This article was published in an Elsevier journal. The attached copy is furnished to the author for non-commercial research and education use, including for instruction at the author's institution, sharing with colleagues and providing to institution administration.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

<http://www.elsevier.com/copyright>

ELSEVIER

Available online at www.sciencedirect.com
ScienceDirect

Expert Systems with Applications 34 (2008) 2254–2266

Expert Systems
with Applicationswww.elsevier.com/locate/eswa

A JESS-enabled context elicitation system for providing context-aware Web services

Stephen J.H. Yang ^{a,*}, Jia Zhang ^{b,1}, Irene Y.L. Chen ^{c,2}^a *Department of Computer Science and Information Engineering, National Central University, No. 300, Zhongda Road, Zhongli City, Taoyuan County 32001, Taiwan*^b *Department of Computer Science, Northern Illinois University, 1425 W. Lincoln Hwy., DeKalb, IL 60115-2825, USA*^c *Department of Information Management, Ching Yun University, No. 229, Chien-Hsin Road, Zhongli 320, Taiwan*

Abstract

Providing context-aware Web services is an adaptive process of delivering contextually matched Web services to meet service requesters' needs. We define the term "context" from two perspectives: one from service requesters; and the other from Web services. From the former perspective, context is defined as the surrounding environment affecting requesters' services discovery and access, such as requesters' preferences, locations, activities, and accessible network and devices. From the latter perspective, context is defined as the surrounding environment affecting Web services delivery and execution, such as networks and protocols for service binding, devices and platforms for service execution, and so on. This paper presents a Java Expert System Shell (JESS)-enabled context elicitation system featuring an ontology-based context model that formally describes and acquires contextual information pertaining to service requesters and Web services. Based on the context elicitation system, we present a context-aware services-oriented architecture for providing context-aware Web service request, publication, and discovery. Implementation details of the context elicitation system and the evaluation results of context-aware service provision are also reported.

© 2007 Elsevier Ltd. All rights reserved.

Keywords: Context; JESS; Ontology; Service oriented architecture; Web services

1. Introduction

When a Web service is developed, it typically has its most suitable context (e.g., platforms and devices) for the best execution performance. For example, a Web service may be developed oriented to desktop browsers instead of PDA browsers. As people are constantly on the move in nowadays heterogeneous working environment, resources (e.g., computational devices and communication network coverage) are more frequently prone to change due to physical location changes. Therefore, in the process of a service

consumption, a service requester may need to smoothly switch to different services adapting to the ever-changing environment (i.e., context), especially when a gross mismatch between resources requirements and supplies occurs (Satyanarayanan, 2004). Providing context-aware Web services is an adaptive process of delivering contextually matched Web services to service requesters. In our research, context-aware Web services emphasize not only on people's mobility and physical location, but also on people's contextual information about their temporary situations (e.g., whether they are in a meeting). We envision that providing context-aware Web services is the first step toward ubiquitous Web services to enhance current e-services by finding right partners, right information, and right services in the right place at the right time.

In this paper, we present an ontology-based context model to formally define contextual descriptions pertaining

* Corresponding author. Tel.: +886 3 4227151x35308.

E-mail addresses: jhyang@csie.ncu.edu.tw (S.J.H. Yang), jiazhang@cs.niu.edu (J. Zhang), irene@cyu.edu.tw (I.Y.L. Chen).¹ Tel.: +1 312 718 2468.² Tel.: +886 3 4581196x6310.

to both service requesters and Web services. For context elicitation, we implement the ontology-based context model into a rule-based system using Java Expert System Shell (JESS) (Ernest, 2003), which uses an algorithm called *Rete* to match rules. The *Rete* algorithm makes JESS much faster than a simple set of cascading *if-then* statements adopted in conventional programming languages. A JESS-enabled context elicitation system has been developed with the consideration of truth maintenance, due to potentially constant changes of requesters' and services' contextual information. Newly detected contextual information must be consistent with the existing ones; thus, a rule base needs to perform truth maintenance to preserve its correctness. Based on the context elicitation system, we present a context-aware services-oriented architecture (CA-SOA) for providing context-aware services request, publication, and discovery.

