Ubiquitous Provision of Context Aware Web Services

Irene Y.L. Chen[1], Stephen J.H. Yang [2], Jia Zhang[3]

Ching Yun University, Taiwan[1] irene@cyu.edu.tw[1]
National Central University, Taiwan[2] jhyang@csie.ncu.edu.tw[2]
Northern Illinois University[3] jiazhang@cs.niu.edu[3]

Abstract

Context-aware Web service is an interactive model between the context of service requesters and the services in Web-enabled environments. We envision that providing context-aware services is the first step toward ubiquitous Web services to enhance current Web-based e-business by finding right business partners, right business information, and right business services in the right place at the right time. The major contributions of this paper are the development of our context model and Context Aware Service Oriented Architecture (CA-SOA). We have developed a context model to formally describe and acquire contextual information pertaining to the service requesters and services. Based on the model, we have constructed CA-SOA for ubiquitous Web service discovery and access based on service and requesters' surrounding context.

Keywords: Context aware, Ubiquitous Web services, Service oriented architecture, OWL-S

1. Introduction

The objective of ubiquitous provision of Web services is to move Web and mobile services a step further from Web services at anytime anywhere to be at the right time and right place with right services. While mobile service emphasizes on users' mobility and physical location, ubiquitous service extends its emphasis to users' social perspectives and personal accessibility. We summarize the characteristics of ubiquitous Web services and their requirements as follows.

- Mobility: The continuousness of computing capability while moving from one point to another. Requirements include mobile computing on portable devices with embedded software.
- Location awareness: The capability of detecting and identifying the locations of persons and devices. Requirements include outdoor positioning and indoor positioning.
- Interoperability: The capability of interoperable operation between various standards of resource exchange and service composition and integration. Requirements include standards of content, services, and communication protocols.
- Seamlessness: The capability of providing an everlasting service session under any connection with any device. Requirements include state transition of network roaming and service migration.
- Situation awareness: The capability of detecting and identifying persons-situated scenario. Requirements include knowing what the person is doing with whom at what time and where.
- Timely adaptation: The capability of dynamically adjusting service/content depending on users' services needs. Requirements include knowing people's accessibility and preferences.
- Pervasiveness: Provide intuitive and transparent way of service/content access. Requirements include knowing what a user wants before he/she explicitly expresses it.

The characteristics of ubiquitous Web services pose significant challenges yet to be overcome before we can realize ubiquitous environment. Such characteristics and constraints need to be formalized with requirements specification in order to satisfy the demands of a ubiquitous environment.

In this paper, we present our context aware Web services, which are designed to connect, integrate, and share three dimensions of business resources: business partners, business information, and business services. In this paper, business partners refer to various roles in a typical inter organizations business process, such as suppliers, buyers, mediators, or simply any business workers. Business information means product description, products, or simply any information. Services are Web services such as Web meetings, suppliers discovery, products discovery, information sharing, communication among business partners, etc.

Context aware Web service is an interactive model between the context of service requesters and the services in Web-enabled environments [13]. We define the term "context" from two perspectives: one is from service requesters, and the other is from services. From the service requesters' perspective, context is defined as the surrounding environment affecting requesters' Web services discovery and access, such as requesters' profiles and preferences, network channels and devices the requesters are

IEEE International Conference on Services Computing (SCC'06)
0-7695-2670-5/06 $20.00 © 2006 IEEE

using to connect to the Web, etc. From the services' perspective, context is defined as the surrounding environment affecting Web services delivery and execution, such as service profiles, networks and protocols for service binding, devices and platforms for the service execution, etc.

For better illustration, let us consider a typical scenario demanding ubiquitous services. Steve is a manager, who just finished a product presentation in a trade fair and wants to call for a "PC-based Web business meeting" service and discuss some timely issues with his colleagues. The "availability" of the service must be 99% or above; otherwise, he might lose valuable business information. During the Web meeting, Steve communicates with his colleagues using various devices for exchanging multimedia-based information. After thirty minutes of "PC-based Web business meeting," Steve needs to drive back to his office for a pre-scheduled face-to-face meeting with his boss. He wants to continue the "Web business meeting" with his colleagues while he is driving. Hence, Steve needs to automatically switch to "PDA-based Web business meeting." Based on Steve's requirements, a published Web service entitled "Web business meeting" may be a qualified candidate. The service can be used by either PCs or PDAs via wireless LAN or GPRS with 99% availability for any requesters when they are on travel.

