GloServ: Global Service Discovery using the OWL Web Ontology Language

Knarig Arabshian and Henning Schulzrinne
Department of Computer Science
Columbia University, New York, NY
{knarig,hgs}@cs.columbia.edu

Dirk Trossen and Dana Pavel
Nokia Research Center
Burlington, MA
{dirk.trossen,dana.pavel}@nokia.com

March 24, 2005

Abstract

Due to the growth in ubiquitous computing technology in the past few years, the need for context-aware service discovery across wide area networks is becoming prevalent. Current service discovery protocols lack in the ability to scale to large networks as well as semantically describe services. Thus, we propose GloServ, which is a global service discovery architecture that locates services throughout wide and local area networks. Intelligent agents within GloServ use the OWL Web Ontology Language to provide detailed service descriptions, and provide users with automatic registration and querying. This paper discusses a revision of GloServ that uses the OWL Web Ontology Language, as its ontology for describing services instead of the Resource Description Framework Schema (RDFS). We will describe how OWL is used for automated registration and querying of services throughout a global network.

1 Introduction

GloServ is a global service discovery architecture that addresses the need for wide area service discovery for ubiquitous and pervasive computing. It operates on wide area as well as local area networks. Any type of service is handled which may include events, location-based services, communication or web services.

There are many different scenarios global service discovery systems are useful in. Imagine an avid traveler who is interested in different events or services that are taking place in a particular city. A New Yorker visiting Paris, for instance, wants to be notified of all the concerts taking place in Paris. When he lands in the airport, his device will issue a query pertaining to his preference, such as classical music concerts, and then try to locate classical concerts in that area. Another example would be of a doctor visiting another hospital who wants to discover different medical services. The doctor issues a query and obtains the types of services she is interested in automatically.

Originally, GloServ was designed using the Resource Description Framework Schema (RDFS) [5]. However, RDFS has a main limitation in that it provides only a hierarchical classification of services. Thus, we have redesigned the system to use an ontology for service descriptions such as the OWL Web Ontology Language [1], which has been adopted as the ontology of choice for the future of the Semantic Web. An ontology defines the terms used to describe and represent an area of knowledge. OWL defines classes and the relationships between them as well as their various properties. Rather than

just having a hierarchical relationship between classes, OWL is a logic-based language that provides greater detail in describing the relationships between classes. It expresses various class relationships using Boolean expressions, equivalence and inverse relations, quantification and other logical semantics. By using OWL, intelligent agents within GloServ servers can automatically decide where services should be registered to and how they should be queried for.

Below, we describe the details of the GloServ revision. Section 2 discusses related work. An overview of OWL is given in Section 3. Motivation for an ontology-based design is described in Section 4. Section 5 describes ontologies that will be used in GloServ. The system architecture, automated registration and querying is described in Section 6. The implementation of the revision and future work of GloServ is discussed in Section 7. Finally, we conclude in Section 8.

2 Related Work

Service discovery protocols in use today include SLP [11], standardized by the IETF, Sun Microsystem's Jini [13], and Microsoft's UPnP (Universal Plug and Play) [9]. Additionally, [15] describes an Internet telephony gateway location protocol for voice over IP applications. This protocol architecture called Brokered Multicast Advertisements (BMA) addresses the problem of an IP host finding the IP address of the appropriate gateway in order for a call to be made to a user on the PSTN. These service discovery protocols have various similarities and differences. We will look at a few of these protocols briefly in this section.

SLP has three main components: *User Agents (UA)* which perform service discovery on behalf of a client, *Service Agents (SA)* which advertise location and characteristics of the service on behalf of the service, and *Directory Agents (DA)*, which are optional, record available services and also respond to service requests from UAs. In SLP, there are two modes of operation: one includes the DA and the other does not. When a DA exists, the UAs learn about services by unicasting messages to the DA. Otherwise, UAs send multicast messages to the SAs and learn about available services. When a DA is present, there is faster response time and thus it is scalable to larger networks.