To facilitate presentation, let us consider a Web meeting scenario that demanding context-aware Web services as an illustrative example throughout this paper. Steve is a product manager on site of a trade fair. He is hosting a Web meeting with his remote colleagues. Since he needs to transmit some real time materials, the "availability" of the Web meeting services must be 99% or above. Thirty minutes after the Web meeting starts, Steve needs to drive to a client for another pre-scheduled visiting. He wants to continue this ongoing Web meeting in a voice-based format while he is driving. This scenario obviously requires context-aware services, which can automatically detect Steve's contextual changes. For example, a location tracking service needs to detect Steve's position change while he is moving. Meanwhile, an adaptation service needs to adjust Web meeting formats to match Steve's contextual changes due to location, activity, network and device changes. Format changes include from indoor meeting to outdoor meeting, from fix-place meeting to driving-based meeting, from wired- to wireless-based meeting, from notebook- to PDA-based meeting, and from video-based to voice-based meeting.

The remainder of this paper is organized as follows. Section 2 addresses related research. Section 3 presents JESS-enabled context elicitation system featuring a context model for context description and contextual information collection. Section 4 presents our CA-SOA for context-aware Web service provision. Section 5 demonstrates the system along with experiment results and discussions. We conclude this paper in Section 6.

2. Related research

2.1. Web services and service oriented architecture

The emerged Web services technology provides a flexible and efficient approach to reuse existing Web-based applications and services. Web service technology concentrates not only on interoperability but also on how to describe, publish, locate and invoke Web services. A number of stan-

dards and specifications created from industry and academia have contributed to the development of Web services, such as WSDL (<http://www.w3.org/TR/WSDL>) and UDDI (<http://www.uddi.org/>). Service providers can describe services in WSDL to specify what a service can do and how to invoke the service. A UDDI registry creates a standard interoperable platform that enables businesses and applications to quickly, easily, and dynamically locate Web services over the Internet. However, the above Web service infrastructure can only behave well in syntactical level and is lack of semantic information to support inferential capability. This motivates Web service a step further to semantic Web (Berners-Lee, Hendler, & Lassila, 2001).

Semantic Web description languages, such as DAML + OIL (<http://www.daml.org/>) and Web Ontology Language (OWL, <http://www.w3.org/TR/owl-features/>), provide predicate-like and description logic-based annotation of Web services to encourage inferential procedures on annotated Web services. The combination of semantic Web's expressive power and Web service infrastructure results in the concept of semantic Web services. As the first step toward semantic Web services, OWL-S (<http://www.daml.org/services/owl-s/1.1/overview/>) is a specific application for Web services by applying OWL on a service's capability representation. It supports Web service providers with a core set of markup language constructs for describing properties and capabilities of their Web services in an unambiguous, machine interpretable way. We choose OWL-S as the implementation language of our context-aware Web services.

Service oriented architecture (SOA) is a conceptual model that inter-relates services through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner independent of hardware platforms, operating systems, and programming languages in which the services are implemented. This allows services built on a variety of such systems to interact with each other in a uniform and universal manner. Three fundamental roles exist in SOA – service providers, service requesters, and service matchmaker. The service providers publish services to the matchmakers. The service requesters send requests to the matchmakers when they need services. The matchmakers perform service matching to find available services and bind the services from the providers to the requesters. While Web services are moving toward semantic Web, the matchmakers in SOA are also favor semantic-based service matchmaking.

LARKS is a semantic matchmaker that performs both syntactic and semantic matching (Sycara, Klusch, & Lu, 2002). It can identify different degrees of partial matching based on five filters: context matching, profile comparison, similarity matching, signature matching, and constraint matching. Requesters can select any desired combination of these filters based upon different concerns, such as efficiency and computation cost. DAML-S/UDDI is another semantic matchmaker that expands UDDI to provide semantic matching (Sycara, Paolucci, Ankolekar, & Srinivasan,

2003). It can perform inferences based upon subsumption hierarchy leading to the recognition of semantic matches regardless of their syntactical differences. DAML-S/UDDI adopts a flexible matching strategy based on inputs and outputs to identify similarities between requests and service's advertisements. However, although DAML-S/UDDI matchmaker expands functionality of UDDI registry to enable capability match, it does not take requesters' contextual information into account. Requesters may receive some invalid services, and they need to manually verify all matched services. In contrast, our context-aware SOA can provide automatic and better contextually matched services to meet requesters' context changes.

2.2. Context and context-aware applications

Context can be interpreted differently from various perspectives. In this research, we will address context from the aspect of mobile and pervasive computing. Schilit, Adams, and Want (1994) point out that one of the major challenges of mobile computing is how to exploit the changing environment with a new class of applications that are aware of context. They believe context encompasses more than just people's location; instead, context should also include environment's lighting, noise level, network communication, and even people's social situation. They also believe that context-aware applications should be able to adapt to changing environment according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as changes to such things over time.