Meanwhile, there is a lot of contextual information that may affect how well Steve can be served with such a ubiquitous provision of Web services. For example, who else attend the conference? Who are Steve's colleagues and what are they doing during the meeting? What if Steve's colleagues are not available at the time? Can Steve find some other information or people with proper expertise? What kind of network channels and devices is Steve using to connect to the Web? What will the different situations, e.g., driving, affect Steve's device usage, network access and service delivery? These contextual requirements also need to be addressed to find appropriate services.

This example shows the necessity of ubiquitous service provisioning based upon ever changing contextual environments of services and service requesters. We envision that providing context-aware services is the first step toward ubiquitous Web services to enhance current Web-based e-business by finding right business partners, right business information, and right business services in the right place at the right time.

In this paper, we present our context model to formally define context description pertaining to service requesters and services. A context acquisition mechanism is designed for collecting contextual information at run time. Based on the context model, we propose our Context Aware Service Oriented Architecture (CA-SOA) for Ubiquitous Web services discovery and access based on service and service requester's surrounding context.

The remainders of this paper are organized as follows: Section 2 addresses related work. Section 3 addresses context description and context acquisition to form our context model. Section 4 presents our CA-SOA model. We conclude this paper in Section 5.

2. Related Work

One of the essential characteristics of a Service Oriented Architecture (SOA) is to help service requesters discover and locate desired services. Widely regarded as part of the foundation of Web services, UDDI [11] is a centralized service registry that defines a standard method for publishing and discovering Web service components in a service-oriented architecture. However, most of UDDI-based inquiries always return a lot of unrelated results and, even worse, requesters will receive the same reply service no matter where they are and what devices they use. More specifically, the lack of taking contextual information into account usually leads to low-recall, low-precision and irrelevant results of service discoveries.

Context refers to any information that can be used to characterize the situation of an entity, which can be a person, a place, a physical or computational object [10]. Literature has witnessed many research efforts on the development of context-awareness toolkits supporting Web services provisioning, including HP's Cooltown project [3], Dey's The Context Toolkit [2], the CB-SeC framework [8], the Gaia middleware [9], etc. These toolkits either provide functionalities to help service requesters obtain services based on their contexts or enable content adaptations according to requester's contextual information.

In order to promote context aware service provisioning, Kouadri et al. [6] proposed a formal definition of service contexts to model a service's contextual information. Lemlouma et al. [7] proposed a framework to assert metadata information of Web contents. They both used Composite Capabilities/Preferences Profiles (CC/PP) as interoperable context representation to enable communication and negotiation of device capabilities and user preferences in terms of a service invocation. Zhang et al. [15] further proposed extensions to CC/PP to enable transformation descriptions between various receiving devices. Besides, several

OWL-based context models are presented [1,4,5] to provide high-quality results of service discoveries beyond the expressive limitations of CC/PP. These researchers utilized ontology to describe contextual information including location, time, device, preference and network, etc. By combining semantic contextual information with inductive or deductive techniques, these models can perform matches against both service and receiver's context semantically.

In contrast to aforementioned related works, our approach stands out from three aspects: (1) we formalize a ontology-based context model; (2) we provide comprehensive real-time context acquisition methods; and (3) we employ a rule-based matching algorithm with truth maintenance to enhance the recall and precision of context-aware service discovery.

3. Context Model

In this section, we present our context model that is developed to formally define context description pertaining to service requesters and services. A context acquisition mechanism is also designed for collecting contextual information at run time.

The underlying foundation of our context model is to treat a context-aware Web service as a parameterized abstract machine. An abstract machine is characterized by statics (i.e., state variables) and dynamics (i.e., state functions). Contextual information is represented as functions that may change the state of the service. All functions are equipped with formally defined pre-conditions, post-conditions and invariants based upon the ontologies defined in our context model. The abstract machine structure is defined as follows:

```
MACHINE M(X,x)
ONTOLOGIES O
DEFAULTS D
SETS S; T={a,b}
PROPERTIES P
VARIABLES V
INVARIANT I
ASSERTIONS J
INITIALIZATION B
REQUIREMENTS
u1 <- O1(w1) = PRE Q1 THEN V1 END
…
un <- On(wn) = PRE Qn THEN Vn END
END
```

The abstract machine has free dimensions X (set) and x (scalar). ONTOLOGIES describes semantic meanings of machine parameters. SETS contains finite or named sets that the machine can use. DEFAULTS describes default values. PROPERTIES takes form of conjoined predicates specifying invariants involving defaults and sets. VARIABLES lists state variables, and INVARIANT describes static properties of the machine, that must be maintained with contextual settings. ASSERTIONS is deducible from PROPERTIES and INVARIANT, and exists purely to ease the proving of machine correctness. INITIALIZATION initializes state variables. REQUIREMENTS lists possible requirements of an abstract machine, with pre-conditions PRE and post-conditions THEN.