Jini is built on top of the Java object and RMI system. Service registries, similar to SLP's DAs, are used to register service proxy objects and act as lookup services. Through a discovery process, a client downloads the service proxy and invokes it to access the service. The Java class hierarchy defines services and their attributes.

UPnP differs from SLP and Jini in that it doesn't have a central service registry but services just multicast their announcements to control points that are listening to these messages. Control points can also multicast discovery messages and search for devices within the system. XML describes the services in greater detail.

SLP and Jini can cover small networks as well as larger enterprise networks whereas UPnP is appropriate for home or small office networks. The query languages for SLP and Java are simple text-based attribute-value pairs. UPnP provides more descriptive queries through XML. The main drawback to these systems is that they do not cover a wide area network that spans the whole Internet, but rather they are limited to a certain vicinity such as home or enterprise networks. Querying in SLP and Jini is not very detail-oriented. GloServ addresses these issues by functioning in both the wide area as well as providing greater semantic detail in querying.

3 Overview of OWL

In our previous paper [4], we used RDF and RDFS to classify services hierarchically. However, the World Wide Web Consortium has recently approved OWL as a standard for the Semantic Web. OWL builds on RDF and RDF Schema and adds more vocabulary for describing properties and classes such as: relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes. Below we give an overview of the sublanguages of OWL and the characteristics of OWL Classes and Properties.

3.1 Sublanguages of OWL

There are three sublanguages in OWL: OWL Lite, OWL DL and OWL Full. OWL Lite is the least expressive of the three sublanguages. Although it is a bit more expressive than RDFS that in addition to supporting a classification hierarchy, it also provides simple constraints of classes and properties. OWL DL is modeled after description logics and supports maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs. OWL Full is the most expressive of the three sublanguages. The main difference between OWL DL and OWL Full is that in OWL DL, a class is only expressed as a collection of individuals and can not be regarded as an object in and of itself. However, in OWL Full, a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Due to this difference, OWL Full can not be completely supported by OWL Description Logic Reasoners to check for soundness. We have chosen to use OWL DL for two reasons: 1)a service class will only represent a collection of individuals and does not need to be an individual in its own right and 2)we would like to use OWL DL reasoners such as Racer to check for the soundness of OWL documents.

3.2 Characteristics of OWL Classes and Properties

User-defined classes in OWL are all subclasses of the root class owl:Thing. A basic class contains individuals, which are instances of the class, and other subclasses. In addition, OWL supports set operators on classes such as union, intersection and complement. It also allows class enumeration and disjointness.

A property is a binary relation that specifies class characteristics. There are two types of simple properties: datatype and object properties. Datatype properties are relations between instances of classes and RDF literals or XML schema datatypes. Object properties are relations between instances of two classes. Properties can also have greater logical capabilities such as being transitive, symmetric, inverse and functional.

Ontology mapping is useful for combining a set of ontologies and seeing where various equivalences lie. Classes and properties can be equivalent. Individuals, which are instances of classes, can be categorized as the same or different from each other.

3.3 OWL-based Languages

Due to the flexibility of the OWL ontology, specialized OWL-based languages are now being created for specific applications such as service discovery. OWL-based languages for service discovery require languages that give structure to describing and querying services. Two such languages we are considering are OWL-S [6] and OWL-QL [8] which are used for services and querying respectively.

OWL-S is an upper ontology that allows services to be expressed by three main components: the service profile for advertising and discovering services; the process model, which gives a detailed description of a service's operation; and the grounding, which provides details on how to interoperate with a service, via messages. All three components do not have to be used at once, but based on the need of the service. Since GloServ is in its initial stage of service discovery, only the service profile is utilized and the description within the profile is written in OWL DL. However, once GloServ extends to service access and execution, the other two components will be used as well.

