# Context-aware Specialization of Semantic Rules for choosing Services in Pervasive Environments

Noha Ibrahim, Frédéric Le Mouël

ARES INRIA / CITI, INSA-Lyon, F-69621, France
{noha.ibrahim, frederic.le-mouel}@insa-lyon.fr

***Abstract*— The development of many highly dynamic environments, such as pervasive environments, has modified the behavior of users and consequently, their expectations of systems and applications executed in these environments. Thus, a user can connect to different places, and, each time, would like to use the functionalities offered by the physically or logically close environment. This paper proposes a novel approach for specializing strategies for choosing, and then using services in a pervasive environment. To the best of our knowledge, strategies for the use of services in a pervasive environment are dynamic but ad-hocly defined during the development phase. By changing the context, the strategy can change and be replaced by another one predefined in the application development phase. Our idea is to provide a context-aware mechanism which is able to adapts dynamically the condition and action parts of the rule of the strategy to the context without the need to specify how, during the development of applications. Only a general strategy is defined in the development phase. At run time, a dynamic specialization adapted to the context, provides a specialized strategy from the general one[1].**

## I. Introduction

Pervasive environment is becoming more than a vision and is closer to users' everyday reality. Devices tend to be more and more tiny and even invisible. These devices are embedded in almost any type of imaginable objects such as appliances and various consumer goods. They are also more and more mobile and thus, can move from a context to another one. The diversity of services offered by these devices, and their flexible availability can be an issue for the user to accomplish a specific task. Users would like to, whenever it is possible, use services available everywhere in the environment, without the permanent need to be aware of their context and of the services in it. In this paper, we propose a novel approach for specializing strategies for choosing, and then using services in a pervasive environment. Usually in other approaches, strategies for the use of services in a pervasive environment are dynamic but ad-hocly defined during the development phase. In these approaches, by changing the context, the strategy can change and be replaced by another one predefined. The granularity of the adaptation is the strategy; these approaches replace a strategy by another one. The users or applications select the most appropriate substitution mechanism with respect to its requirements by providing various substitution strategies. Our idea is to provide a context-aware mechanism that adapts dynamically the rules of the strategies rather than the strategies themselves. The mechanism will adapt dynamically the *condition* part and *action* part of the rule to the context without the need to specify how, during the development phase of applications. At this phase, only a general strategy is defined. At run time, a dynamic specialization provides a specialized strategy from the general one, adapted to the context.
The paper is organized as follows: section 2 gives an example scenario to explain the motivation of our work. Section 3 details the strategy specialization and the ontology used. Section 4 describes the implementation status. Section 5 lists some existing related works. Finally, section 6 concludes and gives future research directions.

[1]This work is part of the ongoing European project: IST Amigo-Ambient Intelligence for the Networked Home Environment [1].

## II. Motivation

### A. Application scenario: Perva-Home

Liz lives in a pervasive house. She has a special service that awakes her every morning to go to work, wherever she is. This service works in all the rooms of the house. If Liz is in her room when it is 8 o'clock, the radio turns on and Liz is awoken listening to her favorite radio station. If Liz is sleeping in her child room, she is awoken by the HiFi stereo playing songs for children. If she is sleeping in the guest room, she is awoken by the light switching on. The service adapts to the environment in which Liz is present. The strategy *'in the morning & at home, awake Liz'* is adapted and generated dynamically according to the context where Liz is. If Liz goes to her house in Paris, the service *'in the morning & at home, awake Liz'* will awake her in the morning by ringing the phone in her room. To do so, first a search of the available services that can awake Liz is done. Once the appropriate service is found, the strategy uses this new service, here ringing the phone, to awake Liz.

### B. Context-Aware Strategies

In the scenario above, the general strategy is *'in the morning & at home, awake Liz'* which is specialized according to Liz' varying context. We define strategy as a set of rules. A rule has three parts, an *event* part, a *condition* part and an *action* part. In our example the *event* is *it is the morning*, the *condition* (set of *events*) is *'in the morning & at home'* and the *action* is *'awake Liz'*. The general strategy is specialized depending on the context, and the devices around Liz. This will be very useful in pervasive environments as a user can move from

one context to another while using his/her services everywhere without having to know what is available around him/her and how these new services work. Liz would like to be awoken all the working days of the week in the morning no matter where she decides to sleep: in her room, in her child room or even in the guest room. The notion of *home* includes all Liz's houses, whether they are in New York or Paris. She has set an initial service to do that using a general strategy and this service will take care of adapting the strategy to her context. Once she moves to her new environment, or once new devices are present in the environment, Liz does not have to search and find herself the services capable of awakening her in the morning. Nor does she need to know how the services/devices work.

