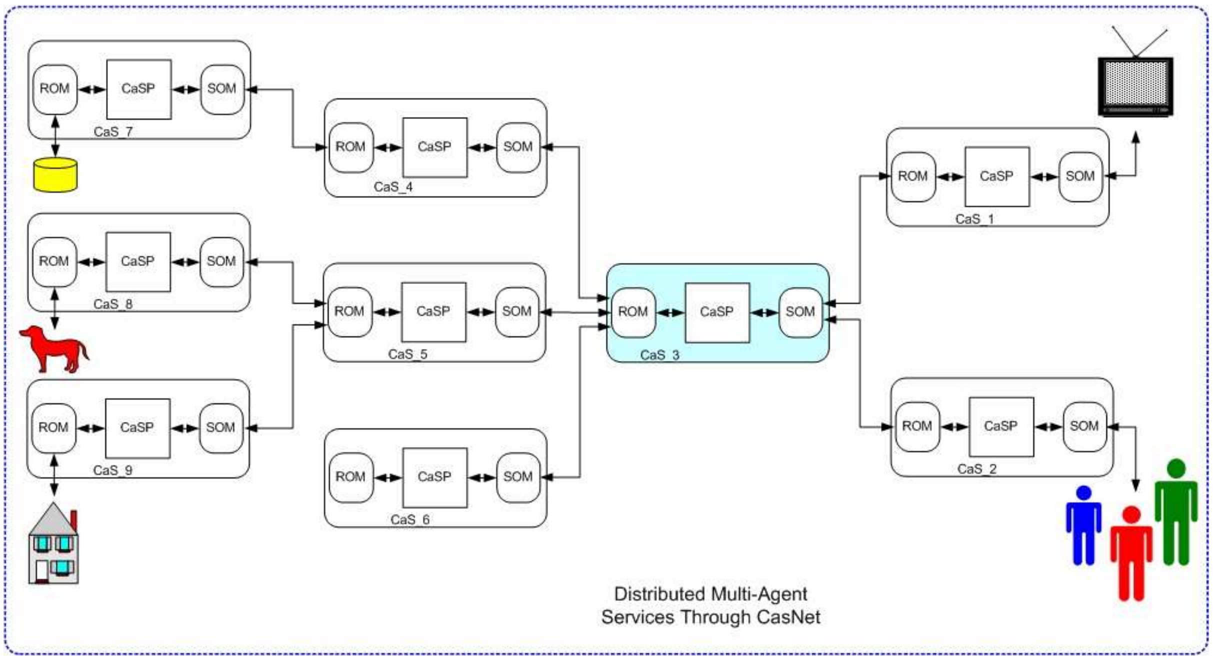

Fig. 6. Multi-Agent Services through CaSNET (Context-aware Server NETwork).

6.2 Service Planning

Operations for planner can be derived from service profile, which gives a brief description about the service and consists of service name, preconditions, effects, inputs, and outputs of the service. The following OWL-S statements indicates the profile of *Smart Alarm Clock*.

```
<profile:Profile rdf:ID="alarm">
<profile:serviceName
  rdf:datatype="http://www.w3.org/...#string">
    smart alarm
</profile:serviceName>
<profile:hasPrecondition
    rdf:resource="#alarm_precond"/>
<profile:hasInput>
   <process:Input rdf:ID="message_stream">
      <process:parameterType rdf:datatype=
          "http://www.w3.org/...#anyURI">
           http://...#MessageStream
      </process:parameterType>
   </process:Input>
</profile:hasInput>
           ....
</profile:Profile>
```

This example shows a service named `smart alarm` has precondition `alarm_precond` and its input parameter belongs to `MessageStream` class.

Service model gives detailed description of the service and each service is modeled as process. There are three types of process: atomic process is the primary process without any subprocess, simple process are used as elements of abstraction, it can either represents as atomic process or composite process, and composite process consists of subprocesses. A composite process can be decomposed by using control operators such as `sequence`, `split`, `split+join`, `choice`, `any order`, `if-then`, `iterate`, `repeat-until`, and `repeat-while`. Figure 8 is the control flow for *Smart Alarm Clock*, it uses operator `choice` to compose the process. `TextMessageProcess`, `VideoPlayerProcess`, and `AudioPlayerProcess` are atomic process, and *Smart Alarm Clock* can be served by using one of the three atomic processes. An example of `AudioPlayerProcess` shows as follows.

```
<process:AtomicProcess rdf:ID="AudioPlayerProcess">
```