The OWL Query Language (OWL-QL) is a formal language and protocol that allows querying and answering agents to conduct a query-answering dialogue in ontologies represented by OWL. The design of OWL-QL makes certain assumptions which fit perfectly with the GloServ architecture. It provides a generic query-answering dialogue system that can be easily implemented in the Semantic Web. We will describe the use of OWL-QL in GloServ further in section 6.2.1

4 Ontology-Based System Design

This section aims at motivating the ontology-based system design. For this, we will use the example of an architecture for distributed context-aware applications. We will first briefly outline this architecture before describing the use of ontologies in such system.

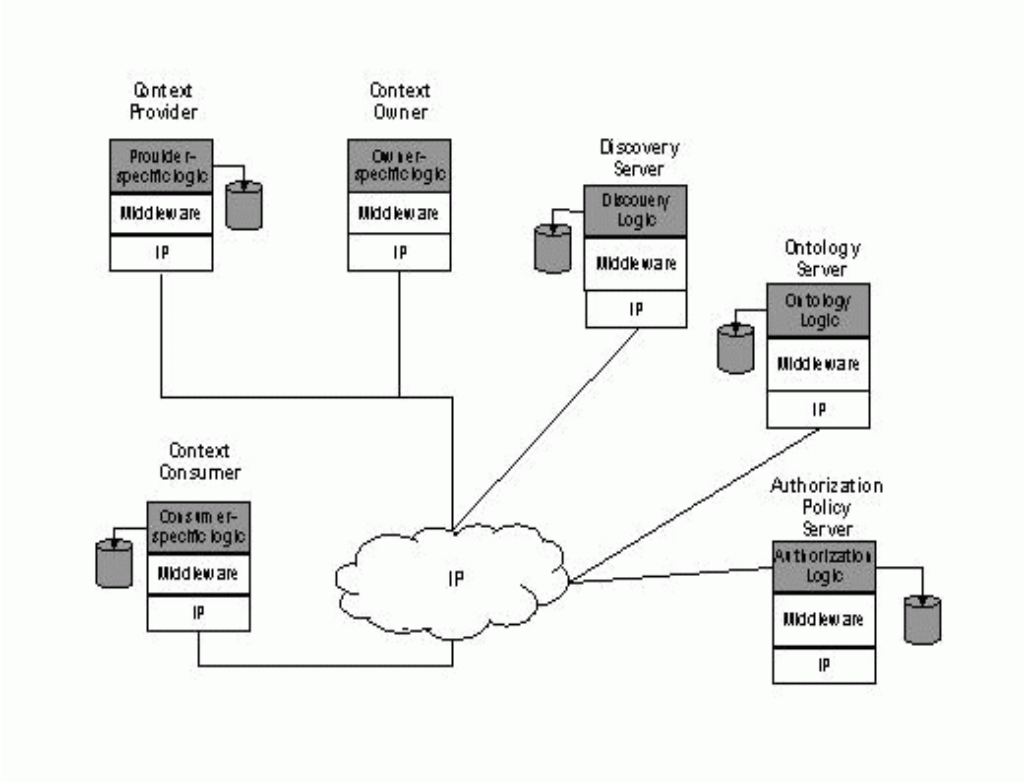

Figure 1: Architecture for distributed context-aware applications

4.1 A System for Distributed Context-aware Applications

Figure 1 shows the architecture of our system. It enables distributed context-aware applications throughout the Internet, i.e., it supports the acquisition of context information from distributed sources in the Internet together with the discovery of these sources. As context information, [7] generally describes "any information that characterizes the situation of an entity". Examples for such information are location, time, activity, or affective state of a person. The acquisition of the information is based on access policies defined by the context owner. Note that it is not intended to dive into the depth of this particular architecture rather than to motivate the use of ontologies for particular pieces of information provided throughout the system. However, the work in [14] outlines issues regarding implementing the system in Figure 1 within a SIP-based environment.

The shown architecture provides different roles within the system. The context consumer and provider engage in a provisioning relationship for a particular piece of context information, such as location, time, activity etc. Access policies of particular context information are defined by the context owner (such as the person whose location is provided), the policies being hosted at the authorization policy server. The discovery of context providers is implemented in the discovery server. The ontology server hosts different system-specific ontologies and related functionality (e.g., implementing algorithms for selecting appropriate ontologies for different usages). In addition to the system ontology server, ontologies in the Internet can be used, possibly within different vertical application areas, such as health care, mobile work force etc. Such usage is enabled through pointing to these ontologies through URLs.

