CONTEXT BASED SYSTEMS ARCHITECTURE

Alexandru PETCULESCU[1]

Some of the most dynamic systems being built today consist of physically mobile hosts and logically mobile agents. Such systems exhibit frequent configuration changes and a great deal of resource variability. ln this article we present the idea used by context-aware systems, continuing with a practical example - a semantic matching algorithm. Finally we present results obtained in the implementation of this algorithm and our conclusions.

1. Introduction

The current era in computer science is an era of integration where users are allowed to access different resources and applications anyplace, anytime under no constraints. This diffusion requires an appropriate support in terms of application platforms that adapt to different scenarios and supports application mobility and increased complexity.

In this dynamic environment the software development increases in complexity as the setting is more open and dynamic compared to the traditional environment. In these terms the applications need an opportunistic behavior which is highly adaptive and dependent on resource availability, resources that may be transient in nature. The trend in software development for this environment is not to eliminate the complexity and burden, but to reduce this complexity by shifting as much logic as possible to the support infrastructure. Here a high abstraction level will provide the developer increased power in code development and solution architecture.

The term of context-aware computing refers to a system that is capable to explicitly detect and adapt to environment changes, e.g. an interruption of a video service due to network failure, low battery exception, or appearance in the network of certain resources. Most of the current facilities that support context-awareness regard simple measurements like temperature or location that are measured by host sensors. But there are times when the applications need to go beyond the basics and then the programmer needs to include more complex processes including communication and discovery. In the traditional network concepts the connection persists extended periods of time while in the ad-hoc network the frequent disconnections implies a significant increase in the

[1] PhD Student, Computer Science Department, University POLITEHNICA of Bucharest, Romania, e-mail: alexp@microsoft.com

programming effort. The paradox is that the mobile devices need access to a broader area of resources than common systems, making the problem of acquiring and maintaining connection to resources a problem of great interest. In a static paradigm a programmer simply declares the resource that the application needs. For the mobile paradigm we need a solution that maintains the access to specific resources despite environment's rapid changes caused by mobility, software components' migration and connectivity changes. An example would be a device that needs access to a printer and as the user moves a printer should be available as any printer is available in wireless coverage. Today this task may be fulfilled, but the coding of this workflow cannot be easily implemented as in traditional paradigm. In fact for a developer to implement this scenario he should deal with constant discovering of the wireless neighbors while maintaining existing connections. Our idea is that we may implement this at a middleware level that would make transparent to the user the process of identifying and maintaining access to the resources declared in the application.

In this article, after a brief introduction of dynamic environments we will address the problems rising in mobile and dynamic applications and all the necessary measures that need to be fulfilled in order to have an adequate environment dedicated to running this type of applications. The rest of the article will present a semantic matching algorithm used in context-aware applications and the results obtained by implementing this algorithm.

2. Context-Aware middleware requirements

In comparison to conventional applications the mobile paradigm applied to computing environment arise new issues that make service availability a critical problem in application development. Mobility is the key in this new paradigm here both in terms of users and devices. Users should be able to access their applications from any point even while on the move roaming between access points maintaining continuous connectivity. In the mobile paradigm network disconnections are frequent for users' devices as long as they are both voluntary in order to preserve battery power or costs and involuntary while on the run as wireless connectivity is lost. Another problem arising is the degree of heterogeneity in terms of devices – screen resolution, computing power, operating system, memory – and network technologies.

The most important issue in pervasive computing is the challenge in distributed resource retrieving and operating and destroys some assumptions of the traditional service based scenarios. This impact is created by a new meaning of the notion of context. There have been some definitions for the term of context [1]. In this paper from now on the term context will refer any information used for

describing the state or the activity of an entity or the environment where it operates.

The traditional middleware is independent in term of resource availability from user location and device properties as it relies on a static context. Changes in a static environment in terms of resources are small, predictable and rare. The traditional middleware systems are designed for a static context making transparent to the user the low level details of the network and provide applications a transparent view of the execution platform. In this new paradigm the context is very dynamic and can't be foreseen as context variations are frequent.

Thus supporting mobile applications in dynamic environments becomes an important problem as it requires to present information about location, system level data as device properties, environment conditions etc. all of them creating the before mentioned context. The logic of the context behavior should be clearly separated from the service logic reducing the complexity of the development of the services that are part from the mobile paradigm.