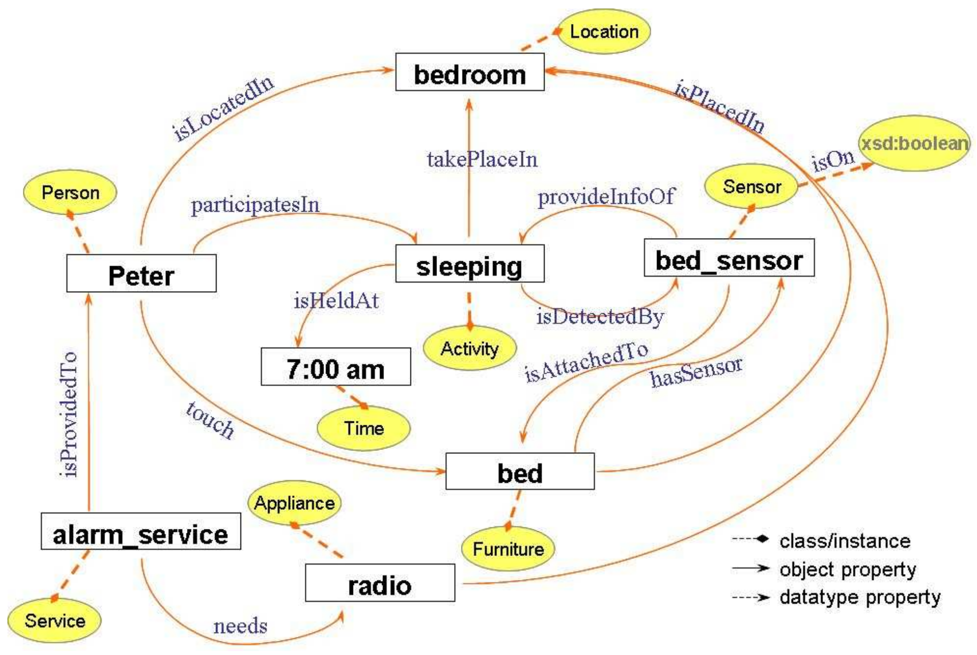

Fig. 7. Instance Network about Sleeping

```
    <process:hasInput>
      <process:Input rdf:ID="audio_stream">
        <process:parameterType
            rdf:datatype="http://...#anyURI">
               http://...#AudioStream
        </process:parameterType>
      </process:Input>
    </process:hasInput>
    <process:hasPrecondition>
       <expr:KIF-Condition rdf:ID="alarm_precond">
         <expr:VariableBinding
             rdf:ID="isSleepVariablebinding">
           <expr:theObject
             rdf:resource="http://...#sleeping"/>
           <expr:theVariable rdf:datatype=
                     "http://...#boolean"> true
             </expr:theVariable>
           </expr:VariableBinding>
         <expr:expressionData
             rdf:datatype="http://...#string">
             precondition of smart alarm
         </expr:expressionData>
       </expr:KIF-Condition>
     </process:hasPrecondition>
     <process:hasResult>
       <process:Result rdf:ID="alarm_done"/>
     </process:hasResult>
       ......
  </process:AtomicProcess>
```

Descriptions of atomic process are similar with that of profile, except the service model describes process in more details. For example the input data type of `AudioPlayer Process` is belong to `AudioStream` class, whereas the alarm service profile only gives an upper-level data type `Message Stream`. Moreover, atomic

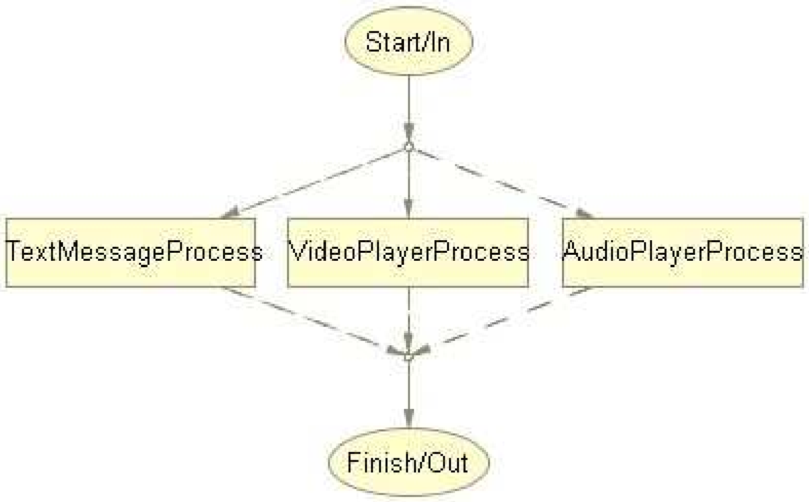

Fig. 8. Process Graph of Smart Alarm Clock

process describes detailed expression of preconditions, for instance, it binds an instance `sleeping` of Activity class to a boolean variable.