4.2 Use of Ontologies For Context Provisioning and Discovery

In Figure 1, the major piece of information in this system is the particular context provided to the context consumer. In this, the context provider constitutes a particular form of a service provider, namely provisioning requested context information. The system in Figure 1 supplements the functionality of context acquisition with support of access policies and discovery of particular context providers, similar to general-purpose service provisioning environments. As generally described in [4], ontologies are used for various purposes within such system:

- *Automated context provider discovery*: In a context-aware environment, a context consumer needs to know which context provider could potentially deliver the context it needs. For example, an application may need to know the people within 20 meters vicinity of the user. In order to provide such information, the application needs to find the context provider that can provide this kind of information. With the use of ontologies, each context provider can model the kind of context information it provides and advertise it to a service registry. This will enable context consumers to automatically discover providers by querying for appropriate characteristics of the required services, which can be mapped by the discovery server onto the available registered context provider.

- *Automated context acquisition invocation*: Service invocation in a context-aware system, i.e., the acquisition of particular context information, is supposed to happen in an automated manner, i.e., there is no need for the user to manually configure provider relationships or edit configuration information. Ontologies provide a declarative and computer-interpretable description of the parameters for remote service invocation methods. Hence, a context consumer is able to interpret the required input and expected output from the ontology description.

- *Automated context (de-)composition and interoperation*: High-level services do not need to be atomic, i.e., provided by a single entity as a single service. Instead, it is possible that a high-level service is composed of different sub-services, along a well-defined goal. In a context-aware

system, such service composition can be seen as a high-level context description that can be decomposed into further subcomponents, eventually into atomic pieces of context information. Such decomposition even enables to replace pieces of context information throughout the runtime of the provisioning in cases where such piece of context information is no longer available but can be replaced by alternative sources. For that, the prerequisites and consequences of using individual pieces of context are to be defined in a proper ontology.

- *Automated execution monitoring*: Through the runtime of a particular context acquisition, the context consumer might want to know the status of the acquisition. The use of ontologies allows for defining declarative descriptions for the state of execution of the acquisition.

In conclusion, the use of ontologies generally facilitates automation of discovery, provisioning, execution and composition of services, i.e., of all major parts in the service execution environment. This is achieved through separating the semantics of the particular functionality from the specifics of its provisioning. Ontologies such as OWL-S allow such separation through definition of service models, process models and the particular grounding within a particular provisioning system, as outlined in the previous section. The remainder of the paper will focus on the enablement of automated discovery. We will generally describe the modeling of services and their registration within a system, such as the one presented in Figure 1.

5 GloServ Ontologies

OWL ontologies in GloServ are used in different ways to represent the knowledge base. There are ontologies for classifying services and GloServ servers that hold service provider information. Below, we describe a few of these ontologies.

5.1 Service Ontology

In the previous version of GloServ, we designed it to be similar to DNS. Services were categorized hierarchically and accessed through URIs that were descriptive of the service and its location. However, with the use of ontologies, this can be simplified. A separate classification system similar to North American Industry Classification System (NAICS) [2] classifies services and establishes ontologies that describe each type of service. An authority such as ICANN [3] delegates the top level services and events to individual GloServ servers. The service categorization is similar to a yellow pages directory. Although the categories may change from time to time, they are not expected to change drastically or frequently, which gives us the ability to pre-define the service classification and implement caching in local user agents.

An ontology needs to be created that describes all service classes and their set relationships. Since there are multiple terms that can describe a certain service, using OWL's equivalence relations, services can be classified into their equivalence classes as well. For example, the service "restaurant" can be made equivalent to "dining". Thus, when creating a service ontology, all the terms that are ordinarily used to describe the service should be included in the ontology as an equivalence relation. As terms continue to evolve, the only thing that needs to change is removing or adding classes to the ontology.