These considerations create the need for designing a new middleware platform that supports context driven applications. The key features here are:

- The application management layer should interact with underlying layers in order to retrieve information that helps decision taking mechanisms based on user's context and device characteristics. This layer interaction should enable the middleware to dynamically collect, represent and process context driven information and push it to the application level.

- This level of the middleware dedicated to context driven information processing should not interfere nor modify the application logic.

3. Current work in context modeling

In the last few years context information management systems came into researchers focus in order to create suitable models for them. The first attempts were directed towards context modeling for particular applications or classes of applications but soon they discovered that a better alternative is to direct their efforts towards a generic approach for a context provisioning model so that it would fit many different applications. The first approaches have been fulfilled in order to obtain a model that would get to an agreement in terms of common information that has to be collected as location, identity and time, but the term

context is still subject to different interpretations and one purpose is to define this concept as generic as possible leaving applications the possibility of extending and refining the concept of context accordingly.

The definition for the right model of context representation is an important step in the designing of context-aware application. Context-aware term first appeared in literature in [1] describing the context as location, identity information of persons in vicinity together with objects and their changes. Another definition worth looking over is given by the same authors: "any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves" [2]. Of course there are several others definitions in literature but the one described earlier are the ones that we will take into account in this thesis.

Below we will explore the most relevant approaches to context modeling and we will classify them conforming to the data structures used for representing the context correlated with the corresponding system.

- **Key-Value.** This model is the simplest data structure used for keeping context information. Schilit et al. [3] used this model to provide context information as values to an environment variable of an application. This model is used in many implementations of service discovery frameworks where the functionalities are described as a list of attribute-value pairs used in the discovery procedure. An example of a framework that uses this example is Jini [4]. The advantage of the key-value model is that is easy to manage but the disadvantage is that it has no capabilities to model a more complex structure needed by efficient context collecting algorithms.

- **Markup scheme.** Markup scheme models are modeled as a hierarchical data structure by using markup tags for the attributes. The content kept by markup tags is defined by other markup tags recursively. Typical way of utilizing markup scheme model is to describe metadata by using eXtensible Markup Language (XML). XML is a textual language used to describe arbitrary data structures in machine-readable form by utilizing documents called XML Schemas. By using XML Schemas it is possible to pre-determine the format, vocabulary and the structure of an XML document. Consequently, this forces the designers of an application, which utilizes the XML

Schema, to comply with the requirements of the given XML Schema. Hence, all the documents complying with the XML Schema are unambiguous to both machine and human readers. XML Schemas are well-suited in the mobile service landscapes due to their computational lightness.

- **Graphical model.** A well-known modeling tool is the Unified Modeling Language (UML) together with is graphical interface – UML diagrams. The UML is a suitable tool for modeling the context information because of its generic structure. Its best example in defining an object based on its context is the approach used in Entity-Relationship(ER) diagrams used in database representation diagrams making it easy for representing an entity and keeping the representation in a relational database.

- **Object-oriented model.** This model uses concepts from object oriented programming as encapsulation and reusability in the concept of context modeling. The idea is to rely on an abstract definition of an object and encapsulate the details about information collection and processing and hide it from the application level and offering only relevant information using interfaces. An example here is the TEA project [5] and their concept of cues that offers an abstraction of sensors: a function that takes a sensor as an input and returns the symbolic representation of a particular context.

- **Logic-based models.** Uses facts, expressions and rules in order to represent the context. This model uses the logic in order to represent the conditions that takes initial facts and expressions and applies the logical conditions to get to a derived set of expressions and facts. These logical conditions are in fact a set of rules applied by logical expressions. Logic based models have in common a formalization in context processing and representation. An example from this category is the GAIA framework [6] using first-order predicate logic in order to formally represent the context. There are different logic types inferred in this model – other solutions adopting fuzzy logic for representing the context.

- **Ontology-based models.** Ontology is "a formalization of a conceptualization" according to Gruber [7]. Ontologies allow context description using explicit formalisms. There is a new approach in context representation using semantic based context as semantics allows knowledge sharing and reusing. As an example is the CONON approach used by Wang et al. [8].

4. Context-awareness based on semantic metadata

The solution for supporting context based application and to permit correct application management is the use of metadata in implementing the model for context information representation and the decisions in the application behavior having clearly separated the management layer from the logic layer of the application. Metadata is used in describing the context information of a system and its structure together with the management operations abstracted.

