Agent-Based Context Consistency Management in Smart Space Environments*

Wan-rong Jih, Jane Yung-jen Hsu, and Han-Wen Chang

Department of Computer Science and Information Engineering
National Taiwan University
jih@agents.csie.ntu.edu.tw, yjhsu@csie.ntu.edu.tw, r96922005@ntu.edu.tw

Abstract. Context-aware systems in smart space environments must be aware of the context of their surroundings and adapt to changes in highly dynamic environments. Data management of contextual information is different from traditional approaches because the contextual information is dynamic, transient, and fallible in nature. Consequently, the capability to detect context inconsistency and maintain consistent contextual information are two key issues for context management. We propose an ontology-based model for representing, deducing, and managing consistent contextual information. In addition, we use ontology reasoning to detect and resolve context inconsistency problems, which will be described in a *Smart Alarm Clock* scenario.

1 Introduction

In the past several years mobile devices, such as smart phone, personal digital assistants (PDAs), and wireless sensors, have been increasingly popular. Moreover, many tiny, battery-powered, and wireless-enabled devices have been deployed by researchers in smart spaces with the goal of collecting contextual information about their residents. Customized information can be delivered across mobile devices, based on specific contexts (location, time, environment, *etc.*) of the user. The Aware Home[1], Place Lab[2], Smart Meeting Room[3],and vehicles[4] provide intelligent and adaptive service environments for assisting users in concentrating on their specific tasks.

Context-awareness is the essential characteristic of a smart space, and using technology to achieve context-awareness is a type of intelligent computing. Within a richly equipped and networked environment, users need not carry any devices on their person; instead, applications adapt the available resources to deliver services to users in the vicinity, as well as track the location of these users. Cyberguide[5] uses the user's locations to provide an interactive map service. In the Active Badge[6], every user wears a small infrared device, which generates a

* This work was partially supported by grants from the National Science Council, Taiwan (NSC 96-2628-E-002-173-MY3), Excellent Research Projects of National Taiwan University (97R0062-06), and Intel Corporation.

R. Kowalczyk et al. (Eds.): SOCASE 2009, LNCS 5907, pp. 41–55, 2009.

unique signal and can be used to identify the user. Xerox PARCTab[7] is a personal digital assistant that uses an infrared cellular network for communication. Bat Teleporting[8] is an ultrasound indoor location system.

In a smart space, augmented appliances, stationary computers, and mobile sensors can be used to capture raw contextual information (*e.g.* temperature, spatial data, network measurement, and environmental factors), and consequently a context-aware system needs to understand the meaning of a given context. Therefore, developing a model to represent contextual information is the first step of designing context-aware systems. Context-aware services require the high-level description about various user states and environmental situations. However, high-level context cannot be directly acquired from sensors. The capability to infer high-level contexts from existing knowledge is required in context-aware systems. Consequently, how to derive high-level contexts is the second step in designing context-aware systems. As people may move within and between environments at any time, it is increasingly important that computers adapt to context information in order to appropriately respond to the needs of users. How to deliver the right services to the right places at the right time is then the third step in designing context-aware systems. Inconsistent contexts may appear in context-aware systems as systems react to the rapid change of contextual information. Systems having inconsistent knowledge will fail to provide correct services. Therefore, a context-aware system must maintain a consistency knowledge base and react to the dynamic change of contexts, which is the fourth step in designing context-aware systems.

In this research, we leverage multi-agent and semantic web technologies that provide the means to express context and use abstract representations to derive usable context for proactively delivering context-aware service to the user. We propose an ontology-base model for supporting context management, which can provide high-level context reasoning and detect knowledge inconsistency. In addition, a *Smart Alarm Clock* scenario serves as an explanatory aid in illuminating the details of our research.

2 Background Technologies

An overview of the context models, context reasoning, and ontology is introduced in this section.

2.1 Context Representation

Context is mainly characterized by four dimensions[9]: location, identity, activity and time. Location refers to the exact position of the user. If we know a person's identity, we could easily identify related information such as birth date, social connectivity, or email addresses when the appropriate access to user-related data is given. Knowing the location of an entity, we could determine its nearby objects and users.