5.2 Ontologies for distributing GloServ servers

For location-based services, GloServ servers are distributed according to location. Besides maintaining a service ontology, a location ontology of all the GloServ servers within an area is also maintained. The

location ontology will consist of location classes with at least one "hasServer" property that points to a GloServ server within the vicinity. Each location server will maintain an ontology of services in its current vicinity. A central governing authority such as ICANN needs to maintain this location ontology of GloServ servers as well.

A benefit to using a location OWL ontology is that locations need not only be classified hierarchically but by other user-defined properties. For instance, using the symmetric property in OWL, an "adjacentRegion" property can be established so that all locations that are near each other can be automatically identified through symmetry. For instance, when looking for a service in a certain region of Manhattan such as the Upper West Side, this region can have the symmetric property of being adjacent to Midtown and Upper East Side. If a user is querying for a certain store in the Upper West Side, and the GloServ server in that area does not have that store, the servers in the adjacent regions are automatically searched.

GloServ servers may also be distributed by other means besides location. For instance, events related to time or other global services not tied to a particular location will have ontologies that distribute GloServ servers via other mechanisms such as specific service classification. For the purposes of this paper, we will focus mainly on location-based services and describe such an architecture. However, other types of services discovery can easily be plugged into GloServ.

5.3 Other Ontologies

In addition to these ontologies, there are others that maintain registration information for every service type. Each service will have different registration components. These ontologies will specify how to register for a service, what information to provide, and verification mechanisms for these service providers.

There may also be ontologies for representing general knowledge such as dictionary or thesaurus ontologies. These will improve performance of the intelligent agents residing within GloServ servers in registration and querying.

6 GloServ Architecture

The revised GloServ architecture is similar to the original except that it now has registries that hold OWL ontologies as well as intelligent agents that enable effective registration and querying. Also, the distribution of GloServ servers is now classified using ontologies rather than having a DNS naming scheme of URIs. Thus a GloServ server will hold information on a variety of services in its local vicinity.

GloServ servers hold ontologies of service providers classified by service type. We abbreviate this term to *GloServer*. There are also registries that maintain ontologies of services. A *Service Registry* holds the skeletal service classification ontology. The *GloServer Distribution Registry* holds the ontologies of the GloServer distribution such as a location-based or subject-based. In addition, intelligent *Service Agents* and *User Agents* handle service registration and querying for service providers and users. Service providers registering in GloServ are verified by *Verification Servers* that specialize in a particular service.

6.1 Service Registration in GloServ

The previous design of GloServ allowed the service provider to choose where to register itself in the service hierarchy, the current version of GloServ automates this process where a user only provides its service type and an intelligent Service Agent deduces which GloServer the document should be registered in. Below we describe the registration mechanism using the restaurant service as an example.

Registration can have either a web-based front end or another means of transferring information. In a web-based mechanism, a user will fill out a generic registration form of a service category. How-

ever, if this is a type of service that doesn't need any human interaction for registration, the registration mechanism can follow another protocol where ontologies are exchanged, parsed and information sent via automated programs. We are currently focusing on web-based registration of services for the purpose of this paper, but we do plan on extending GloServ to encompass all types of automated services as well in the future.

When a restaurant service provider registers within GloServ, it will go to a generic GloServ site and enter in the keyword of its service type. The Service Agent will contact the ontology registries to match any equivalence relationships. It will recognize various user inputs such as "deli" or "diner" as a restaurant service and retrieve the restaurant registration ontology from the Registrar. If no terms are found, then it will display all of its closely related service categories to the user who can then choose the category that matches its own.

Once the service category is identified, and the service registration ontology is retrieved, it is converted to an HTML form for the service provider to fill out. For instance, a default restaurant registration ontology would probably need certain values such as: location information, price range, rating, cuisine type, etc. However, in addition to default tags, a user can enter extra text information that it wants the servers to be aware of such as "music" and "dress code." This free text information is entered under the "hasFreeText" property slot of the ontology. By combining ontological searches with free text searches, there is a greater probability of finding a service that matches the user's query specifically.