Dey and Abowd (1999) define context as “*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves*”. They point out that certain types of context that are, in practice, more important than others, such as identity, location, activity, and time. Given a person's identity, one can acquire many pieces of related information, such as the person's contact information and preferences. Given a person's location information, one can determine what other resources or people are nearby and what activity is occurring close to the person. Given a timestamp or period of time, one can find out what events are taking place or what activities have occurred at the same time. Based on the four primary context types, Dey and Abowd (1999) further define context-aware system as “*if it uses context to provide relevant information and/or services to the user, where relevance depends on the user's task*”. They also refer context-aware applications as applications that automatically provide information and take actions according to the user's present context as detected by sensors. In addition, context-aware applications look at *who's*, *where's*, *when's* and *what's* (that is, what a person is doing), and use this information to determine *why* the situation is occurring.

In addition to the aforementioned definitions, there are many comprehensive survey articles and special issues on

journals covering context and context-aware applications, readers can refer to articles such as Dey and Abowd's (1999) “Toward a Better Understanding of Context and Context-awareness”, Korkea-aho's (2000) “Context-Aware Applications Survey”, and Chen and Kotz's (2000) “A Survey of Context-Aware Mobile Computing Research”. In addition, two special issues on journals can be found on “Context-aware computing” on *IEEE Pervasive Computing* (Abowd, Ebling, Hung, Lei, & Gellersen, 2002) and *Human-Computer Interaction* (Moran & Dourish, 2001), respectively.

There are also many research efforts on the development of toolkits supporting context-aware services provision, including HP's Cooltown project (<http://www.hpl.hp.com/archive/cooltown/>), Dey's Context Toolkit (<http://www.cs.cmu.edu/~anind/context.html>), Mostefaoui, Tafat-Bouazid, and Hirsbrunner's (2003) CB-SeC framework, and Roman et al.'s (2002) Gaia middleware. These toolkits either provide functionalities to help service requesters find services based on their contexts or enable content adaptations according to requester's contextual information.

In order to promote context-aware service provision, Kwon et al. have applied context-aware multi-agent approach for personal reminder and negotiation support systems (Kwon, Choib, & Park, 2005; Kwon, Shin, & Kim, 2006). They also utilize context-aware and ubiquitous computing technologies on decision making applications (Kwon, Yoo, & Suh, 2005; Kwon, 2006). Mostefaoui and Hirsbrunner (2004) propose a formal definition of service contexts to model a service's contextual information. Lemlouma and Layaida (2001) propose a framework to assert metadata information of Web contents. They both use 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, Zhang, Quek, and Chung (2005) further propose extensions to CC/PP to enable transformation descriptions between various receiving devices. Besides, several OWL-based context models are presented (Broens, Pokraev, Sinderen, Koolwaaij, & Costa, 2004; Khedr & Karmouch, 2004; Khedr, 2005) to provide high-quality results of service discoveries beyond the expressive limitations of CC/PP. These researchers all utilize ontology to describe contextual information including location, time, device, preference, and network.

However, existing context definitions either focus on service requesters or on services alone. We argue that effective and efficient context-aware service provision needs to consider from both sides. In contrast to aforementioned related work, our approach stands out from three aspects: First, we formalize an ontology-based context model considering both service requesters and services. Second, we provide a JESS-enabled context elicitation with tailored contextual information collection. Third, we employ semantic matchmaking to enhance the recall and precision of context-aware service provision.

3. JESS-enabled context elicitation system

Context-aware Web service provision can be implemented by an event-driven reactive system. If an event is triggered when a context changes (e.g., location change), then a corresponding reaction to the changed contextual information is provided (e.g., providing a contextually matched Web services). Thus, we have designed our context-aware service provision as an event-driven rule base for responding to people's context changes, including the changes of condition, location, activity, device, network, and time. Since the construction of a rule base is incremental, new facts will be asserted when new contextual information is acquired, and new context elicitation rules will be inserted when new context types are considered. In contrast, facts and rules will be removed when they are no longer being true when people's contexts change. Since any insertion and deletion of facts and rules to an existing rule base may result in inconsistency or conflict, truth maintenance is needed to ensure the consistency of a rule base. We have developed a truth maintenance mechanism for verifying rule base concerning redundancy, inconsistency, incompleteness, and circularity to ensure the incrementally constructed rule base is sound and complete. For readers who are interested in this subject, please refer to our previous research of truth maintenance (Yang, Tsai, & Chen, 2003).