Metadata effectiveness is based on the language used for implementing metadata specifications and the environment that act as running platform for the metadata. Metadata specification should use declarative languages to ease the understanding of users of different levels, to reduce the overhead spent in metadata reusing and modification and to easily highlight potential conflicts and inconsistencies. Metadata runtime is responsible for metadata maintenance and policy operations independent from application logic. In the sections below we will look over the existing models for metadata specification together with examples of current middleware platforms that use metadata to acquire context-aware adaptation of running applications.

Considering these conclusions the following guidelines for designing a new middleware platform for mobile devices emerged:

- **Context-awareness support.** This new middleware should offer context information to support its representation and management. The middleware needs to push context information to the application level and to provide adaption strategies based on this context.

- **Semantic metadata support.** A type of metadata defined in an unambiguous manner making it easy to be interpreted by a computer. This middleware should reason about metadata and the entities it describes and take the appropriate decisions based on the information provided.

5. Mobile Services Discovery

In a mobile environment a user that joins the network is able to access all the resources available in that network anytime, anyplace accordingly to the rights he is granted in the network and using all the connectivity options provided by the mobile device. Due to this assumption that a user is allowed to access public services when join a network it means that we need a mechanism that allows dynamic discovery of the services a user needs to fulfill his goals, minimizing the costs of service discovery, binding and configuration in terms of user's involvement, discovery based on users context e.g. his geographical position. Because we are talking about a mobile environment the service discovery task becomes a complex task due to terminal heterogeneity, service availability which may change frequently and environment conditions change quite often. The services that a user needs to discover in order to fulfill their goals are not known beforehand and the providers are supposed to promote their services to users that held devices with different technical capabilities not known until the interaction time. More than that, service developers cannot know and code all the existing configurations a device can have at access time – and here we include data format and discovery protocol.

In order to attain a supportive solution for pervasive environments we have to work in the area of service discovery and retrieval. The discovery model needs to move from network oriented discovery to context oriented discovery. In this new mobile concept the old model based on network infrastructure or administrative domain isn't versatile enough to allow a proper definition of discovery boundaries and an automatic selection of the proper service. The central piece of the puzzle now is the user and the user's context. The discovery solution we are looking for should take advantage of the user's context in order to retrieve the services list and select the proper service for the task. This would bring great benefit in users experience as it would save time and effort in service discovery and selection as the searching would be constrained to certain scopes based on user context.

As no assumptions can be taken for good about a user's context and services operate in an open and dynamic environment the need of semantic language adoption becomes a crucial point. The first advantage brought by the use of semantics is the formal representation of context and service properties in an abstract manner. Another advantage is automated interpretation of the context and service representation. More than that allows interoperability of initially unknown entities. Service retrieval could also benefit from the use of semantics as traditional queries are likely to fail in pervasive environments as a user doesn't have all the identifiers needed in order to call a service. Discovery matching based on service attributes isn't suitable here as it relies on exact keyword matching.

Our approach in this perspective is to use metadata in order to represent the properties of the entities in a mobile network and automatically infer requested capabilities compared to offered capabilities of a service. The automatic inferring is based on semantic matching algorithm as described in next chapter.

6. Semantic matching algorithm

In this section the algorithm used for semantic matching between user requests and service profiles is presented.

The figure 1 is a pseudo code representation of the semantic matching algorithm that has as inputs a required and an offered capability and returns the compatibility degree in terms of semantics. A capability is described by its properties. By offered capability we understand individuals – class instances – and by requested capability we understand a class explained by restrictions and service properties restrictions are user defined values. The algorithm iterates through capabilities collection taking into account each required capability and consider its relation with a compatible offered capability. There are three possible cases of relations between an offered capability and a required capability: the case when the offered capability is an exact instance of the required capability (is class), the case where the offered capability is a superclass of the required capability (is superclass), and the case where the offered capability is a subclass of the offered capability (is subclass) [9]. These relations are computed by reasoning about the values and the class types of offered and required capabilities. If we have an exact match for all capabilities then there is full capability between the user's request and the service requested. If we don't have an exact match, than we evaluate the compatibility based on preferences. We consider subclass and superclass cases compatible if there is a user preference that states that a certain constraint may be relaxed over that particular property.