3.1 Context Description

We conceive context awareness as an interactive model between service requesters and services; thus, we need to address the context description from both parties. We have developed two types of context ontology for describing the circumstances of requesters and services respectively: requester ontology and service ontology.

The major difference between the requester ontology and service ontology is their profiles. The requester ontology contains requester profiles such as personnel profile, accessibility and preferences, calendar profile, social profile, and location profiles as follows.

```
Requester_ontology =
  {Profiles, Preferences, QoWS,
     Environment, Devices}
  Profiles =
    {Personnel, Location, Calendar, Social}
  Personnel_profile =
    {name, role, id, email, accessibility}
  Location_profile =
    {office, building, home}
  Calendar_profile =
    {owner, event , time, attendee*, location}
  Social_profile = {owner, partner+}
  QoWS =
    {Functional requirements,
     non-functional requirements}
  Environment = {Network_channel, Situation}
  Network_channel = {wired, wireless}
  Situation =
    {normal, meeting, walking, driving}
  Devices = {hardware, software}
```

The service ontology contains service profile such as input, output, pre-condition, and effect of service execution as follows.

```
Service_ontology =
  {Profile, QoWS, Environment, Devices}
Service Profile =
  {name, id, description, input, output,
   pre-condition, effect}
QoWS =
  {Functional requirement,
   non-functional requirement}
Environment =
  {Network channel, Situation}
Network channel = {wired, wireless}
Situation = {normal, meeting, walking, driving}
Devices = {hardware, software}
```

In addition to profiles, both requester ontology and service ontology contains surrounding context such as Quality of Web Services (QoWS), environment profiles, and device capability profiles. QoWS profiles contain both functional and non-functional constraints. Functional QoWS constraints can be described by network bandwidth and response time; non-functional QoWS constraints can be described by reliability, availability, and cost. Environment profiles contain network channel constraints and situated location constraints. Network channel constraints can be used to describe the types of channels such as wired or wireless; situated location constraints can be used to describe requester situated environment such as in a meeting, reading, walking, or driving. Devices profiles contain a device's hardware and software constraints. Various devices such as PDAs and mobile phones are equipped with different hardware and software constraints. Hardware constraints can be used to describe hardware capabilities of a device such as platform, CPU speed, memory size, screen size and resolution. Software constraints can be used to describe software capabilities of a device such as operating system, browser, playable media type, and resolution.

3.2 Context Acquisition

With our ontology models, both service requesters and services could define their contextual information accordingly. We define *context acquisition* as a process of obtaining the values of the properties defined using the requester ontology and service ontology. We separate the context acquisition function from the context aware services. This decoupling enables the reuse of existing context acquisition functions for various services.

Context acquisition can be conducted in three ways: form-filling, context detection, and context extraction. In the *form-filling* approach, contextual information is acquired directly from requesters' inputs. In the *context detection* approach, we utilize various sensing, recording, and positioning systems (e.g., GPS, RFID, and sensor networks) for location detection. In the *context extraction* approach, we derive contextual information from requester ontology and service ontology. The first approach form-filled can be used to construct personnel profile, preferences, calendar profile, and social profile; the term form-filled explains for itself. We thus will concentrate on the other two approaches: context detection and context extraction.

3.2.1 Context Detection Context detection is to detect and analyze contextual information such as location, environment and device profiles during the run time. We have designed a context detection environment that facilitates the process from both service side and service requester side. On the service requester side, we have utilized smart devices and sensor networks to sense and react to the requesters' surrounding environment, as we reported in [14]. On the service side, we have designed a Web service portal to accept service requests featuring a recording capability [14]. Whenever a requester logs in, our portal catches the request, analyze what kind of devices the requester is using to build the device profile, and detect what kind of network channel the requester is connecting to the Internet to build the environment profile. The situation such as whether the requester is in a meeting or is driving remains unknown at this stage; and we defer this analysis to the context extraction phase. Besides detecting the request, our portal also records and keeps the history of a service request associated with every requester who registers in this portal. Based on the historical information, we can conduct analysis about the requesting behaviours and requesting patterns that are important references for building requesters' preferences.

