Distributed Context-aware Agent Architecture for Global Service Discovery

Knarig Arabshian and **Henning Schulzrinne** [1]

Abstract. We present a novel distributed context-aware service discovery system that is built on top of a global service discovery architecture, GloServ. A context ontology maps context attributes to service classes within GloServ. Context-aware agents are distributed globally and map combinations of context attributes to specific GloServ queries. These agents also record history of context combinations in order to provide services to those users who can specify their context, but are not aware of appropriate services within their environment. We discuss the details of the architecture, the algorithms for mapping context attributes to services, and the details of the protocol.

Keywords: Context-aware computing; pervasive computing; service discovery; ontologies

1 Introduction

In our previous work, we describe a novel global service discovery architecture that is used as an underlying framework for ubiquitous and pervasive computing environments, GloServ [4] [6] [7] [5]. GloServ operates on wide as well as local area networks and supports a large range of services that are aggregated and classified in ontologies. A partial list of these services include: events-based, physical location-based, communication, e-commerce or web services.

Currently, context-aware systems, such as, [11], [14], are limited to local centralized networks which prevent global scaling. The goal of these architectures is to aggregate context from different sources within a local environment in order to determine the appropriate service for the user. The goal of these architectures is to aggregate context from different sources within a local environment in order to determine the appropriate service for the user. Context-aware systems, that are distributed, use simple attribute-value descriptions of context and services, which results in only exact query matches without considering related ones. We improve on current systems by providing a distributed context-aware service discovery system that receives context information from various sources and translates them into specific GloServ queries. Additionally, we use the Web Ontology Language (OWL) [1] to classify context and the services within them resulting in richer semantic descriptions and query results that return not only exact information, but related information. As a part of our ongoing work, we have added an agent architecture on top of GloServ that classifies the services according to context. These agents receive context information, map them to the appropriate GloServ service classes, and translate them to GloServ queries.

There has been much work done in defining and categorizing context [12] [3] [17] [8]. We adopt the definition and categorization of context specified in [12], which is defined 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 applications themselves." Furthermore, it is categorized into three main classes:

- Computing environment: available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing.
- User environment: location, collection of nearby people, and social situation.
- Physical environment: lighting and noise level.

We use a service context ontology that classifies GloServ service classes according to their context attributes. Agents, distributed similarly to GloServ and built on top of it, hold this context information. The service context ontology has two main purposes: 1)it describes the context attributes of each of the high-level service classes within GloServ 2)it is used to map a distributed network topology.

The service context ontology classifies GloServ service classes within the three main context classes defined above. GloServ service classes are then categorized into these context classes. Although many services can be classified into all three context categories, one of the categories will be of highest priority. Service attributes are ranked in a partial order within GloServ in order to prioritize the importance of each attribute. The service class is categorized in the context ontology according to these priorities.

To further motivate the necessity for a distributed context-aware service discovery architecture, we look at the following scenarios that use user-environment contexts. The first one is of a traveler who already knows what type of services he wants based on his context. Suppose he has a rule set up that when his context information consists of time, location and activity: *day time*; *Paris*; *tourism* and *walking*, then he must query for a Restaurant or Museum service. His device automatically sends this information to a context-aware agent, along with his preferences, when the context conditions are satisfied. The agent routes this information to the correct agent that handles this service context class. That agent then stores the context information, processes it, and presents the user with a specific GloServ query. The traveler stores this query and the next time this context condition is true, a direct query is issued to GloServ. The second scenario is when the traveler does not know what service he needs but knows his given context. In this case, it sends the context information to the agent and the agent searches the context ontology and matches the given context combination to a set of service contexts. The user receives a list of possible services that are appropriate to his situation and indicates which ones he is interested in, along with his

[1] Department of Computer Science, Columbia University, New York, NY 10027, USA, email: {knarig, hgs}@cs.columbia.edu

preferences, and the agent again provides him with specific GloServ queries.