Let's now evaluate an example: we consider V the vocabulary that describes the service ontology expressed in OWL-DL (SHOIN(D)). Then we take C as a class in V that describes a concept capability. C' is a concept that is subsumed by C in the ontology interpretation. C" is a concept that C is subsumed by. We consider now a user request having a capability that is specified by a set of restrictions contained by C. We now have the following situations:

- **Case is class.** This is the easiest case when the offered capability is a capability of type C. So both the offered capability and required capability are instances of the same class C and we compare the values of the characterizing properties of the offered capability against the values of the requested capability in order to fulfill the restrictions.

- **Case is subclass.** The offered capability is capability of type C'. In this case the required capability is of a type that is a super class of the type of the offered capability. In our case the values of the properties that characterize the offered properties satisfy the restrictions of the required capability. So in the case where the offer is more specific than the request usually it offers the same properties as the superclass and some specific properties. But it may be the case where some properties from the superclass are not defined in the subclass as they may have no meaning in the subclass (E.g. We have a Vehicle class and a Rocket subclass; Number of wheels property in Vehicle class has no meaning for Rocket though it is not defined). In this situation the algorithm behavior depends on the user preferences. If the property is relaxed in terms of priority or is optional than it is set as compatible with the request.

- **Case is superclass.** The offered capability is capability of type C". As C' is a superclass which is more generic than the requested capability, any property constrained in the request that exists in the superclass will have an assigned value in the offered capability.

The use of priorities in requested capability definition is used in calculating the degree of compatibility between offered and requested compatibilities. Another use of the compatibility may be to establish the order used for checking compatibility for requested/offered capabilities. This prioritization may be consider among the key features of MASF as it allows service properties evaluation based in the importance assumed in the request. This feature allows a flexible filtering of the services in the process of discovering the personalized view of the available services corresponding to a particular user.

```
Is OFFERED_CAP type of  REQUESTED_CAP?
if (is class)
{
        foreach REQUESTED in CAP_PROPERTIES
        {
                1. Identify OFFERED
                // find corresponding offered capability  for the requested one
                2. Does OFFERED satisfy REQUESTED?
                if (true)
                        success match for REQUESTED
                else
                        set compatibility level  to REQUESTED
         }
}
if (is superclass)
{
        foreach REQUESTED in CAP_PROPERTIES
        {
                // super-class restrictions
                1. Identify OFFERED
                2. Is OFFERED an instance of  REQUESTED_RESTRICT or  superclass ?
                if (superclass)
                        i. create restriction that includes REQUESTED
                        ii. use it as REQUESTED restriction
                if (class)
                        same as class case
                3. Does OFFERED satisfy REQUESTED?
                if (true)
                        superclass success for REQUESTED
                else
                        set compatibility level REQUESTED
        }
}
If (is subclass)
{
        foreach REQUESTED in CAP_PROPERTIES
        {
                 //sub-class restrictions
                1. Identify OFFERED
                2. Is OFFERED an instance of REQUESTED_RESTRICT or subclass?
                        if (subclass)
                         {
                        // check restriction based on sub-property and check if restriction is included in range
                                if (true)
                                        i. create restriction included by REQUESTED
                                        ii. use it as REQUESTEDrestriction
                                else
                                        return failure for REQUESTED
                         }
                        if (class)
                                same as class case
                3. Does OFFERED satisfy REQUESTED?
                        if (true)
                                set subclass success for REQUESTED
                        else
                                apply preference to REQUESTED
}
else
        return failure
```

Fig. 1 Semantic matching algorithm

7. Evaluation

The following practical examples were for testing purposes only without being used in a real commercial scenario. Its purpose was to test and evaluate the practical use and performance of a service discovery scenario based on semantic matching algorithm.

Scenario

Tested scenario simulates an event registry system offering support for registration at different types of events in a commercial area: restaurants, movies, theaters, concerts.

Assuming that a user Alex that was authenticated in the mobile network wants to make a reservation at a restaurant he starts discovering available reservation services based on his requirements. These requirements are provided in a semantic form as in figure 2. Based on the user request the semantic processor runs the semantic algorithm in order to create the list of the services that qualify for the user request. Let's assume that Alex wants to make a reservation at a restaurant. The initial scope includes all the restaurants in Alex's vicinity, but in the interval 13-14 some of them are closed. Alex wants to make a reservation at 13:30. One of the requirements in Alex's request is the interval that he wants to use for reservation, expressed in the semantic form. The matching algorithm will exclude from the retrieved list the restaurants that are closed in the interval 13-14. If Alex changes the requirements for reservation to start at 14:00 then these restaurants won't be excluded from the list. The dynamic behavior explained is shown in figure 3.