Many context-aware systems concentrate on location aware services. Ye *et al.*[10] use a lattice model to represent spatial structure, which can deal with

syntactic and semantic labels. This general spatial model provides both absolute and relative references for geographic positions, as well as both the containment and connection relationships. MINDSWAP Group at University of Maryland Institute for Advanced Computer Studies developed Semantic geoStuff[1] to express basic geographic features such as countries, cities, and relationships between these spatial descriptors.

The RFC 2445[2], which defines the iCalendar format for calendaring and scheduling applications, enables users to create personal event data. Google Calendar[3] is a popular web-based calendar which supports the iCalendar standard and allows users to share their own personal activities with others. These human activities can be broken down into people, time, and location. Consequently, the contents of a person's schedule can help us to derive his or her location at a given time.

2.2 Ontology

Strang and Linnhoff-popien[11] concluded that an ontology is the most expressive model. Gruber[12] defines ontology as an "explicit specification of a conceptualization". An ontology is developed to capture the conceptual understanding of a domain in a generic way and provides a semantic basis for grounding fine-grained knowledge.

COBRA-ONT[13] provides key requirements for modeling context in smart meeting applications. It defines concepts and relations of physical locations, time, people, software agents, mobile devices, and meeting events. SOUPA[14] (Standard Ontology for Ubiquitous and Pervasive Applications) uses other standard domain ontologies, such as FOAF[4] (Friend of A Friend), OpenGIS, spatial relations in OpenCyc, ISO 8601 date and time formats[5], and DAML time ontology[15]. Clearly, these ontologies provide not only rich contextual representations, but also make use of the abilities of reasoning and sharing knowledge.

2.3 Context Reasoning

Design and implementation of context reasoning can vary depending on types of contextual information that are involved. Early context-aware systems[16,17,18] embedded the understanding of the contextual information into the program code of the system. Therefore, developed applications often had rigid implementations and were difficult to maintain.

Rule-based logical inference can help to develop flexible context-aware systems by separating high-level context reasoning from low-level system behavior. However, context modeling languages are used to represent contextual information and rule languages are used to enable context reasoning. Accordingly, in

[1] http://www.mindswap.org/2004/geo/geoStuff.shtml
[2] http://tools.ietf.org/html/rfc2445
[3] http://calendar.google.com
[4] http://xmlns.com/foaf/spec/
[5] http://www.w3.org/TR/NOTE-datetime

most cases, these two types of languages have different syntaxes and semantic representations; thus it is a challenge to effectively integrate these distinctive languages in the development of context-aware systems. A mechanism to convert between contextual modeling and reasoning languages is one of solutions for this challenge. Gandon and Sadeh[19,20] propose e-Wallet which implements ontologies as context repositories and uses a rule engine Jess[21] to invoke the corresponding access control rules. The e-Wallet using RDF[6] triples to represent contextual information and OWL[7] to define context ontology. Contextual information is loaded into the e-Wallet by using a set of XSLT[8] stylesheets to translate OWL input files into Jess assertions and rules.

Ontology models can represent contextual information and specify concepts, subconcepts, relations, properties, and facts in a smart space. Moreover, ontology reasoning can use these relations to infer the facts that are not explicitly stated in the knowledge base. Ranganathan*et al.*[22] assert that ontologies make it easier to develop programs for reasoning about context. Chen[23] proposes that the OWL language can provide a uniformed solution for context representation and reasoning, knowledge sharing, and meta-language definitions. Anagnostopoulos *et al.*[24] name the Description Logic as the most useful language for expressing and reasoning contextual knowledge. The OWL DL was designed to support the existing Description Logic business segment and to provide desirable computational properties for reasoning systems. Typical ontology-based context-aware application is Smart Meeting Room that uses OWL to define the SOUPA ontology and OWL DL to support context reasoning[3],Gu *et al.*[25] propose an OWL encoded context ontology CONON in Service Orientated Context Aware Middleware (SOCAM). CONON consists of two layers of ontologies, an upper ontology that focuses on capturing general concepts and a domain specific ontology. The Smart Meeting Room and SOCAM use an OWL DL reasoning engine to check the consistency of contextual information and provide further reasoning over low-level contexts to derive high-level contexts.