3.1. Ontology-based context model

Based on the four primary context types (i.e., *identity*, *location*, *activity*, and *time*) defined by Dey and Abowd (1999), we have developed an ontology-based context model, which is illustrated in Fig. 1. Two types of context ontologies – requester ontology and service ontology are developed for describing requesters and services, respectively. The major differences between the two ontologies are the definition profiles. The requester ontology contains profiles describing the four primary context types. A personal profile is used to describe requester's *identity* and preference; a location profile is used to describe *location*; a calendar profile is used to describe what, when, and where an *activity* occurred as well as whom are with the requester. The service ontology is OWL-S based, which contains service profiles describing the input, output, precondition, and effect of service execution. Please be noted that the attributes listed in Fig. 1 is selected for ease of illustration; the actual contextual information will be much more complicated in the real world.

Other than the profiles, both requester ontology and service ontology define Quality of Services (QoS) and environment profiles. A QoS profile contains functional and non-functional constraints. Functional QoS constraints can be described by response time and throughput; non-functional QoS constraints can be described by reliability, availability, and cost. An environment profile is used to describe computing and communication environment that contains

network constraints, situated condition, and device constraints. Network constraints can be used to describe the types of communication protocols, such as GPRS, 3G, VoIP, and WiFi. Situated condition can be used to describe requesters' situation, such as in a meeting or driving condition. Device constraints are used to describe device's hardware and software configurations, which can be defined by Composite Capabilities/Preferences Profile (CC/PP) and User Agent Profile (UAProf).

3.2. Rule-based context elicitation with JESS

Context elicitation is a process of deriving the values defined by the pairs of (attribute, value) in the context ontology. As shown in Fig. 2, a JESS-enabled context elicitation system is designed comprising three phases – form-filling, context detection, and context extraction. In the form-filling phase, contextual information is acquired directly from requesters' inputs. In the context detection phase, we utilize various sensing and positioning techniques (e.g., GPS, RFID, and sensor networks) for location detection. In the context extraction phase, we utilize JESS rules to derive contextual information from both requester ontology and service ontology.

We use form-filling to construct requester ontology based upon requesters' inputs, e.g., personal, environment, calendar, and social profiles. As shown in Fig. 2, we implemented an ontology editor to prompt, receive, and organize input information. We also designed a portal service to keep historical contextual information that can be used to predict people's requesting behaviours and patterns. In other words, the portal helps to construct requesters' context automatically. In the form-filling phase, obtained requester-inputted contextual information is stored in the portal.

The goal of the context detection phase is to collect requesters' dynamic contextual information. We have designed and implemented two Web services to facilitate the process: requester context detection service and service context detection service. For example, we utilize smart devices and sensor networks to detect requesters' surrounding environments. When requesters log in, our engine detects and analyzes the kind of devices and network they are using so as to build environment profile. The information retrieved is then stored in the portal too.

We equipped our environment with a set of location detection services, which can be automatically invoked based on requesters' actual contexts. For example, if a requester, Steve, is outside of a building, a GPS location detection service will be invoked to return his location in terms of the building name. If Steve is inside of a building, an indoor tracking service provided by RFID will be invoked to return his location in terms of the room number. Once the location is positioned, our location detection services will decide whether to disclose the location based on requester's privacy preferences. One thing worth noting is that in our design, we consider privacy preferences in a

Fig. 1. Context ontology including requester ontology and service ontology.

dynamic manner and can be adjusted based on location and temporal constraints. For example, if Steve is in his office during his office hours, he is willing to disclose his office number to his colleagues. If he is out of town in a trip, however, he may only disclose his position to his family members.

If form-filling and context detection can not obtain requesters' contextual information, then we need to derive such information by context extraction. We define context extraction as a process of deriving contextual information from requesters' context ontology. Comprehensive contextual information can be extracted combining static and

dynamic approaches. The static context extraction elicits a requester's default context from his/her predefined preferences and personal profiles. The dynamic context extraction deduces a requester's actual context from his/her calendar profile and social profile.

In the static approach, except for those required attributes that requesters must specify, other attributes defined in personal profile and preferences have predefined default values. As a result, our approach is to fill in the default values on behalf of requesters if they do not explicitly specify attributes' values. We refer this process as *context wrapping*, and we will address it in detail in Section 4.