Our environment is equipped with a set of location detection Web services that can match all possible location tracking functionalities currently available for requesters' devices, and filter them based on requesters' actual contexts. For example, if a requester is outside of a building, then a GPS location detection service will be invoked to return her location in terms of building name or number. If the requester is inside of a building, then an indoor tracking service provided by RFID or sensor network

will be invoked to return her location in terms of room numbers. Once the location is positioned, our location detection services can further decide whether to disclose the location based on requester's privacy preference. One thing worth noting is that in our environment, we consider privacy preference in a dynamic manner and can be adjusted based on location and temporal constraints. For example, if a requester is in her office during her office hours, she is willing to disclose the room number she is currently in to her students. If she is out of town in a trip, she may only disclose her position to her colleagues and family members.

3.2.2 Context Extraction If the context detection services can not detect current context explicitly, requesters' profiles will be taken into consideration. We define *context extraction* as a process to derive contextual information from requesters' preferences and profiles. Comprehensive contextual information can be extracted combining static and dynamic approaches. The static context extraction elicits a requester's default context from her predefined preferences and personnel profile. The dynamic context extraction deduces a requester's actual context from her calendar profile and social profile.

In the static approach, except for those required properties for which requesters must specify their values such as name, id, role, email, etc, other properties defined in personnel profile and preferences have predefined default values. As a result, our system fills in the default values for the requesters if they do not explicitly specify the property values. We refer this process as *context wrapping*, and we will address this part in more detail in Section 4.

In the dynamic approach, we analyze a requester's calendar and social profiles to deduce her actual contextual information. Using our example earlier, by checking Steve's calendar profile and social profile, we can find out at 10:15am on Wednesday, he is in a meeting with his team members.

As shown below, we formally define a requester's calendar profile with properties that indicate the owner of this calendar, the privacy of using this calendar, the event title and description, the begin time and end time of the event, as well as the attendee and location of the event. Privacy is a property containing policy rules to determine whether this calendar is accessible for public or for private reference only.

```
Calendar_profile = {owner, event, time,
attendee*, location}
    owner = {name, id, privacy}
    event = {title, description}
    time = {begin(yyyy:mm:dd;hh:mm),
            end(yyyy:mm:dd;hh:mm)}
    attendee = {name, contact_info}
        location = {place, contact_info}
```

Social profile is used to find the most related business partners when a requester does not explicitly specify whom she is working with. Social profile is also useful when a requester makes her calendar profiles private but such information is needed to locate her contexts. This goal can be fulfilled by querying every partner's calendar profile to find the events associated with the target requester. A requester can have various types of business partners such as an individual or a group of team workers. The formal definition of social profile is as follows.

```
Social_profile = {owner, partner+}
    owner = {name, id, privacy}
    partner = {type, name, context_info},
    type =
        {individual | working_team | enterprise |
        community}
```

4. Context Aware Service Oriented Architecture

Based on our context model, we propose a Context Aware Service Oriented Architecture (CA-SOA). As shown in Figure 1, CA-SOA consists of three components for ubiquitous discovery and access of Web services based on surrounding contexts: an agent platform (service, broker, and request agents), a service repository (service profile and service ontology), and a semantic matchmaker (capability matchmaker, context matchmaker, context reasoner, and service planner).

We identify three types of agents in this CA-SOA: service agents, broker agents, and request agents. The agents have been implemented to enhance the context-oriented service description, publication, registration, discovery, and access. As shown in Figure 1, a service agent is designed to help service providers formally describe and wrap the service with contextual descriptions. The service agent then sends the service with the contextual descriptions to the broker agent. A request agent is designed to help service requesters formally describe their service requests and wrap the requests with requesters' surrounding contexts. The request agent then sends the requests with requester's context descriptions to the broker agent. A broker agent takes the service publishing requests from the service agent and saves

the service descriptions and service contextual descriptions into the service profiles and the service ontology, respectively. The broker agent also takes the requests from the request agent and initiates context aware semantic matchmaking.