Since we expect that these users have thin clients, they have limited processing capability. Thus, holding large context ontologies on their systems is not wise. Instead, a generic context ontology is distributed across context-aware agents that keep track of context history and find appropriate services for users. This asserts that distributing context is necessary.

Below we describe the architecture and algorithms that allow these types of service requests to be issued. Section 2 gives background information on GloServ. The distributed context-aware agent architecture and the mechanisms for context attribute matching to services is described in Sections 3 and 4. Section 5 describes related work in the area and we conclude in Section 6.

2 Overview of GloServ

GloServ classifies services in an OWL DL ontology. This classification defines service classes and their relationships with other services and properties. Scaling is achieved by mapping the ontology onto a hierarchical peer-to-peer network of services. This network exploits the knowledge obtained by the service classification ontology as well as the content of specific service registrations. The hierarchical network is formed by connecting the nodes between the high-level, disjoint services within the service classification. The peer-to-peer network is formed between equivalent or related services via a Content Addressable Network (CAN) [16].

GloServ servers (GloServers) have three types of information: a service classification ontology, a thesaurus ontology and if part of a peer-to-peer network, a CAN lookup table. The high-level service classification ontology is not prone to frequent changes and thus can be distributed and cached across the GloServ hierarchical network. Each high-level service will have a set of properties that are inherited by all of its children. As the subclasses are constructed, the properties become specific to the particular service type. The thesaurus ontology maps synonymous terms of each service to the actual service term within the system. We employ methods described in [2] for creating the thesaurus ontologies using OWL. Figure 1 gives an overview of how servers are found in GloServ.

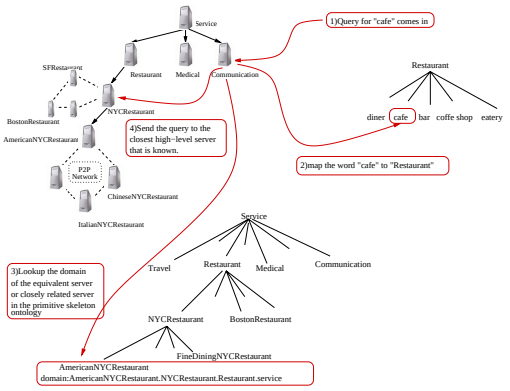

Figure 1. Finding servers in GloServ

Services are represented as instances of the service classes and usually reside in the more specific, lower levels of the ontology. At the lower levels, maintaining a purely hierarchical ontology structure becomes difficult as there are many overlaps between classes. Thus in order to efficiently distribute service instances according to similar content, servers that hold information on similar classes are distributed in a peer-to-peer network. We employ a Content Addressable Network peer-to-peer architecture to distribute classes with similar content. The CAN architecture is generated as a network of n-level overlays, where n is the number of subclasses nested within the main class. An example of an ontology classification using the *Restaurant* class and the CAN overlay network generated is seen in Figure 2. The first CAN overlay is a d-dimensional network which has the first level of subclasses of the *Restaurant* class. The number of dimensions is determined by the number of nodes contained within the CAN.

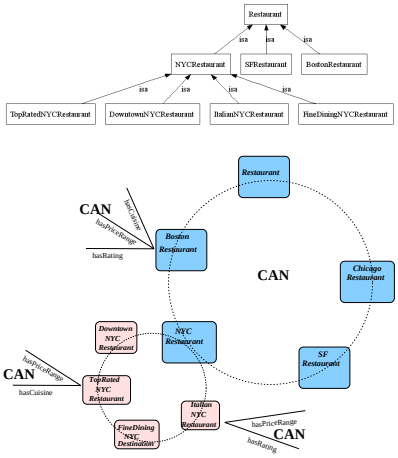

Figure 2. CAN overlay network

Due to using ontologies and mapping these to a hierarchical peer-to-peer network, services are described with richer semantics and distributed efficiently. This results in automated and intelligent registration and querying of services in a global environment.