3 System Architecture

Fig. 1 shows our Context-aware System Architecture, which can continuously adapt to changing contexts and proactively provide services to the user. The top part of Fig. 1 depicts a smart space environment, which is equipped with devices and applications, such as a personal calendar, weather forecast, location tracking system, contact list, shopping list, as well as raw sensing data. The system can provide both contextual information and deliver context-aware services. The Context Collection Agents obtain raw sensing data from the context sources and convert the raw context into a semantic representation. Each Context Collection Agent will deliver the sensed contextual information to the *Context Management* component after receiving sensing data.

[6] `http://www.w3.org/TR/rdf-concepts/`
[7] `http://www.w3.org/TR/owl-features/`
[8] `http://www.w3.org/TR/xslt`

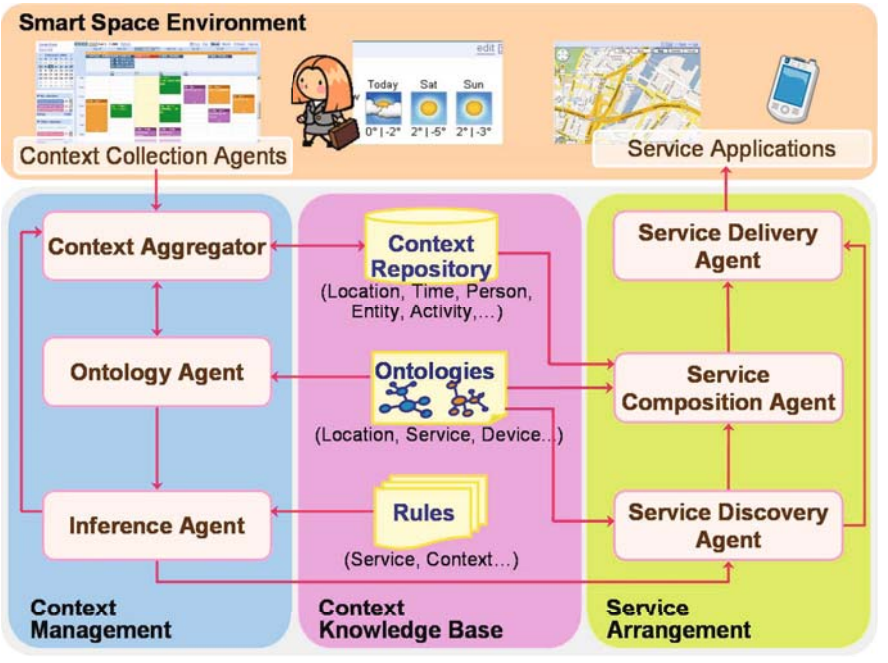

Fig. 1. A Multi-agent System Architecture

The lower part of Fig. 1 illustrates our context-aware system architecture, which consists of three components: *Context Management*, *Context Knowledge Base*, and *Service Arrangement*. The functions of *Context Management* component include monitoring the contextual information and managing the environmental resources. Contextual information, domain knowledge, and service profiles are stored in the *Context Knowledge Base*. The *Service Arrangement* component performs service discovery, composition, and execution. After assigning the specified services, the system will invoke Service Applications to provide services in the smart space environment.

4 Context-Aware Agents

Agents in Fig. 1 accomplish the functions of managing contextual information and delivering context-aware services. Functions of *Context Management* component include gathering context information from the surrounding environment, providing methods for querying and storing the contextual information, and providing the context in an ontology-based representation that facilities knowledge representation and inference. The *Service Arrangement* module checks whether any service operations match the request under the current situation. If no operation matches the request, it combines operations to match the request.

4.1 Context Aggregator

The Context Aggregator component collects contextual information from Context Collection Agents and stores the context information to the Context Repository for context inference, consistency checking, and knowledge sharing. There are two types of input context information data, the raw context and the high-level context. Raw context referring to the sensing data is directly obtained from context sources. For example, bed sensors can provide lay-on-bed sensing and a weather forecast API can provide forecasting information. The Context Aggregator component subscribes to the specified Context Collection Agent in order to retrieve the contextual information, which define in the context ontology. Consequently, Context Collection Agents are the providers of low-level contexts while the high-level contexts are derived from the Ontology Agent and Inference Agent.