Fig. 2. JESS-enabled context elicitation system.

In the dynamic approach, we analyze a requester's calendar and social profiles to deduce his/her actual contextual information. As indicated in Fig. 1, we formally define a requester's calendar profile with attributes indicating requester's id, activity, time of the activity, attendees and location of the activity, and privacy issue. Privacy is a property containing policy rules to determine whether this calendar is accessible for public or for private reference only. A social profile describes a person's related partners. It is useful when someone tries to locate a person by querying his/her partners' calendar profiles.

3.3. Implementation of JESS rule base

Using our earlier example, by checking Steve's calendar profile and social profile, we can infer that for a given time, he is in a Web meeting with his colleagues in distance. Location detection services can locate Steve's position and read his calendar and social profiles to find out where he is at a time point, what he is doing, and the person with whom he is currently working. Therefore, context-aware Web services can provide services to better match Steve's contextual changes. Besides deriving contextual information, we can also help requesters to raise query in order to derive more contextual information such as finding the calendar owner

during a certain period of time; finding what activity occurred during a certain period of time; finding all the attendees who are involved in an activity; finding the location where an activity occurred; finding someone based on whom the calendar owner is currently with; finding someone based on the calendar owner's location; and finding someone based on the activity the calendar owner is involved.

The aforementioned scenarios can be implemented by JESS queries as follows. Let a given time is "2006:09:06:10:00", which denotes 2006, September the 6th, 10:00 o'clock in the morning. With the JESS rules, our system can answer typical queries such as:

Query#	Query
Q1	What is the conference Steve attends?
Q2	Where is the conference held?
Q3	Who else attend this conference?
Q4	Who are Steve's colleagues and what are they doing during the conference?
Q5	What kind of network and device is Steve using to connect to the Web?
Q6	What is Steve's current situation?
Q7	What are Steve's default and required QoWS requirements?

The corresponding JESS rules for answering above queries are as follows:

Web service request, publication, and discovery. As shown in Fig. 3, CA-SOA consists of three major components – an

```

;; Facts for describing calendar profile
(deftemplate Calendar "define calendar profile"
  (slot Person-ID)
  (slot Location)
  (slot Activity)
  (slot Time)
  (slot Privacy)
  (slot Attendee))
;; Facts for describing social profile
(deftemplate Social "define social profile "
  (slot Person-ID)
  (slot Privacy)
  (slot Partner))
;; Facts for describing environment profile
(deftemplate Environment "define environment profile "
  (slot Person-ID)
  (slot Network)
  (slot Situation)
  (slot Device))
(deftemplate QoS "define quality of service profile "
  (slot Person-ID)
  (slot FuncReq)
  (slot NonFuncReq))

;; context-aware inference rules
(defrule Q1 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00")))
=>
(assert (Calendar (Person-ID ?personID) (Activity ?activity))))
(defrule Q2 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00")))
=>
(assert (Calendar (Person-ID ?personID) (Location ?location))))
(defrule Q3 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00")))
=>
(assert (Calendar (Person-ID ?personID) (Attendee ?attendee))))
(defrule Q4 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00"))
(Social ((Person-ID ?personID&Steve) (Partner ?partnerID)))
=>
(assert (Calendar (Person-ID ?partnerID) (Activity ?activity))))
(defrule Q5 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00"))
(Location ?location))
(Environment ((Person-ID ?personID) (Location ?location)))
=>
(assert (Environment ((Person-ID ?personID) (Network ?network) (Device ?device))))
(defrule Q6 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00"))
(Location ?location))
(Environment ((Person-ID ?personID) (Location ?location)))
=>
(assert (Environment ((Person-ID ?personID) (Situation ?situation))))
(defrule Q7 ((Calendar ((Person-ID ?personID&Steve) (Time ?time& "2006:09:06:10:00"))
(Location ?location))
(QoS ((Person-ID ?personID)))
=>
(assert (QoS ((Person-ID ?personID) (FuncReq ?funcReq) (NonFuncReq ?non-funcReq))))

```

4. Context-aware Web service provision

Based on the contextual information collected by the context elicitation system, we present our context-aware service oriented architecture (CA-SOA) for providing

agent platform, a service repository, a context-aware matchmaker.

To enhance context-aware service provision, we have implemented three categories of agents in the agent platform – service agents, broker agents, and request agents.

Fig. 3. Context-aware SOA.