Performance evaluation

The testing environment used for the comparison of traditional web service vs. our idea was a virtual lab containing a Microsoft Windows 2008 server with IIS 7.0 and .NET Framework 3.5. The web services and the semantic algorithm were implemented using ASP.NET technology and the development language used was C#.

We simulated an environment conforming to the environment described above and we used as variables the number of services available as well as the number of semantic requirements used for reasoning and we measured the overhead introduced by the semantic matching algorithm. Based on the whole set of operations – including semantic matching, service invocation, gathering the results – the semantic matching algorithm takes about 5% of the entire time. Of course, variations in connectivity conditions may affect significantly the results. As you can see in the results presented in table 1 we have approximately 9 ms for

a semantic evaluation over 100 services for one requirement and around 11 ms for 4 requirements over the same test sample.

```
<base_p:service rdf:ID="RestaurantReservation">
 <base_p:profile rdf:ID="Restaurant_Profile">
  <base_p:id>
   <n:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Moods Restaurant
   </n:name>
   <n:location rdf:resource="&locate-ont;MallSecondFloor"/>
  </base_p:id>
  <base_p:requirement_s>
   <io_cap:Endpoint rdf:ID="OutputReq">
    <base_p:requires>
     <device_cap:DeviceCap rdf:resource="&device-ont;GPS"/>
    </base_p:requires>
   </io_cap:Endpoint>
  </base_p:requirement_s>
  <base_p:requirement_d>
   <base_p:DynamicReq rdf:ID="DynamicReq">
    <base_p:requirementCond>
     ...
    </base_p:requirementCond>
    <base_p:condition>
     ...
    </base_p:condition>
   </base_p:DynamicReq>
  </base_p:requirement_d>
  <base_p:capability_s>
   <io:Endpoint rdf:ID="OutputCap">
    <io:outFormat rdf:resource="&format-ont;PlainText"/>
    <io:outFormat rdf:resource="&format-ont;HTML"/>
   </io:Endpoint>
  </base_p:capability_s>
  <profile:capability_d>
   <profile:DynamicCap rdf:ID="DynamicCap">
    <profile:capabilityCond>
     <client_cap:ReservationCap rdf:ID="ReservationCap">
      <client_cap:bookedEntity rdf:resource="&reserv-ont;Table"/>
     </client_cap:ReservationCap>
    </profile:capabilityCond>
    <profile:condition>
     <time:TimeCond rdf:ID="Condition_1">
      <time:startTime rdf:resource="&time-ont;09:00"/>
      <time:endTime rdf:resource="&time-ont;17:00"/>
     </time:TimeCond>
    </profile:condition>
   </profile:DynamicCap>
  </profile:capability_d>
 </base_p:profile>
</base_p:service>

Legend:
    • Identification
    • Requirements
    • Capabilities
```