4.2 Ontology Agent

The Ontology Agent loads and parses an OWL context ontology into RDF triples, which allows other agents to represent and share context in the system. The Context Aggregator component sends the current state of contexts to the Ontology Agent as soon as the subscribed context are changed. The other agents can send their queries to the Ontology Agent in order to retrieve the updated knowledge. According to the structures and relationships between contexts that define in the context ontology, the Ontology Agent performs the subsumption reasoning to deduce new contextual information. For example, it can deduce the superclasses of a specified class and decide whether one class consists of members of another, *e.g.*, a building may spatially contain a room.

4.3 Inference Agent

The Inference Agent adopts an OWL DL reasoning engine for supporting context reasoning and conflict detection. When the Inference Agent receives contextual information from the Ontology Agent, the reasoning engine will invoke rules and trigger actions that may deduce new high-level contexts and derive associated service requests. Combining the inferred contexts with the original context ontology, the Inference Agent can detect the context inconsistency. Either new high-level contexts or service requests can be derived from the Inference Agent and be delivered to the Context Aggregator or the Service Discovery Agent, respectively.

4.4 Service Discovery Agent

The Service Discovery Agent maintains the service ontology. An OWL-S[9] file defines the service ontology, which includes three essential types of knowledge about a service: the service profile, process model, and service grounding. The OWL-S service profile illustrates the preconditions required by the service and

[9] `http://www.w3.org/Submission/OWL-S/`

the expected effects that result from the execution of the service. A process model describes how services interact and functionalities are offered, which can be exploited to solve goals. The role of service grounding is to provide concrete details of message formats and protocols. According to the description in the service ontology, the Service Discovery Agent keeps the atomic services information. When the Service Discovery Agent receives a service request, it checks whether any single service satisfies the requirement under the current situation. If an atomic service can accomplish the request, the associated service grounding information will be delivered to the Service Delivery Agent.

4.5 Service Composition Agent

If a service request cannot be achieved by a single service, the Service Composition Agent will compose atomic services to fulfill the request. The service profile of a service ontology defines the service goals, preconditions, and effects. According to these semantic annotations, AI planning has been investigated for composing services. The state transition is defined by the operations, which consist of preconditions and effects. Initial states of the AI planner are combined with the current contexts and context ontology. The service request is the planning goal. Therefore, giving initial states, goals, and operations, the Service Composition Agent will derive a service execution plan, which is a sequence of operations that starts from initial states and accomplishes the given goal.

4.6 Service Delivery Agent

The service ontology defines the information for service grounding, which specifies the details of how an agent can access a service. According to the description of service grounding, the Service Delivery Agent invokes the specified Service Application with the required protocol and message contents.

5 Context Ontology Model

Context-aware applications need a unified context model that is flexible, extendible, and expressive to adapt to a variety of context features and dependency relations. The ontology models can fulfill these requirements; therefore, we deploy an ontology context model to represent contextual information in smart space environments. The ontology fills the need to share knowledge about locations, time, and activities so that context-aware applications can infer the environmental contexts and trigger services.

5.1 Context Repository

The Context Repository stores a set of consistent context, which including location, person, and activity information. Both raw and high-level context have a unique type identities and values. The associated value in the timestamp represents when the corresponding context arrived. The context ontology defines

the classes of contexts and the relationships between the instances of context objects. A RDF-triple represents a context that contains a subject, a predicate, and an object. The subject is a resource named by a URI with an optional anchor identity. The predicate is a property of the resource, while the object is the value of the property for the resource. For example, the following triple represents "Peter is sleeping".

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

Where `Peter` represents a subject, `participatesIn` is a predicate, and the activity `sleeping` is an object. We use subject and predicate as the compound key of the Context Repository. When a context has been updated, the associated timestamp will be changed accordingly.