Service agents are designed to help service providers formally describe and wrap Web services with contextual descriptions obtained from the service ontology. Request agents are designed to help service requesters formally describe and wrap their service requests with requesters' contextual information obtained from the requester ontology. On one hand, broker agents take services' publishing requests from service agents and save service descriptions and contextual descriptions into service profiles; on the other hand, broker agents take requests from request agents and initiate context-aware matchmaking, which consists of two phases-capability matchmaking and semantic matchmaking.

As shown in Fig. 3, our service repository is designed to encompass a general UDDI Registry associated with service profiles. If the required services cannot be found by a capability matchmaking process in the UDDI Registry, the semantic matchmaker will be invoked. The semantic matchmaker will decompose service requests into a set of sub-requests, and then schedule an integrated composite service based on the decomposed requests.

Based on the CA-SOA, we present how to provide context-aware service request, publication, and discovery in the following sub-sections.

4.1. Requests' contextual description and wrapping

Service requests are generally specified with keywords or descriptions regarding inputs, outputs, pre-conditions, and

effects of the requests. However, it will cause tremendous overhead if requesters need to manually input much contextual information. Thus, we utilize request agents to automatically wrap requests' contextual information. Fig. 4 indicates a step-wise procedure to describe and wrap requests' contextual information and transform it into a request package.

1. A request agent provides a request parser as an interface (shown in Fig. 5) for guiding requesters to input their request descriptions.
2. The request parser transforms request descriptions to OWL-S.
3. The request agent instantiates the requester ontology to generate instances of requesters' static context profiles.
4. The request agent provides a context wrapping service for obtaining requesters' static contextual information by tagging the attributes defined in requester's instances of static context profiles.
5. The request agent enacts a context elicitation service for obtaining requesters' dynamic contextual information.
6. The request agent wraps and packages the request along with requesters' contextual information.
7. The request agent sends the request package to a broker agent.

A request package in OWL-S as shown below is wrapped to describe a request's contextual information. This partial OWL-S description denotes three conditions:

Fig. 4. Procedure of wrapping requesters' contextual information into a request package.

Fig. 5. Interface of the request parser.

first, the requester is willing to disclose his/her profiles since the value of attribute, *Privacy*, is "*Public*"; Second, the requester is equipped with a "*PDA*" device connected to

the Web through a "*wireless_LAN*" network; third, the requester is in a "*Driving*" situation and he/she is requesting for a Web service with "*99%*" availability.

```

<rdf:RDF>
<owl:DatatypeProperty rdf:ID="Owner_Privacy">
  <rdfs:domain rdf:resource="#Calendar_Owner"/>
  Public
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Default_Value_of_Device">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  PDA
  <rdfs:domain rdf:resource="#Preferences"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Network">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  Wireless_LAN
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Availability">
  <rdfs:domain rdf:resource="#Non-functional_Constraints"/>
  >
  99%
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
  <Situation rdf:ID="Driving"/>
</rdf:RDF>

```

4.2. Services' contextual description and wrapping

To avoid overhead caused by manual inputs of services' contextual description, we utilize service agents to automatically wrap services' contextual information. Fig. 6 illustrates a step-wise procedure to wrap services' contextual information and publish it to broker agents.

1. A service agent provides a service parser as an interface (shown in Fig. 7) for guiding service providers to input services' descriptions.
2. The service parser transforms service description to OWL-S.

3. The service agent instantiates the service ontology to generate instances of services' static context profile.
4. The service agent wraps and packages the service along with services' contextual information for publication.
5. The service agent sends the published service to a broker agent.

The partial OWL-S description shown below is wrapped to describe a service that can be executed by either "PC" or "PDA" devices via "wireless_LAN" or "GPRS" with "95%" availability.

```

<owl:Datatype-
Property
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="Network">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  Wireless_LAN, GPRS
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Device">
  <rdfs:domain rdf:resource="#platform"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  NB,PDA
  <rdfs:domain rdf:resource="#OS"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  Windows NT, Windows XP
  <rdfs:domain rdf:resource="#Browser"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  Microsoft IE, Mozilla Firefox, Netscape
</owl:DatatypeProperty>

```

Fig. 6. Procedure of services' contextual information wrapping and publication.

Fig. 7. Interface of the service parser.