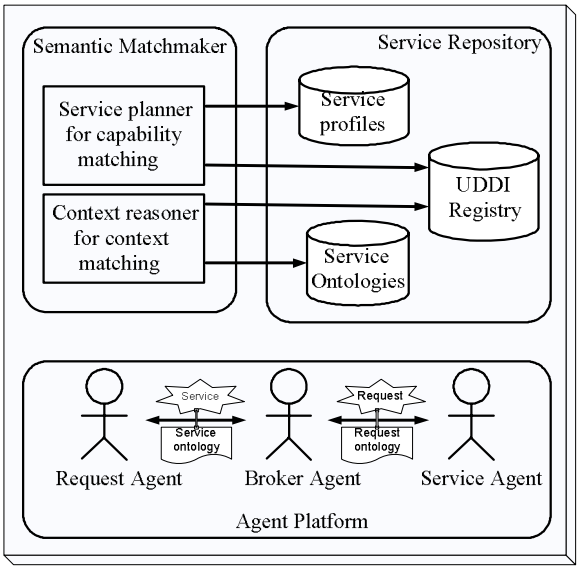

Figure 1. Context aware SOA

Context aware semantic matchmaking consists of two stepwise phases: capability matching and context matching. As shown in Figure 1, our service repository is designed to encompass a general UDDI Registry associated with service profiles and service ontologies. If the required services are found by the capability matchmaker in the UDDI Registry, the semantic matchmaker will proceed the context matching. The semantic matchmaker consists of a context reasoner and a service planner. The context reasoner decomposes the service request, based on requesters' ontology sent along with the service request by the request agent, into a set of sub-requests. The service planner performs a context matching process in order to schedule an integrated composite service based on the decomposed request.

4.1 Context-oriented request description

Generally speaking, requesters input the keywords of their requests to indicate the titles of services they want. A sophisticated query interface may ask the requesters to give the inputs and outputs, or even pre-conditions and effects of the requested services. Nevertheless, none of the existing request query considers requesters' surrounding contexts, which fact may reduce the precision of search results. Moreover, if the query user interface asks requesters to explicitly tag the contextual information, it will cause the requesters tremendous overhead and discourage them to use context-aware Web services. We thus utilize the request agent to automatically wrap the contextual information as defined in requester ontology. The steps of wrapping contextual information and transforming them into requester ontology are as follows:

1. The request agent provides a query interface for service requesters to input the keywords of their query.
2. The request agent instantiates an instance of requesters' ontology.
3. The request agent enacts a context acquisition service that consists of context detection and context extraction.
4. The request agent provides a context wrapper service to tag the properties defined in the requesters' personal ontology.
5. The request agent provides a request wrapper service to package the request query along with the tagged requester ontology.
6. The request agent sends the request package to the broker agent.

Using our example, Steve, as a requester, inputs a service request for "Web business meeting" through the query user interface. Steve is a manager who just finished a product presentation in a trade fair and wants to call for a "Web business meeting" and discuss with his colleagues. The "availability" of the service requested by Steve needs to be beyond 99% or he could lose valuable business information during the meeting. The result of contextual information wrapped by the request agent is shown as follows. Due to the page limit, we only show part of the transformed OWL-S code with contextual description of the request.

```
<?xml version="1.0"?>
<rdf:RDF>
   <owl:DatatypeProperty
rdf:ID="Social_Owner_Name">
      <rdfs:domain
rdf:resource="#Social_Owner"/>
      Steve
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty
rdf:ID="Owner_Privacy">
      <rdfs:domain
rdf:resource="#Calendar_Owner"/>
```

```
        Public
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty
rdf:ID="Default_Value_of_Device">
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>
       NB, PDA
       <rdfs:domain rdf:resource="#Preferences"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty rdf:ID="Bandwidth">
       <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>
       10Mbps
       <rdfs:domain
rdf:resource="#Functional_Constraints"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Availability">
       <rdfs:domain
rdf:resource="#Non-functional_Constraints"/>
       99%
     <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSche
ma#string"/>
    </owl:DatatypeProperty>
    <Situation rdf:ID="Meeting"/> Audio Off
    <Situation rdf:ID="Driving"/> Video Off
</rdf:RDF>
```

4.2 Context-oriented service description and publication

To avoid overhead, we utilize the service agent to automatically wrap the contextual information as defined in service ontology from the service side. The steps of wrapping contextual information and transforming them into service ontology are as follows:

1. The service agent provides a parser user interface for service providers to input their service descriptions in OWL-S.
2. The service agent instantiates an instance of service ontology.
3. The service agent enacts a context acquisition service that consists of context detection and context extraction.
4. The service agent provides a context wrapper service to tag the properties defined in the service ontology.
5. The service agent provides a publishing request wrapper service to package the service along with the tagged service ontology for publication.
6. The service agent sends the publishing request package to the broker agent.