The registration ontology also provides information on what type of GloServ distribution this service type uses and references another ontology to access the GloServer information. For location-based services, as in the case of restaurants, the registration information of the service provider is sent to the GloServer nearest to the service provider. The nearest GloServer is determined by mapping the location of the user and retrieving the GloServer URI from the location ontology. Once the GloServer is identified, a Service Agent within the GloServer inserts a reference to the ontology under the correct service type.

Although all GloServers will share the same high-level service classification ontology, each GloServer node might defer in further breakdown of service classification. For instance, restaurants are in abundance in New York City. Thus, each GloServer can have a detailed breakdown of the restaurant service class according to the types of restaurants it has in its vicinity. Subclasses are created for: fast food restaurants, fine dining restaurants, and ethnic restaurants. Each subclass is related to each other in various ways. Fast food restaurants are classified as the inverse of fine dining restaurants. Certain fine dining restaurants will have values from ethnic restaurants. The service agent will look through the ontology submitted by the service provider and place a reference to the restaurant's ontology and URI under the branches that correspond to the appropriate class restrictions. Figure 2 illustrates how services can be classified using OWL.

6.2 User Querying in GloServ

Users searching for services can issue queries in GloServ and find a list of closely matched services. We have chosen to use the OWL Query Language OWL-QL because of its flexibility in issuing different types of queries. Below, we will give a brief overview of OWL-QL and how it ties in with GloServ.

6.2.1 OWL-QL

OWL-QL has been designed with varoius assumptions in order to adapt to different systems. It is assumed that the query answering agent uses automated reasoning to derive answers to queries. It also expects that some servers will only have partial information about a certain subject area. Thus it provides

```
<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID="FastFoodRestaurant">
 <rdfs:subClassOf>
  <owl:Class rdf:ID="Restaurant"/>
 </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Restaurant">
 <rdfs:subClassOf>
  <owl:Class rdf:ID="Service"/>
 </rdfs:subClassOf>
 <owl:equivalentClass>
  <owl:Class rdf:ID="Dining"/>
 </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Event"/>
<owl:Class rdf:ID="Political">
 <rdfs:subClassOf rdf:resource="#Event"/>
</owl:Class>
<owl:Class rdf:ID="Concert">
 <rdfs:subClassOf rdf:resource="#Event"/>
</owl:Class>
<owl:Class rdf:ID="Plumber">
 <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>
<owl:Class rdf:ID="FineDiningRestaurant">
 <rdfs:subClassOf rdf:resource="#Restaurant"/>
</owl:Class>
<owl:Class rdf:ID="RealEstate">
 <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>
<owl:Class rdf:ID="EthnicRestaurant">
 <rdfs:subClassOf rdf:resource="#Restaurant"/>
</owl:Class>
<owl:Class rdf:ID="Holiday">
 <rdfs:subClassOf rdf:resource="#Event"/>
</owl:Class>
<owl:Class rdf:about="#Dining">
 <owl:equivalentClass rdf:resource="#Restaurant"/>
 <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>
```