4.3. Context-aware service discovery

Service discovery is a process of matchmaking between a service request and published services. Based on the aforementioned contextual description of requests and services, our context-aware service discovery utilizes capability matching and semantic matching to perform service discovery. The capability matching approach is to identify the similarities of inputs and outputs between requests and published services. Three matching degrees are defined in capability matching – exact match, plug-in match, and subsumed match. (IN_{Req} and OUT_{Req} denote the inputs and outputs of a request, while IN_{Pub} and OUT_{Pub} denote the inputs and outputs of a published service.)

1. *Exact match*: A published service exactly fulfills a request's requirements. Both the inputs and outputs of the published service match the request's requirements, i.e., $IN_{Pub} = IN_{Req}$ and $OUT_{Pub} = OUT_{Req}$.
2. *Plug-in match*: A published service sufficiently fulfills a request's requirements. The outputs of the published service provides more detailed information than the request's requirements, or the inputs of the published service require less specific information than the request's requirements, i.e., $OUT_{Req} \subset OUT_{Pub}$ or $IN_{Pub} \subset IN_{Req}$.
3. *Subsumed match*: A published service only partially fulfills a request's requirements. The outputs of the published service provide less information than the request's requirements, or the inputs of the published service require more detailed information than the request's requirements, i.e., $OUT_{Pub} \subset OUT_{Req}$ or $IN_{Req} \subset IN_{Pub}$.

For semantic matchmaking, we have utilized broker agents, as show in Fig. 3, to handle the context-aware service discovery. Broker agents help service repositories to maintain references of service profiles published by service agents. They only keep references while the service agents keep the original service ontology. Thus, service agents need to inform broker agents whenever service profiles are updated. Broker agents also provide caching mechanisms to improve search performance when the same service is requested by multiple requesters. Based on Fig. 3, the procedure of broker agents-enabled context-aware semantic matchmaking is addressed as follows.

1. A request agent sends a request wrapped with requester's contextual information to a broker agent.
2. The broker agent forwards the request to the capability matchmaker and performs capability matching.
3. If no matched service can be found, the semantic matchmaker will decompose the request into sub-requests based on requester's contextual information, and repeat capability matchmaking in Step 2 for each sub-request.
4. The semantic matchmaker returns matched services to the broker agent.
5. The broker agent replies the matched services to the request agent.

5. Experiments and discussions

We use two indexes – *Precision* and *Recall* to evaluate the performance of context-aware service provision (e.g., providing service discovery). *Precision* is the fraction of

the found services that are considered as relevant; *Recall* is the fraction of the relevant services that has been found. We formally define them as follows:

$$\text{Precision} = \frac{|Ra|}{|A|}, \quad \text{Recall} = \frac{|Ra|}{|R|}$$

where A contains a set of relevant services been found, $|A|$ is the number of services in A ; R contains a set of services been found that are considered relevant, $|R|$ is the number of services in R ; and Ra contains a set of services as the intersection of the sets R and A . $|Ra|$ is the number of services in Ra .

We have implemented and compared three approaches of service discovery – keyword search, capability matching, and semantic matching. Let a service request of a Web-based meeting over the Internet is described by the following four service descriptions, Web meeting, net meeting, tele-meeting, and wireless-meeting, respectively. The search

Table 1
Service description and search results corresponding to the three approaches

Service description	Keyword search		Capability matching (CM)		Semantic matching (SM)	
	Precision	Recall	Precision	Recall	Precision	Recall
Web meeting	0.636	0.671	0.721	0.746	0.818	0.834
Net meeting	0.573	0.623	0.653	0.632	0.783	0.726
Tele-meeting	0.517	0.546	0.561	0.612	0.729	0.674
Wireless-meeting	0.459	0.531	0.542	0.565	0.751	0.713

Fig. 8. Precision of three different matching methods.

Fig. 9. Recall of three different matching methods.

results of using the three approaches are summarized in Table 1. Figs. 8 and 9 illustrate the results in diagrams.

As indicated in Figs. 8 and 9, for the three service discovery approaches, the *Precision* and *Recall* of semantic matching has better performance than the others. This indicates that the services discovered by semantic matching are more relevant because these services have similar contextual descriptions, and they are more likely to be in the same context domain.

6. Conclusion and future research

In this paper, we have presented a context model to formally define contextual description pertaining to service requesters and services. A JESS-enabled context elicitation technique is designed for collecting contextual information. Based on the context model and the JESS rule base, we presented our context-aware service oriented architecture (CA-SOA) for context-aware Web services provision including service request, publication, and discovery. 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-SOA 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 efficiency of service composition, effectiveness of service verification, and reliability of service execution.