For example, a service provider provides a business service entitled “Web business meeting”, which can be used by either PC or PDA devices via wireless LAN or GPRS with 99% availability for any requesters who are out of office. Part of the transformed OWL-S code with contextual description of service is shown as follows.

```
   <owl:DatatypeProperty rdf:ID="Availability">
      <rdfs:domain
rdf:resource="#Non-functional_Constraints"/>
       95%
       <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema
#string"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty rdf:ID="Platform">
      <rdfs:domain rdf:resource="#Software"/>
      NB, PDA
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema
#string"/>
   </owl:DatatypeProperty>
   <owl:DatatypeProperty rdf:ID="OS">
       <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema
#string"/>
       Windows NT, Windows XP
       <rdfs:domain rdf:resource="#Software"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Browsers">
       <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema
#string"/>
       Microsoft IE, Mozilla Firefox, Netscape
       <rdfs:domain rdf:resource="#Software"/>
   </owl:DatatypeProperty>
```

4.3 Context-oriented service discovery

The interactive model is enacted by a semantic matchmaking, which can perform semantic reasoning for context-oriented service discovery and access based on the two context ontologies. Our context-oriented service discovery consists of two stepwise phases: capability matching and context matching. In capability matching, we utilize the DAML-S/UDDI methodology that uses IN and OUT

to find a set of Web services with capabilities matching the requests. In context matching, we refine our query by conduct context matching. We have utilized broker agents to handle the service publishing requests sent by service agents and to handle the service requests sent by request agents.

While handling the service publication and registration, the broker agents help the service repository maintain a copy of service profiles published by service agents. Since the service agents wrap the published services with contextual description, the broker agents will help the semantic matchmaker keep the associated contextual descriptions into the service ontology. The service agents keep the original service profiles and service ontology; and the broker agents keep the copies. The broker agents also provide a caching mechanism to improve search performance if the same services are requested by multiple requesters. Whenever the service profiles and service ontology are updated, the service agents need to inform the broker agents. While handling the service requests, the broker agents take the service requests as input and forward this request to semantic matchmaker for service discovery. The stepwise context-oriented Web service discovery is described as follows.

1. The request agent sends the request wrapped with requester's context to the broker agent.
2. The broker agent forwards the request to service planner and performs capability match.
3. If there is no request matched, the context reasoner will decompose the request into sub-requests based on requester's context, and repeat capability match in Step 2 for each sub-request.
4. The service planner returns matched services to the broker agent.
5. The broker agent forwards the request to context reasoner to perform context matching for finer granularity.
6. The broker agent relays the information to the request agent.

Our capability matching strategy is based on inputs and outputs to identify the similarities between requests and service's advertisements. There are three matching degrees:

1. Exact match: Both outputs and inputs of desired service are equivalent of request's, i.e., $IN_{Ad} = IN_{Req}$ and $OUT_{Ad} = OUT_{Req}$.
2. Plug-in match: The outputs of a service are more specific than request's, or inputs of a service need less information than request provides, i.e., $OUT_{Ad} \supset OUT_{Req}$ or $IN_{Req} \supset IN_{Ad.}$
3. Subsumed match: The outputs of a service can only provide partial information needed by the request, or inputs of a service are more specific than the request provides, i.e., $OUT_{Req} \supset OUT_{Ad}$ or $IN_{Ad} \supset IN_{Req.}$

For context matching, we have implemented the requester ontology and service ontology with rule-base system using the Java Expert System Shell (JESS) and have developed inference rules for context matching with the consideration of truth maintenance, as we reported in [12]. The reason of performing truth maintenance is that the contexts regarding requesters and services change constantly; thus, the newly detected contextual information must be consistent with existing contextual profiles and ontology. Otherwise, we need to do adjustment to preserve the correctness of rule base. According to the scenario we addressed in the Introduction section, with the rule base system, our system can answer typical queries such as: "*What is the main theme of the meeting Steve attends?" "Where is the meeting held?" "When and how long will the meeting be held?"* etc.