Fig. 2 Semantic matching algorithm

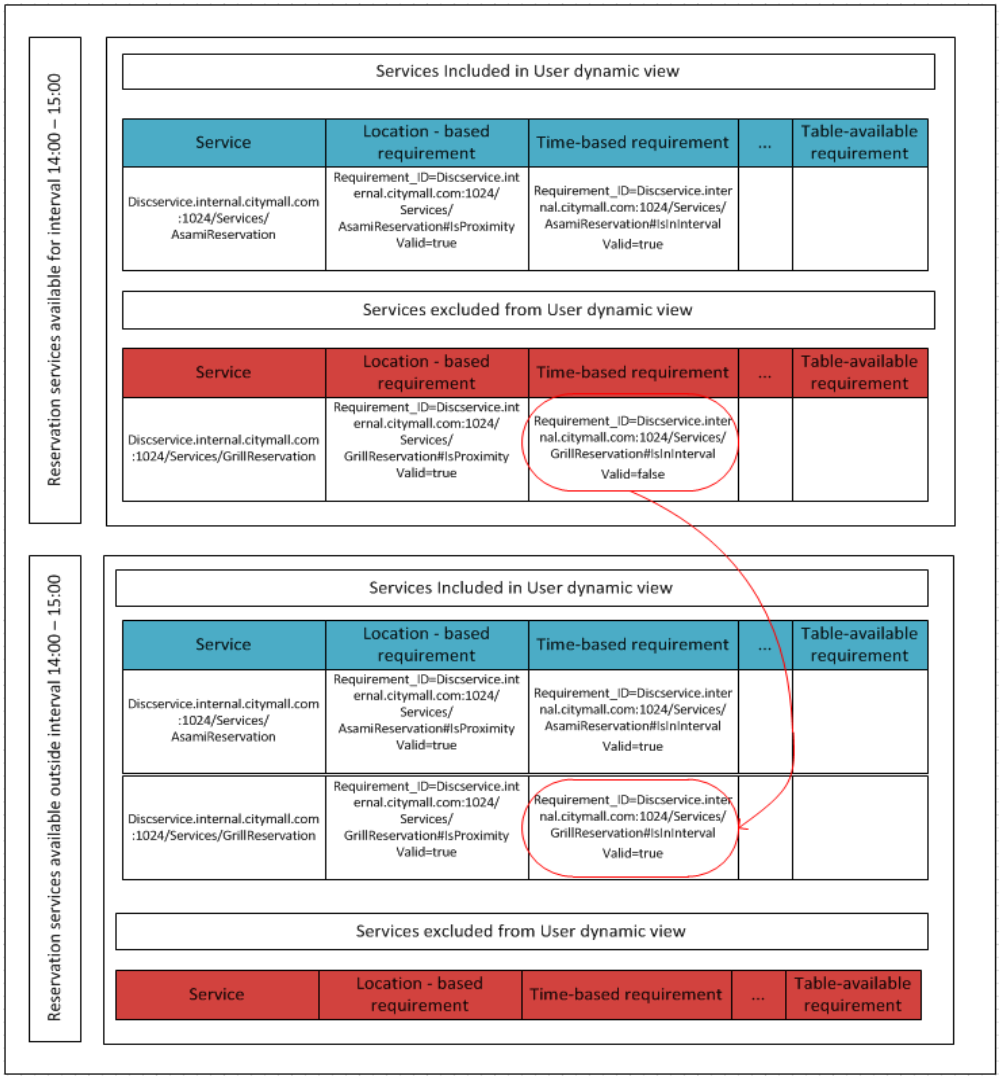

Fig. 3 Dynamic evaluation of requirements

Table 1

Semantic matching algorithm time performance

Semantic Matching Time (ms)			
Nr. of requirements	Nr. of service instances		
	33	66	99
1	7,433	7,832	9,133
2	7,624	8,432	9,624
3	7,989	9,123	10,823
4	8,112	9,155	10,921

8. Conclusion

As the mobile environments become more and more important in our daily life, new technologies emerges in order to accommodate this need for mobility.

What I presented in this paper is a semantic service selection algorithm that may be used in environments where providers and consumers know as little as possible about one another and they need a way in which they may communicate and discover one another.

As future improvements we have identified two directions that need investigating and these are respectively:

- A heuristic-based technique that will improve matching response time by evaluating only best matched services available and sacrificing completeness.

- Take into account the security risks that arise in mobile scenarios between unknown entities – a protection mechanism for protecting users and controlling service access.

REFERENCES

[1] *Anind K. Dey, Gregory Abowd, and Daniel Salber.* "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications." Human Computer Interaction (HCI) Journal, 2001.

[2] *Anind K. Dey.* "Understanding and using context." Personal and Ubiquitous Computing, 2001.

[3] *William N. Schilit.* "A System Architecture for Context-Aware Mobile Computing." PhD thesis, Columbia University, 1995.

[4] Jini Technology. http://www.jini.org/.

[5] *Albrecht Schmidt and Kristof van Laerhoven.* "How to build smart appliances." IEEE Personal Communications.

[6] *Anand Ranganathan and Roy H. Campbell.* "A middleware for context-aware agents in ubiquitous computing environments." In Endler and Schmidt

[7] *Thomas R. Gruber.* "A translation approach to portable ontology specifications." Knowledge Acquisition, 1993

[8] *Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung.* "Ontology based context modeling and reasoning using owl." In PERCOMW, 2004

[9] *Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara.* "Semantic matching of web services capabilities." In I. Horrocks and J. A. Hendler. The Semantic Web - ISWC 2002, page 333 - 347.