References

- Abowd, G. D., Ebling, M., Hung, G., Lei, H., & Gellersen, H. W. (2002). Context-aware computing: guest editors' introduction. *IEEE Pervasive Computing*, 1(3), 22–23.
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Science and Technology at Scientific American.com*. <<http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2>>
- Broens, T., Pokraev, S., Sinderen, M., Koolwaaij, J., & Costa, P. H. (2004). Context-aware, ontology-based, service discovery. In *Proceedings of the 2nd European Symposium on Ambient Intelligence (EUSAI 2004)* (pp. 72–83). Netherlands.
- Chen, G., & Kotz, D. (2000). A Survey of Context-Aware Mobile Computing Research. *Dartmouth Computer Science Technical Report TR2000-381* (pp. 1–16).
- Dey, A. K., & Abowd, G. D. (1999). Toward a better understanding of context and context-awareness. *Technical Report GIT-GVU-99-22*. Georgia Institute of Technology.
- Ernest, F. H. (2003). *Jess in action: Java rule-based systems*. Manning Publications.
- Khedr, M. (2005). A semantic-based, context-aware approach for service-oriented infrastructures. In *Proceedings of second IFIP international*

- conference on wireless and optical communications networks (WOCN 2005) (pp. 584–588). United Arab Emirates.
- Khedr, M., & Karmouch, A. (2004). Negotiating context information in context-aware systems. *IEEE Intelligent Systems*, 19(6), 21–29.
- Korkea-aho, M. (2000). Context-aware applications survey. <<http://users.tkk.fi/~mkorkeaa/doc/context-aware.html>>.
- Kwon, O. (2006). The potential roles of context-aware computing technology in optimization-based intelligent decision-making. *Expert Systems with Applications*, 31(3), 629–642.
- Kwon, O., Choib, S., & Park, G. (2005). NAMA: a context-aware multi-agent based web service approach to proactive need identification for personalized reminder systems. *Expert Systems with Applications*, 29(1), 17–32.
- Kwon, O., Shin, J. M., & Kim, S. W. (2006). Context-aware multi-agent approach to pervasive negotiation support systems. *Expert Systems with Applications*, 31(2), 275–285.
- Kwon, O., Yoo, K., & Suh, E. (2005). UbiDSS: a proactive intelligent decision support system as an expert system deploying ubiquitous computing technologies. *Expert Systems with Applications*, 28(1), 149–161.
- Lemlouma, T., & Layaida, N. (2001). The negotiation of multimedia content services in heterogeneous environments. In *Proceedings of the eighth international conference on multimedia modeling (MMM 2001)* (pp. 187–206). Netherlands.
- Moran, T. P., & Dourish, P. (2001). Introduction to this special issue of context-aware computing. *Human-Computer Interaction*, 16(2–4), 1–8.
- Mostefaoui, S. K., & Hirsbrunner, B. (2004). Context aware service provisioning. In *Proceedings of the IEEE international conference on pervasive services (ICPS)* (pp. 71–80). Lebanon.
- Mostefaoui, S. K., Tafat-Bouaid, A., & Hirsbrunner, B. (2003). Using context information for service discovery and composition. In *Proceedings of fifth international conference on information integration and Web-based applications and services (iiWAS)* (pp. 129–138). Indonesia.
- Roman, M., Hess, C. K., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002). Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1(4), 74–83.
- Satyanarayanan, M. (2004). The many faces of adaptation. *IEEE Pervasive Computing*, 3(3), 4–5.
- Schilit, B. N., Adams, N. I., & Want, R. (1994). Context-aware computing applications. In *IEEE proceedings of the workshop on mobile computing systems and applications* (pp. 85–90). USA.
- Sycara, K., Klusch, M., & Lu, J. (2002). *LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems* (5). Kluwer Academia Publishers, pp. 173–203.
- Sycara, K., Paolucci, M., Ankolekar, A., & Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic Web services. *Journal of Web Semantics: Science, Services and Agents on the WWW*, 1(1), 27–46.
- Yang, S. J. H., Tsai, J. J. P., & Chen, C. C. (2003). Fuzzy rule base systems verification using high level Petri nets. *IEEE Transactions on Knowledge and Data Engineering*, 15(2), 457–473.
- Zhang, J., Zhang, L. J., Quek, F., & Chung, J. Y. (2005). A service-oriented multimedia componentization model. *International Journal of Web Services Research*, 2(1), 54–76.