5.2 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 define a context ontology (as depicted in Fig. 2) as a representation of common concepts about the smart space environment. Context information is collected from real-world classes (Person, Location, Sensor, Time, HomeEntity), and a conceptual class Activity. The class hierarchy represents an `is-a` relation; an arrow points from a subclass to a

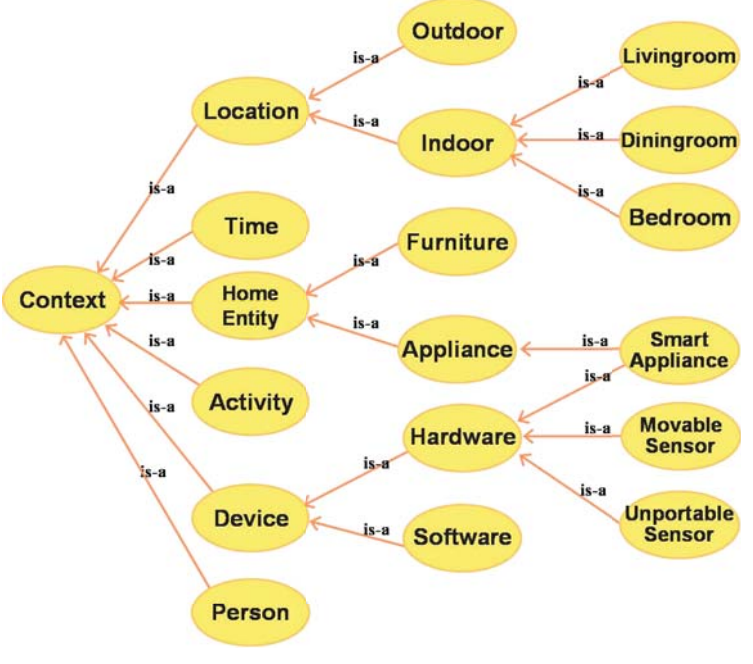

Fig. 2. A Context Ontology

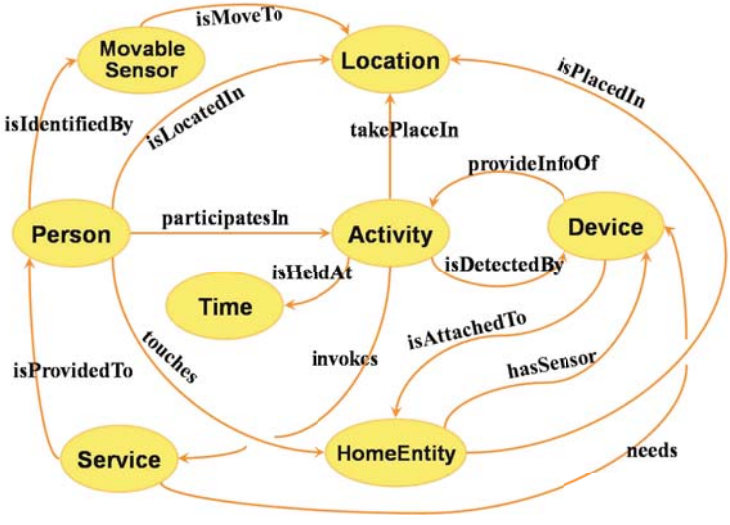

Fig. 3. Context Relationship

superclass. A class can have subclasses that represent the concepts more specific than itself. 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.

The relationship between classes is illustrated in Fig. 3. The solid arrows describes the relation between subject resources and object resources. For example, isLocatedIn describes the relation between the instances of Person and Location while the instances of Person are subject resources and instances of Location are object resources.

A service ontology defined by OWL-S describes available services that are comprised of the service profile, service model, and service grounding.

5.3 Rules

Rules of a rule-based system are simply IF-THEN statements. Context rules can be triggered to infer high-level context. According to the description of Fig. 3, a rule for detecting the location of a user is as follows:

```
[Person_Location:
  (?person isIdentifiedBy ?tag)
  (?tag isMoveTo ?room)
->
  (?person isLocatedIn ?room )
]
```

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

The rule Person_Location is an example that can deduce high-level context. The ?person is an instance of class Person, ?tag is an instance of MovableSensor, and ?room is an instance of Room, the rule Person_Location declares that if any person ?person is identified by a movable sensor ?tag and this movable sensor is moved to a room ?room, we can deduce that ?person is located in ?room.

6 Context Management and Reasoning Mechanism

In order to make our research easier to explain, we use a simple example to describe the detailed mechanisms of context management and reasoning.