Service grounding specifies the details of the way to access the service, and deals with the concrete level of specification. Both OWL-S and WSDL are XML-based languages, therefore, the OWL-S service is easy to bind with WSDL service, for example:

```
<grounding:WsdlGrounding
       rdf:ID="AudioPlayerWSDLgrounding">
  <service:supportedBy
       rdf:resource="#AudioPlayer"/>
  <grounding:hasAtomicProcessGrounding>
     <grounding:WsdlAtomicProcessGrounding
            rdf:ID="WsdlAtomicProcessGrounding">
        <grounding:owlsProcess
            rdf:resource="#AudioPlayerProcess"/>
        <grounding:wsdlOperation>
           <grounding:operation
               rdf:datatype="http://...#anyURI">
                            play
           </grounding:operation>
           <grounding:portType
               rdf:datatype="http://...#anyURI">
                         audio player port type
           </grounding:portType>
        </grounding:wsdlOperation>
     </grounding:WsdlAtomicProcessGrounding>
  </grounding:hasAtomicProcessGrounding>
    ......
</grounding:WsdlGrounding>
```

A WSDL service has construction of `type`, `message`, `operation`, `port type`, `binding`, and `service`. The `AudioPlayerWSDL grounding` briefly shows that OWL-S has provided `operation` and portType mapping. Besides, the XSLT can help the transformation from WSDL descriptions to OWL-S parameters.

7 Related Work

Smart spaces can be the houses, workplaces, cities, vehicles, and the spaces deploy embedded sensors, augmented appliances, stationary computers, and mobile devices to gather contexts of the user. Each place has different challenges, but similar technologies and design strategies can be applied. In order to make the space have capabilities to respond to the complexities of life, researchers explore new technologies, materials, and strategies to make the idea possible.

Department of Architecture research group at Massachusetts Institute of Technology proposes the House_n[12] research, which includes a “living laboratory” residential home research facility called the PlaceLab[22]. Hundreds of sensing components are installed in nearly every part of the house. Interior conditions of the house can be captured by using these sensors, such as temperature, light, humidity, pressure, electrical current, water flow, and gas flow sensors. Eighty wired switches can detect the opening of the refrigerator, the shutting of the linen closet, or the lighting of a stovetop burner events. Cameras and microphones are embedded in the house for recording the resident’s movement. Twenty computers collects all the data streams from these devices and sensors to provide multi-disciplinary research, for example, monitoring the resident’s behavior, activity recognition, and dietary status. Besides, Aware Home[13][23] was proposed by the Future Computing Environments Group at Georgia Institute of Technology. In this house, multi-discipline sensors have been constructed for monitoring the activities of the resident.

These smart space projects didn’t organize the huge sensing data in a formal structured format. An independently developed application can’t easily interpret contexts that have no explicitly represented structure. We use the Semantic Web standards RDF and OWL to define context ontologies which provide a context model for supporting information exchange and interpret contexts. By using the Semantic Web technologies to represent context knowledge, we introduced an infrastructure for inferring higher-level contexts and provide adaptive service to the user.

8 Conclusion and Future Work

This paper presented Context-aware Service Platform, a prototype system designed to attain context-aware services in a smart space. This platform integrates several modern technologies, include ubiquitous and context-aware technologies, semantic web, AI planning, and web service. Besides, reasoning approaches for deriving new contexts and services are adopted in this platform.

Ontologies for contexts and services provide information sharing and make the platform integrating services. Contexts are represented as RDF-triple for exchanging information between agents and deducing new high-level contexts. Moreover, the service planner obtains a goal from context-aware reasoner, such that it makes the services operated adaptively.

The current design assumed that all the context resources provide consistent contexts and no conflict information to disturb the process of Context-aware Service Platform. We should consider the fault tolerance problems but allow some minor errors happened.

As the real-world environment has huge number of contexts and the required tasks are much more complex, rule engine and Graphplan should have the capability to provide solutions in reasonable time. Consequently, the concept of clock timer can be adopted to the reasoning and planning components. In order to provide a possible solution from the partial results, an anytime algorithm should be taken into account.