3 Context-aware Agent Architecture

3.1 Context Ontology

The context-aware agent architecture is based on an OWL context ontology classification. The context ontology represents service context and is formed with the three categorizations defined above: *computing environment*, *user environment*, *physical environment*. Furthermore, context attributes are defined for each of these classes. For example, the *user environment* class has a subset of attributes such as location, activity, time. The attributes are OWL object data type properties, which have as their range a set of OWL classes. The location attribute maps to the *Location* class which classifies global regions into country-state/county-city, zip code, or longitude-latitude coordinates. The *ComputingEnvironmentContext* class may also have these context attributes, but it will have a higher rank for attributes networkCapacity and connectivity. The *Telecommunication* service class within the GloServ service classification ontology,

ranks its properties in a full or partial order. It has networkCapacity, connectivity and may also have location properties, among others. The properties are ranked during the construction of the ontology. A telecommunication service relies heavily on the network capacity and thus this property is ranked highest. It is then matched to all the properties within the service context ontology and grouped within the class that has this property as the highest rank. Since the highest ranking properties for the *ComputingEnvironmentContext* are networkCapacity and connectivity, the telecommunication service context is classified under this context class.

The three context environment classes are disjoint from each other because they each have different priority orders assigned to each of the context attributes. Even if they share a certain attribute, they will not share the priority level of that attribute. This way, when a combination of contexts is classified, the priorities are taken into consideration and this causes it to be classified under one of the three context classes.

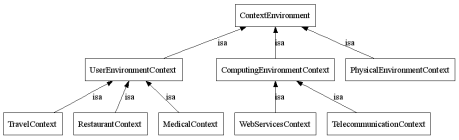

Figure 3. Context Ontology of GloServ Services

3.2 Agent Architecture

The context-aware agent architecture sits on top of the GloServ architecture. It consists of context-aware user agents co-located with GloServers. Since each agent handles a specific service class, it is aware of the GloServer hosts that handle those services. In case a GloServer goes down the agents find the new one by querying for the service class once again. Figure 4 gives an overview of the context-aware agent architecture.

GloServ initially classifies high-level services in a hierarchical primitive ontology where subclasses of a given class are disjoint from each other. Two classes are disjoint when the intersection of their instances is the empty set. These service classes establish the main service classification. As the ontology is formed further down in the hierarchy, classes become more specific and are not strictly disjoint from their siblings. However, for the case of classifying services within a context ontology, working with the high-level services is sufficient. The main purpose of this ontology is to map various context attributes to service classes. GloServ already has a robust hierarchical peer-to-peer network architecture which handles service classification of both high-level services as well as similar services. Thus, it performs querying for a service by returning both exact and related matches. Matching is done by creating a temporary query subclass that contains the query restriction and classifying it. Classes that are equivalent or subclasses of the query subclass have exact matches. Classes that are siblings and share property values are related matches. [5] gives more detail on the query matching algorithm. Since GloServ already handles intelligent querying of related services, it is not necessary to replicate this within the context-aware agent architecture. Therefore, the context-aware agent architecture is a purely hierarchical structure and is created for the purpose of mapping context attributes to high-level service classes within GloServ.

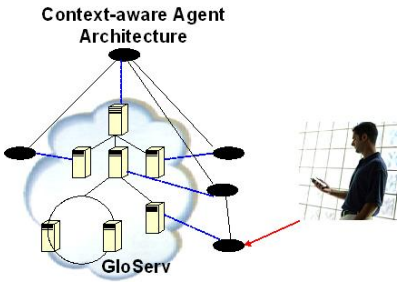

Figure 4. Context-Aware Agent Architecture on top of GloServ

Each agent holds context ontologies of its parent and children in order to navigate the requests to the appropriate agent. It also contains two types of thesaurus ontologies: one mapping synonymous service names to the actual service classes within the context ontology; the other mapping synonymous attributes to actual attributes within the context ontology. So the words ”cafe” and ”diner” map to the *RestaurantContext* service class; the words ”place” and ”street” map to the location attribute.