> *In a smart space, we find a Smart Alarm Clock and a typical user whom we call Peter. The Smart Alarm Clock can check Peter's schedule and automatically set the wake-up alarm so as he will not miss his first appointment. If Peter does not wake up within a 5-minute period after the alarm is sounded, another alarm event is triggered, this time with increased volume. Alternatively, if Peter were to wake up earlier then the alarm time, the alarm would be disabled.*

6.1 Context Reasoning

Before implementing the *Smart Alarm Clock*, we must collect Peter's schedule to ascertain appropriate alarm times and employ reasoning as to whether Peter is awake or not. The Google Calendar Data API[10] can support the information of Peter's calendar events. The position-aware sensors, bed pressure sensors, *etc.* can help to detect whether the user is on the bed or not. For example, RFID technologies can be used to recognize and identify Peter's activities. A wireless-based indoor location tracking system determines Peter's location with room-level precision.

Fig. 4 shows the instance relationships used in detecting whether Peter is currently sleeping or not. Words within ovals represent classes and boxes represent corresponding instances of these classes. For example, bed is an instance of the Furniture class. Dashed lines indicate the connection of a class and its instance. Each solid arrow reflects the direction of object property relationship and is directed from domain to range. In addition, an inverse property can be declared by reversing the direction of a line. For example, the inverse object property of isAttachedTo is hasSensor.

[10] http://code.google.com/apis/calendar/

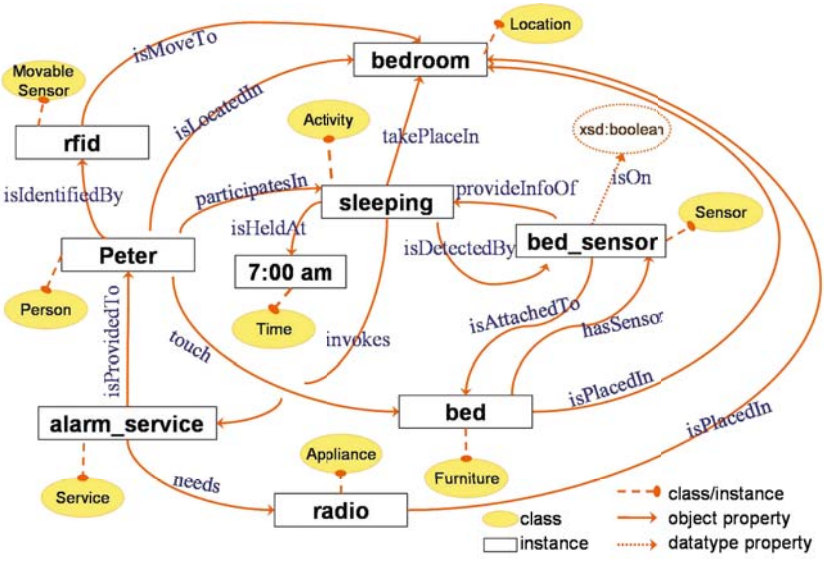

Fig. 4. A Context Snapshot

A boolean data type property isOn is associated with the Sensor class for detecting whether the value of instances are on or off. If someone is on the bed, the value of the bed sensor bed_sensor would be on, that is, the value of isOn would be `true`. Otherwise, when no one occupies the bed, the value of isOn would be `false`. When wake-up calls event has been triggered, a rule for detecting the value of the bed_sensor can be used to decide whether it is necessary to deliver the alarm_service or not.

For reasoning about high-level contexts, we apply rule-based reasoning with horn clauses into the ontology model. The rule `Person_activity` can deduce in which activity the user currently is involved. For example, in Fig. 4, when the time is up, given the location of Peter and the status of the bed sensor, the rule `Person_activity` will be triggered and can deduce whether Peter is sleeping or not. The rule `Invoke_service` can deduce what service should be delivered to the user. Given the instances of Fig. 4, the rule `Invoke_service` reflects "if Peter is sleeping, deliver smart alarm service".

```
[Person_activity:
  (?person touch ?entity)
  (?entity hasSensor ?sensor)
  (?sensor providesInfoOf ?activity)
->
  (?person participatesIn ?activity) ]
```

```
[Invoke_service:
  (?person participatesIn ?activity)
  (?activity invokes ?service)
->
  (?service isProvidedTo ?person)]
```

6.2 Context Management

Changes of environmental contexts are transient in the sense that context may appear and vanish at anytime. Algorithm 1 shows how the Context Aggregator manages the contextual information.