5. Conclusions and Future Research

In this paper, we have presented our context model to formally define context description pertaining to service requesters and services. A context acquisition mechanism is also designed for collecting contextual information at run time. Based on the context model, we have presented our Context Aware Service Oriented Architecture (CA-SOA) for ubiquitous Web services discovery and access based on services and service requesters' surrounding contexts.

In this research, we utilize OWL-S as a vehicle to carry contextual information. Although OWL-S is a known tool from the semantic Web society tailored for Web services descriptions, our context model and CA-SOA are not limited to OWL-S. For instance, we can use Web Services Agreement Specification (WS-Agreement) from Global Grid Forum or Web Service Level Agreement (WSLA) from IBM to carry the knowledge.

In our future research, we will continue to enhance our CA-SOC in the categories of rule base truth maintenance, request decomposition, service planning, and service verification. We also plan to conduct more experiments to examine performance metrics including precision and recall ratio of services discovery, efficiency of service composition, effectiveness of service verification, and reliability of service execution.

Acknowledgment

This work is supported by NSC, Taiwan under grants 94-2524-S-008-001, 93-2524-S-009-004-EC3.

References

[1] T. Broens, S. Pokraev, M.v. Sinderen, J. Koolwaaij, and P.D. Costa, "Context-aware, Ontology-based, Service Discovery," *Proc. of 2nd European Symposium on Ambient Intelligence (EUSAI 2004)*, Nov. 8-10, 2004, Eindhoven, The Netherlands, pp. 72-83.

[2] A.K. Dey, "The Context Toolkit," *http://www.cs.berkeley.edu/~dey/context.html*

[3] Hewlet Packard, "Cooltown Project," *http://www.cooltown.com/cooltown/index.asp*

[4] M. Khedr and A. Karmouch, "Negotiating Context Information in Context-Aware Systems," *IEEE Intelligent Systems*, Vol. 19, No. 6, pp. 21-29. 2004.

[5] M. Khedr "A Semantic-Based, Context-Aware Approach for Service-Oriented Infrastructures," *Proc. of 2nd IFIP International Conference on Wireless and Optical Communications Networks (WOCN 2005)*, Mar. 6-8, 2005, Dubai, United Arab Emirates UAE, pp. 584-588.

[6] S.K. Mostefaoui and B. Hirsbrunner, "Context Aware Service Provisioning," *Proc. of the IEEE International Conference on Pervasive Services (ICPS)*, Jul. 19-23, 2004, Beirut, Lebanon, pp.71-80, 2004.

[7] T. Lemlouma and N. Layaida, "The Negotiation of Multimedia Content Services in Heterogeneous Environments," *Proc. of The 8th International Conference on Multimedia Modeling(MMM 2001)*, Amsterdam, The Netherlands, Nov. 5-7, 2001, pp. 187-206.

[8] S.K. Mostefaoui, A. Tafat-Bouzid, and B. Hirsbrunner, "Using Context Information for Service Discovery and Composition," *Proc. of 5th International Conference on Information Integration and Web-based Applications and Services (iiWAS 2003)*, Jakarta, Indonesia, Sep. 15-17, 2003, pp. 129-138.

[9] M. Roman, C.K. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing*, Vol. 1, No. 4, Oct.-Dec. 2002, pp.74-83.

[10] B.N. Schilit, N.I. Adams and R. Want, "Context-Aware Computing Applications," *Proc. of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, Dec. 1994, pp. 85-90.

[11] UDDI.org, "UDDI Specification Version 3," *http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3*.

[12] S.J.H. Yang, J.J.P. Tsai, and C.C. Chen, "Fuzzy Rule Base Systems Verification Using High Level Petri Nets," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 2, Mar.-Apr. 2003, pp. 457-473.

[13] S.J.H. Yang, I. Chen, and N. Shao, "Ontological Enabled Annotations and Knowledge Management for Collaborative Learning in Virtual Learning Community," *Journal of Educational Technology and Society*, Vol. 7, No.4, pp. 70-81. 2004.

[14] S.J.H. Yang, "Context Aware Ubiquitous Learning Environments for Peer-to-Peer Collaborative Learning," *Journal of Educational Technology and Society*, Vol. 9, No. 1, Jan, 2006.

[15] J. Zhang, L.-J. Zhang, F. Quek, and J.-Y. Chung, "A Service-Oriented Multimedia Componentization Model", *International Journal of Web Services Research* (JWSR), Vol. 2, No. 1, Jan-Mar. 2005, pp. 54-76.