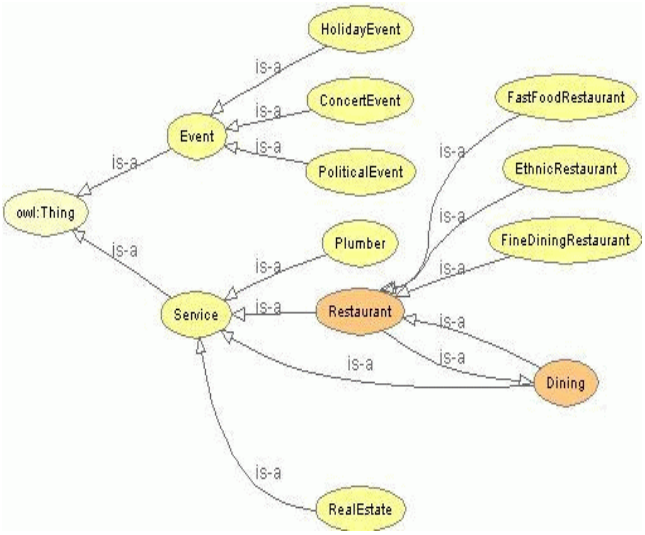

Figure 2: Example of OWL ontology of services

a means for the transfer of partial query results and it also allows the user to specify the maximum number of answers in the result. Additionally, OWL-QL assumes that servers choose the knowledge bases to extract information rather than specifying this in the query itself. Also, the structure of OWL-QL is designed at an abstract level without any constraints on syntactic forms. The specification only describes what types of objects are passed between servers and clients, necessary and optional components of the object types and the responses to expect from the server for each object that is sent. However, the specification does include example syntax that one may use. Finally, the last assumption made in OWL-QL is that knowledge is represented by formally defined semantics and logical theory as in DAML or OWL.

The OWL-QL query pattern is a collection of OWL sentences where some URI references are considered to be variables. For example, the query ("Who owns a red car?") is translated to the following query pattern, (owns ?p ?c) (type ?c Car) (has-color ?c Red). Each binding in the answer is a URI reference or a literal. There are three types of bindings that variables in a query can have: must-bind, may-bind and don't-bind. Since a knowledge base can have a query answer but not have a binding for every variable in the query, it is beneficial to specify how strict the variable binding within a query should be. An example from [8] of how this would be useful is when we consider a knowledge base with sentences that indicate that every person has exactly one father. If Joe is a person, he must have a father but he may not have the value of the property "hasFather". Thus, if the binding is may-bind or don't-bind, Joe will be included in the result but the actual value of the hasFather property will not be indicated. OWL-QL also facilitates

the representation of “if-then” queries by enabling a query premise. A query such as, “If C1 is a Seafood Course and W1 is a drink of C1, then what color is W1?” can be translated into an OWL-QL query. This is exceedingly useful in GloServ where there are many possible answers to a query and using premises narrows down the search pool.

The query-answering dialogue in OWL-QL enables the server to send answers in bundles to the client. This permits the client to engage the user while completing the query. Instead of the user waiting a long time for all the answers to be received at once, it receives periodic answers to its query and terminates the query dialogue once it finds what it needs. The query dialogue includes information for the bundle size, the process handle, and server continuation and termination commands. Figure 3 shows an example of the query-answering dialogue [8].

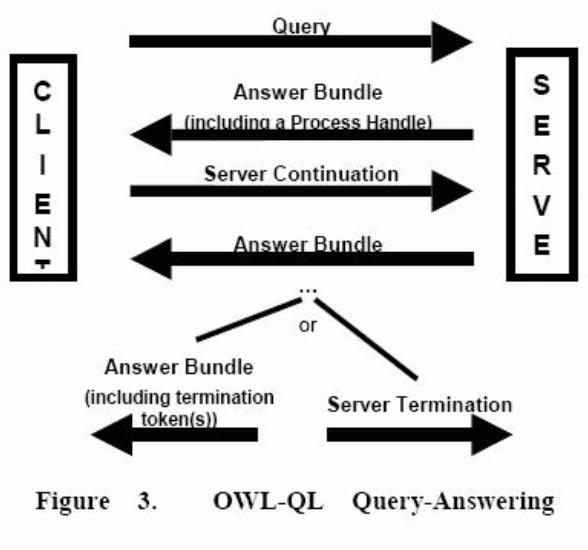

Figure 3. OWL-QL Query-Answering

6.2.2 Using OWL-QL in GloServ

When a user queries in GloServ, the query is sent to a local intelligent *User Agent* (UA). A local UA has a cached copy of the GloServer distribution ontology. This information is cached and periodically renewed to update any changes within the classification. When a user issues a query, it is automatically connected to a local UA. The web interface for querying for services is similar to registration. A generic GloServ query website will be presented to the user for querying. This interface can be changed according to device capabilities.