Algorithm 1. Maintaining Context Repository

Input: c is the new context
C: Context Repository
rdf_i: RDF-triple (s_i, p_i, o_i) of a context i
key_i: key of context i in Context Repository
for all $i \in C$ **do**
  if $isOutdated(i)$ **then**
    $delete(i)$
  end if
end for
if $\exists i \in C$ s.t. $key_i = key_c$ and $isOne2One(p_c)$ **then**
  $update(key_c, c)$
else
  $insert(key_c, c)$
end if

We use RDF-triple to represent a context while an associated compound key comprises the subject and object. When a new context arrives, the Context Aggregator uses the key of the new context to query the Context Repository. If a context exists in the Context Repository and the associated predicate represents a one-to-one relationship, the new context will replace the old one. Otherwise, the new context will be inserted into the Context Repository. Functions $update(key_c, c)$ and $insert(key_c, c)$ perform the context replacement and insertion, respectively. When a context is no longer applicable, it should be removed. function $delete(i)$ can remove the specified context from the Context Repository. We use a decay function to determine the existence of a given context. Different contexts are associated with different decay functions. This function can either be an objective function for predicating a specified activity or simply be a constant function. The function $isOutdated(i)$ applies the context decay function to decide whether the context exists or not.

6.3 Inconsistency Resolution

The Context Repository is dynamically updated to reflect the change of context. Therefore, we must ensure incorrect or outdated contexts do not exist in the

Context Repository. If a raw context is changed, some of the inferred high-level contexts may also be changed. For example, if Peter walks from the living room to the bedroom, the corresponding RDF-triple will be changed from <Peter isLocatedIn living_room> to <Peter isLocatedIn bedroom>. The Context Aggregator will update the location context of Peter because the property isLocatedIn is one-to-one relationship. If a predicate allows one-to-many relationship, the original context will be preserved.

It is a challenge that when a raw context is changed, we need to updated the associated high-level contexts. However, it is often difficult to find the corresponding high-level contexts using the context dependencies of inference rules. Updating a context may easily trigger an infinite context dependency checking loop and can lead to unpredictable situations. To address this issue, we categorize the data in the Context Repository to three types: core knowledge, raw-level contexts, and high-level contexts. The OWL ontologies define core knowledge that is static and persistent. The raw-level context is the raw sensor data that is delivered by the Context Collection Agents in Fig. 1. Using the core knowledge and raw-level contexts, rule-based reasoning can deduce high-level contexts. When a raw context has been removed, we discard the original set of high-level contexts and perform context reasoning. By design, the deduced high-level contexts are consistent with the current raw contexts and the Context Repository can maintain context consistency. This approach is while simple, but can efficiently resolve context inconsistency without recursively checking context dependencies.

7 Implementation

Our agent is deployed on JADE[11] (Java Agent DEvelopment Framework), which is a FIPA-compliant software framework for multi-agent systems, implemented in Java and comprised of several agents. Jena[12], a Java framework for building Semantic Web applications, is used for providing a programmatic environment for RDF, RDFS, and OWL. Moreover, we use an open-source OWL Description Logics (OWL DL) reasoner Pellet[13] developed by Mindswap Lab at University of Maryland, to infer high-level contexts and detect context conflicts.

8 Conclusion and Future Work

This research presents a context management mechanism in a smart space. We integrate context-aware technologies, semantic web, and logical reasoning to provide context-aware services. An ontology-based model supports a reasoning mechanism, which can deduce high-level contexts and detect context consistency.

We use a simple scenario to demonstrate the mechanism of context management. However, this simple case does not fully illustrate the power of context reasoning. Therefore, design of other scenarios that can better explain and evaluate our approach is one of our future directions.