The scalable and distributed server architecture provides a direction to handle the complexity of real world problems. Implementing the Context-aware Server in SoC and integrating with memories, sensors, or MEMS (Micro-Electro Mechanical System) devices using SiP (System in Package) technologies, can enable scalable / distributed Context-aware applications with tiny CaS embodied in objects distributed throughout a wide spread smart space contexts.

This paper provides a simple scenario to demonstrate the idea of the context-aware service platform. However, this simple case does not show the power of automated service composition by using AI planning. Designing other scenarios that can explain and evaluate the needs of service composition is one of our future direction. Applying this platform to other Web Service composition benchmark test is another way to evaluate the performance of this platform.

References

[1] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, January 1992.

[2] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis, and M. Weiser. An overview of the PARCTAB ubiquitous computing experiment. *Personal Communications*, 2(6):28 – 43, December 1995.

[12] http://architecture.mit.edu/house_n/

[13] http://www.awarehome.gatech.edu/

[3] D. Salber, A. K. Dey, R. J. Orr, and G. D. Abowd. Designing for ubiquitous computing: A case study in context sensing. Technical Report GIT-GVU-99-29, Georgia Institute of Technology, 1999.

[4] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6):69–79, November – December 2004.

[5] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Joshi. Intelligent agents meet semantic web in a smart meeting room. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 854–861, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[6] F. Gandon and N. M. Sadeh. A semantic e-wallet to reconcile privacy and context awareness. *Lecture Notes in Computer Science: The SemanticWeb - ISWC 2003*, 2870:385–401, October 20-23 2003.

[7] F. L. Gandon and N. M. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Journal of Web Semantics*, 1(3):241–260, 2004.

[8] H. Chen, T. Finin, and A. Joshi. *Ontologies for Agents: Theory and Experiences*, chapter The SOUPA Ontology for Pervasive Computing, pages 233–258. Whitestein Series in Software Agent Technologies. Birkhäuser Basel, July 2005.

[9] H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard ontology for ubiquitous and pervasive applications. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, pages 258–267, August 2004.

[10] J. R. Hobbs and F. Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP), Special Issue on Temporal Information Processing*, 3(1):66–85, March 2004.

[11] P. V. Biron and A. Malhotra. XML schema part 2: Datatypes second edition. http://www.w3.org/ TR/ xmlschema-2/, 28 October 2004. W3C Recommendation.

[12] F. Bry, F. A. Ries, and S. Spranger. CaTTS: calendar types and constraints for web applications. In *Proceedings of the 14th international conference on World Wide Web (WWW '05)*, pages 702–711, New York, NY, USA, 2005. ACM Press.

[13] J. Ma and P. Hayes. Primitive intervals versus point-based intervals: Rivals or allies? *The Computer Journal*, 49(1):32–41, 2006.

[14] F. Reitsma. Semantic geoStuff. http://www.mindswap .org / 2004/ geo/ geoStuff.shtml, 2004. Maryland Information and Network Dynamics Lab Semantic Web Agents Project (mindswap).

[15] OpenGIS specifications. http://www.opengeospatial.org/ standards/, 2007.

[16] Opencyc selected vocabulary and upper ontology – spatial relations. http://www.cyc.com/ cycdoc/vocab/spatial-vocab.html, December 2002.

[17] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17 – 37, September 1982.

[18] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, February 1997.

[19] T.-C. Lee. Object-based information storage, search and mining system methods. United States Patent 20060136402, June 2006. Kind Code: A1.

[20] C.-y. Lin and J. Y.-j. Hsu. IPARS: Intelligent portable activity recognition system via everyday objects, human movements, and activity duration. In *Modeling Others from Observations (MOO 2006): Papers from the 2006 AAAI Workshop*, number Technical Report WS-06-13, pages 44–52, Boston, Massachusetts, USA, July 16 – 20 2006. AAAI Press.

[21] C.-w. You, Y.-C. Chen, J.-R. Chiang, P. Huang, H.-h. Chu, and S.-Y. Lau. Sensor-enhanced mobility prediction for energy-efficient localization. In *Proceedings of Third Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2006)*, volume 2, pages 565–574, Reston, VA, USA, 2006.

[22] S. S. Intille. Designing a home of the future. *IEEE Pervasive Computing*, 1(2):76–82, April 2002.

[23] G. D. Abowd, C. G. Atkeson, A. F. Bobick, I. A. Essa, B. MacIntyre, E. D. Mynatt, and T. E. Starner. Living laboratories: the future computing environments group at the georgia institute of technology. In *Proceedings of Conference on Human Factors in Computing Systems (CHI '00): extended abstracts on Human factors in computing systems*, pages 215–216, New York, NY, USA, 2000. ACM Press.