When the UA receives the user query, it looks through a thesaurus ontology that maps the words to possible classes and properties within the service classification system and formulates a query pattern. It issues the query pattern to the Service Agents residing in the nearest GloServer. Following the OWL-QL protocol, the query-answer dialogue begins and the user is able to see the answers to the query. If the user is not satisfied with any of the query results after a few query-answer sessions, the UA searches the adjacent GloServers. This goes on until the query propagation value n has been reached or the user has found what she is looking for.

The free text portion of the OWL documents are used for further analysis. If there are no hits to the user’s queries, then the free text portion of the OWL documents is matched to the text query of the user.

Possible results are presented to the user. It is also used to filter out existing queries. If the user's query results in many hits, the text portion is analyzed to see if the results can be filtered out.

In addition, context information can be used to improve the query results. As mentioned in Section 4, context acquisition itself may be deemed as a service within GloServ. When administering a context-aware query in GloServ, the query will first go through the context servers in order to determine the correct context of the user. It will then go through the service registries to find the service that pertains to the user's context. Since OWL-QL allows servers to select the knowledge bases to query from, it facilitates the querying mechanism by allowing the servers to propagate the query according to each service's ontology.

7 Implementation and Future Work

Currently, we are implementing a prototype of the revised GloServ system using Protege [10] and Racer [12]. Protege is an open-source development environment for ontologies and knowledge-based systems. The OWL Plugin is an extension of Protege that supports OWL. The Protege OWL Plugin provides a user-friendly environment to edit and visualize OWL classes and properties. It also has a graphical user interface that allows users to define logical class characteristics in OWL and execute description logic reasoners such as Racer. Protege's flexible architecture makes it easy to configure and extend the tool. Protege has an open-source Java API for the development of custom-tailored user interface components or arbitrary Semantic Web services.

The initial phase of the GloServ implementation concentrates on establishing service discovery with the core components of GloServ using a few specific ontologies. Registration and querying performance will be tested on the initial system once it is complete. The second phase of the GloServ design will look into the creating extensions for accessing services, having a service rating system, policy establishment, and security guidelines.

8 Conclusion

We have described a redesign of GloServ, a global service discovery architecture, using the OWL Web Ontology Language. GloServ functions both on an wide area as well as a local area network. It applies to a broad range of services that are defined flexibly using OWL ontologies. GloServ server distribution is specified in ontologies which allows for greater versatility. Service registration and querying for services also becomes automated and user-friendly due to the use of ontologies.

9 Acknowledgement

This work is supported by a grant from Nokia Research Center.

References

[1] OWL http://www.w3.org/2004/OWL/.

[2] North American Industry Classification System http://www.naics.com.

[3] Internet Corporation for Assigned Names and Numbers http://www.icann.org.

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

[5] D. Brickly and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3c proposed recommendation, World Wide Web Consortium, Feb. 2004.

[6] O. W. L. S. Coalition. OWL-S: semantic markup for web services. 2004.

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

[8] R. Fikes, P. Hayes, and I. Horrocks. OWL-QL - a language for deductive query answering on the semantic web. Technical report, Stanford University, 2003.

[9] U. Forum. UPnP device architecture 1.0. Technical report, Dec. 2003.

[10] J. Gennari, M. 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.

[11] E. Guttman, C. E. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, Internet Engineering Task Force, June 1999.

[12] V. Haarslev and R. Moller. Racer user's guide and reference manual version 1.7.19. Concordia University, Tehcnical Universityh of Hamburg-Harburg, University of Hamburg, 2004.

[13] S. Microsystems. Jini architectural overview. Technical report, 1999.

[14] D. Pavel and D. Trossen. Context provisioning and sip events. In *Workshop on Context Awareness at 2nd Conference on Mobile Systems, Applications, and Services (MobiSys)*, Boston, MA, USA, June 2004.

[15] J. Rosenberg and H. Schulzrinne. Internet telephony gateway location. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 488–496, San Francisco, California, March/April 1998.