[11] http://jade.tilab.com/
[12] http://jena.sourceforge.net/
[13] http://pellet.owldl.com/

References

1. Abowd, G.D., Atkeson, C.G., Bobick, A.F., Essa, I.A., MacIntyre, B., Mynatt, E.D., Starner, T.E.: Living laboratories: the future computing environments group at the georgia institute of technology. In: Proceedings of Conference on Human Factors in Computing Systems (CHI 2000): extended abstracts on Human factors in computing systems, pp. 215–216. ACM Press, New York (2000)
2. Intille, S.S.: Designing a home of the future. IEEE Pervasive Computing 1(2), 76–82 (2002)
3. Chen, H., Finin, T., Joshi, A., Kagal, L., Perich, F., Chakraborty, D.: Intelligent agents meet the semantic web in smart spaces. IEEE Internet Computing 8(6), 69–79 (2004)
4. Look, G., Shrobe, H.: A plan-based mission control center for autonomous vehicles. In: IUI 2004: Proceedings of the 9th international conference on Intelligent user interfaces, pp. 277–279. ACM Press, New York (2004)
5. Long, S., Aust, D., Abowd, G., Atkeson, C.: Cyberguide: prototyping context-aware mobile applications. In: Conference companion on Human factors in computing systems (CHI 1996), pp. 293–294. ACM Press, New York (1996)
6. Want, R., Hopper, A., Falcão, V., Gibbons, J.: The active badge location system. ACM Transactions on Information Systems (TOIS) 10(1), 91–102 (1992)
7. Want, R., Schilit, B.N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J.R., Weiser, M.: An overview of the PARCTAB ubiquitous computing experiment. Personal Communications 2(6), 28–43 (1995)
8. Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.: The anatomy of a context-aware application. Wireless Networks 8(2-3), 187–197 (2002)
9. Dey, A.K.: Providing architectural support for building context-aware applications. PhD thesis, Georgia Institute of Technology, Director-Gregory D. Abowd (2000)
10. Ye, J., Coyle, L., Dobson, S., Nixon, P.: A unified semantics space model. In: Hightower, J., Schiele, B., Strang, T. (eds.) LoCA 2007. LNCS, vol. 4718, pp. 103–120. Springer, Heidelberg (2007)
11. Strang, T., Linnhoff-popien, C.: A context modeling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management at The Sixth International Conference on Ubiquitous Computing (UbiComp 2004), Nottingham, England (2004)
12. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition 5(2), 199–220 (1993); Special issue: Current issues in knowledge modeling
13. Chen, H., Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments. The Knowledge Engineering Review 18(3), 197–207 (2003)
14. Chen, H., Perich, F., Finin, T., Joshi, A.: SOUPA: Standard ontology for ubiquitous and pervasive applications. In: The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004), pp. 258–267 (August 2004)
15. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Transactions on Asian Language Information Processing (TALIP) 3(1), 66–85 (2004); Special Issue on Temporal Information Processing
16. Coen, M.H.: Building brains for rooms: designing distributed software agents. In: Proceedings of the Conference on Innovative Applications of Artificial Intelligence (IAAI 1997), pp. 971–977. AAAI Press, Menlo Park (1997)

17. Wu, H., Siegel, M., Ablay, S.: Sensor fusion for context understanding. In: Proceedings of IEEE Instrumentation and Measurement Technology Conference, Anchorage, AK, USA, May 21-23 (2002)
18. Capra, L., Emmerich, W., Mascolo, C.: CARISMA: Context-aware reflective mIddleware system for mobile applications. IEEE Transactions on Software Engineering 29(10), 929–945 (2003)
19. Gandon, F.L., Sadeh, N.M.: A semantic E-wallet to reconcile privacy and context awareness. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 385–401. Springer, Heidelberg (2003)
20. Gandon, F.L., Sadeh, N.M.: Semantic web technologies to reconcile privacy and context awareness. Journal of Web Semantics 1(3), 241–260 (2004)
21. Friedman-Hill, E.: Jess in Action: Java Rule-Based Systems. Manning Publications, Greenwich (2003)
22. Ranganathan, A., Al-Muhtadi, J., Campbell, R.H.: Reasoning about uncertain contexts in pervasive computing environments. IEEE Pervasive Computing 3(2), 62–70 (2004)
23. Chen, H.: An Intelligent Broker Architecture for Pervasive Context-Aware Systems. PhD thesis, University of Maryland, Baltimore County (2004)
24. Anagnostopoulos, C.B., Tsounis, A., Hadjiefthymiades, S.: Context awareness in mobile computing environments. Wireless Personal Communications: An International Journal 42(3), 445–464 (2007)
25. Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q.: An ontology-based context model in intelligent environments. In: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, pp. 270–275 (2004)