Bootstrapping agents into the context-aware agent architecture is also done similarly to GloServ as seen in Figure 1, except it uses the context ontology for determining the host. Each agent represents a context service class and its hostname is determined by looking at the primitive skeleton ontology. Hostnames will follow the hierarchical format. For instance, as seen in the context ontology in Figure 3 the *Restaurant* class's http URL will be RestaurantContext.UserEnvironmentContext.Context. As an agent is assigned to a hierarchical network, it updates the ontology to include its host information. Users access a well-known URI which maps to a certain number of random high-level context-aware user agents. If the user knows the service classes it is looking for, the agent matches the service class names, using a thesaurus ontology, to the actual class names within the context ontology. The initial agent contacted will forward the user's request to the agent that handles the service context class the user is looking for. Thus, if initially the *TelecommunicationContext* class is contacted, and the user is searching for a ”cafe”, the agent maps the word ”cafe” to the *RestaurantContext* class and checks to see if it knows the hostname of the agent handling that class. If it does not know this, it finds the nearest ancestor to the *RestaurantContext* class that it does know the hostname of and forwards the user's request to that agent. This continues until the correct agent is contacted.

4 Mapping Context Attributes to Specific GloServ Queries

We describe different protocols on how various users can connect to the context-aware agents and search for services. There are two different context-aware query cases, as we mentioned in Section 1. In the first case, the user has a set of rules which translate to a set of services. When the rule conditions are met, it sends this rule to the context-aware agent and after a few more message exchanges, receives a specific GloServ query. The second case assumes that the user has context information but does not know what type of ser-

vice it needs. The context-aware agents must reason what the best service for this user is by matching the user's context attributes to the service context attributes within the context ontology. Below we discuss these two cases.

4.1 Creating GloServ Queries With Known Services

In the first case, the user is aware of the services it needs and wants to set up a set of rules to initiate the service requests. In order to do this, a user contacts a context-aware agent and downloads the context attributes of the service class it wants to set rules for.

To illustrate this with a concrete example, we continue with the tourism example. If the tourist already knows that it wants to be notified of $Restaurant, Museum$ or $Boutique$ services while on vacation, it first contacts a context-aware agent and submits these service classes to it in the following message:

SERVICE
Restaurant
Museum
Boutique

The first agent contacted finds the appropriate agents that handle these service contexts with the methods mentioned in section 3.2. The agents handling the services restaurant, museum and boutique, collect the context information from the user. They also query GloServ for the actual service attributes. Thus if the user submits the service *Restaurant*, the agent handling the restaurant service will find the following attributes:

Context Attributes: location, time, activity
Service Attributes: hasCuisine, hasLocation, hasRating

The agents then present the user with a form that allows him to fill out the appropriate values of each set of attributes.

The agent receives this information, creates a restricted subclass with the specified context attributes and classifies this within its ontology. This allows the agent to keep track of context history of a particular service class since its restricted subclasses represent context combinations that future users can benefit from. The agent then creates a GloServ query with the restrictions specified by the user, instantiates it within the restricted subclass created, and returns the query to the user. The restricted subclass is formed in OWL as follows:

```
<owl:Class rdf:ID="contextClass1">
 <rdfs:subClassOf rdf:resource="#RestaurantContext"/>
 <owl:equivalentClass>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Restriction>
     <owl:onProperty rdf:resource="#activity"/>
     <owl:someValuesFrom>
      <owl:Class>
       <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Vacation"/>
        <owl:Class rdf:about="#Walking"/>
       </owl:intersectionOf>
      </owl:Class>
     </owl:someValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="#location"/>
     <owl:someValuesFrom rdf:resource="#NewYorkCity"/>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="#time"/>
     <owl:someValuesFrom rdf:resource="#OneAM"/>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

The user then modifies his rule so that when his context conditions are met, a query is sent directly to GloServ. We represent the rule in a generic form which represents a case-based reasoning:

$location(?x, ?y) \wedge time(?x, day) \wedge activity(?x, walking) \wedge activity(?x, tourism)$
$\Rightarrow Query(Restaurant, \text{“}(hasCuisine\ some\ Italian) \wedge (hasLocation\ some\ ?y)\text{”})$
$location(?x, ?y) \wedge time(?x, day) \wedge activity(?x, walking) \wedge activity(?x, tourism)$
$\Rightarrow Query(Museum, \text{“}(hasStyle\ some\ Art) \wedge (hasLocation\ some\ ?y)\text{”})$
$location(?x, ?y) \wedge time(?x, day) \wedge activity(?x, walking) \wedge activity(?x, tourism)$
$\Rightarrow Query(Boutique, \text{“}(hasItems\ some\ Clothing) \wedge (hasLocation\ some\ ?y)\text{”})$

Figure 5 illustrates the exchange of messages between the user and the agent.

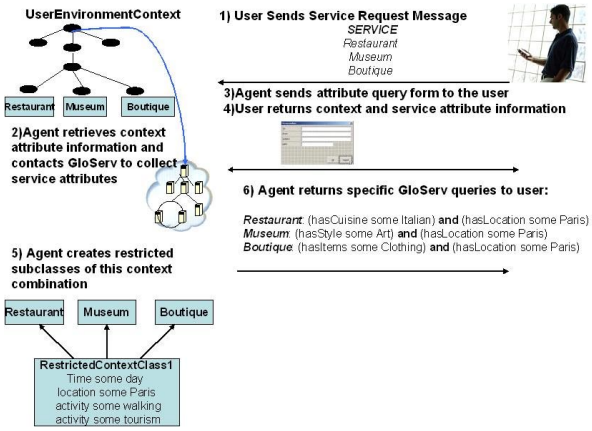

Figure 5. One scenario of context-aware service discovery

4.2 Creating GloServ Queries With Unknown Services

As the agents instantiate restricted context classes, it collects past user experiences for future users who do not know what services to search for but know their context information. In the case of the tourist, he may not know what type of services are available to him, but he is aware of certain context conditions and wants to know the service classes pertaining to him.

The user sends the agent a message similar to the one above, except instead of putting in service classes, it specifies a list of context conditions such as: city=Paris, weather=warm, time=noon, and ranks these attributes by ordering them. The agent uses the attribute thesaurus ontology to match these context attributes to the actual ones in the system. The agent then creates a restricted subclass and since the first level in the ontology is restricted by the rank of each attribute, the restricted class gets classified under one of the three subclasses of the *Context* class. The agent then sends the user's information to these agents and the message continues propagating to the agents below until there is a hit. A hit occurs within an agent when the restricted class is classified as either an equivalent class or superclass of one or more of the classes within the agent's ontology; this signifies that there is a context combination that is a subclass or

equivalent class to the user's context combination. Once a hit occurs, the agent knows the class of services that the user may be interested in and responds to the user with a query form specific to the service class. The steps following are similar to the previous scenario's.

5 Related Work

Current work done in context-aware service discovery concentrates either on local area networks or uses simple representations of context and services. Cobra [11] uses OWL to represent context and collects context from different sources and reasons with rules and policies for user preferences. However, it is not a distributed system and the example scenarios are for home environments.

Distributed context-aware systems developed recently, such as [15] and [10], lack in semantically rich representation of data. [15] describes a scalable system that collects context information from different sources. However, it does not describe the distributed nature of the architecture in much detail. It also uses object oriented or an attribute-value model for service description rather than ontologies. [10] collects context information in a distributed environment using INS [9] as its service discovery architecture. However, the INS system also uses simple attribute-value representation of services using XML. The context-aware discovery system presented in this paper differs from all of these in that it both distributes context data globally as well as classifies them using ontologies. It also uses GloServ for service discovery which also uses ontologies to describe services and is distributed globally.

6 Conclusion and Future Work

In conclusion, we have presented a distributed context-aware agent architecture for global service discovery. Unique characteristics of this architecture is that it aggregates context data in a distributed system using the OWL ontology for richer semantics, in order to learn context history over time. It also uses GloServ to map context attributes to services classes so that users can issue specific context-aware queries.

The design of the distributed context-aware agent architecture is part of our ongoing work. We have implemented the hierarchical peer-to-peer GloServ architecture using OWL and are evaluating the querying mechanism. The context-aware agents will be built on top of GloServ and will use a subset of the GloServ API. GloServ uses the Protege [13] API to process OWL files.

For future work, we will look into saving specific user or group profiles. Currently, the information held in the agent architecture is generic context provisioning information of GloServ services. Implementing user profiles will require that the context-aware agent architecture be more robust and fault-tolerant and will thus be improved to also have a peer-to-peer distribution. We also plan on including not only context-aware searches for structured data, as in GloServ, but also unstructured Google-type data searches.

7 Acknowledgement

This work is supported by a grant from Nokia Research Center. We would also like to acknowledge the contributions of Dirk Trossen and Dana Pavel from Nokia Research.

REFERENCES

[1] Owl web ontology language.

[2] Quick guide to publishing a thesaurus on the semantic web. W3C Working Draft, May 2005.

[3] G.D. Abowd A.K. Dey and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. In *Human-Computer Interaction, 16(2-4):97-166*, 2001.

[4] Knarig Arabshian and Henning Schulzrinne. Gloserv: Global service discovery architecture. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous)*, August 2004.

[5] Knarig Arabshian and Henning Schulzrinne. Hybrid hierarchical and peer-to-peer ontology-based global service discovery system. Technical Report CUCS-016-05, Columbia University, April 2005.

[6] Knarig Arabshian and Henning Schulzrinne. An ontology-based hierarchical peer-to-peer global service discovery system. *Journal of Ubiquitous Computing and Intelligence (JUCI)*, 2006.

[7] Knarig Arabshian, Henning Schulzrinne, Dirk Trossen, and Dana Pavel. Gloserv: Global service discovery using the OWL web ontology language. In *IEE International Workshop on Intelligent Environments (IE05)*. IEE, June 2005.

[8] M. Theimer B. Schilit. Disseminating active map information to mobile hosts.

[9] Magdalena Balazinska, Hari Balakrishnan, and David Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery, 2002.

[10] G. Chen and D. Kotz. Context-sensitive resource discovery, 2003.

[11] Harry Chen, Tim Finin, and Anupam Joshi. Semantic Web in in the Context Broker Architecture. In *Proceedings of the Second Annual IEEE International Conference on Pervasive Computer and Communications*. IEEE Computer Society, March 2004.

[12] A. Dey. Understanding and using context. *Personal and Ubiquitous Computing, Vol. 5, No. 1*, 2001.

[13] J. Gennari, Mark A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubzy, H. Eriksson, N. F. Noy, and S.-C. Tu. Evolution of protg: An environment for knowledge-based systems development. Technical report, Stanford University, 2002.

[14] Masanori Hattori, Kenta Cho, Akihiko Ohsuga, Masao Isshiki, and Shinichi Honiden. Context-aware agent platform in ubiquitous environments and its verification tests. *Systems and Computers in Japan*, 35(7):13–23, 2004.

[15] N. Honle U. Kappeler D. Nicklas T. Schwarz M. Grossmann, M. Bauer. Efficiently managing context information for large-scale scenarios. In *Third IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2005.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[17] T.Winograd. Architectures for context. In *Human-Computer Interaction 16(2-4):401-419*, 2001.