## III. Dynamic Strategy Specialization in Pervasive Environment

In the Perva-Home scenario (cf. figure 1) the applied strategy is dynamically specialized from the general strategy. General strategies are a set of rules such as: *'In the morning & at home, awake Liz'*. Rules are semantically described. For example, the Perva-Home finds the users location and based on this location the general semantic strategy will be specialized with a specialized condition *When it is 8 o'clock at Paris & Liz is in her room* and a specialized action *start radio*. The different layers described in figure 1 are abstract. The devices layer contains the devices present in the house. The services these devices offer are present in the services layer. The strategy applied is described in the strategies layer. This strategy executes the services depending on the location of the devices.

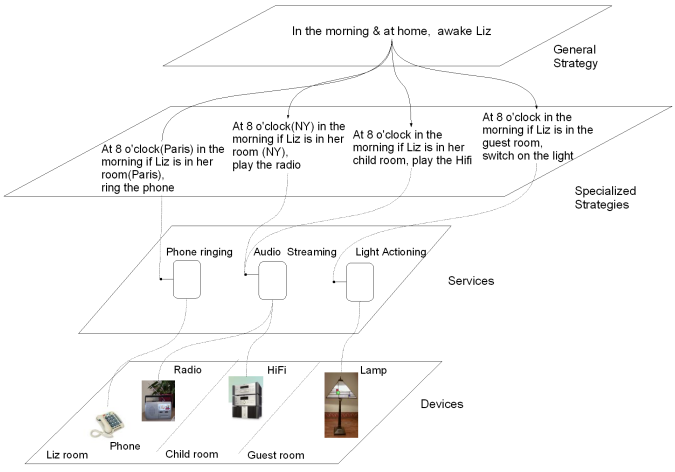


Fig. 1. Perva-Home

In the following, we define our service and context models and present our context-aware ontology related to the Perva-Home scenario. Also, the generation of the specialized strategy is explained. All along we illustrate our specialization proposal with the Perva-Home scenario.

### A. Service Model

Our service publishes interfaces: an interface specifies methods that can be performed on the service. Our service's interfaces are public and so published for an external use. Our service model is independent of any implementations and can be applied to OSGi bundles/services [2] or Web services [3]. A service profile provides the description of the interfaces the service publishes. Standard description languages such as WSDL [4], UDDI/XML and OWL [5] can be used to illustrate these descriptions. We decide to use OWL-S (Semantic Markup for Web Services) [6] because of its ability to support specialization and inheritance.

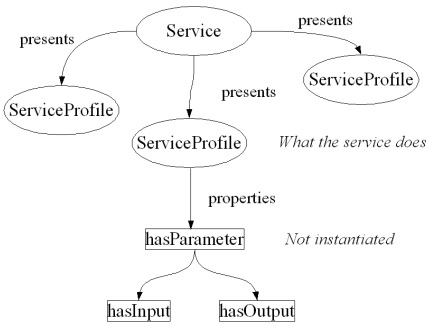


Fig. 2. Service ontology description

The class Service (cf. figure 2) provides an organizational point of reference for declaring our service. One instance of Service will exist for each distinct service available in the environment. The properties *presents* is a property of Service. The class ServiceProfile is the range of this property. Each instance of Service will present a descendant class of ServiceProfile. The service profile "describes what the service does", in other terms it describes its interfaces. A service can present several profiles corresponding to the several functionalities it can publish. For example, the service *Audio streaming* of the *Radio* device presents an entertainment profile for playing music to the user, but also an awaken profile for awakening the user in the morning. The ServiceProfile class has two properties: the hasInput which ranges over an Input instance, and the hasOutput which ranges over an output instance. All these classes and properties are provided by OWL-S [6]. A service's various properties are formally specified with OWL-S in terms of shared concepts in ontologies.

### B. Context Model

In order to clearly set the scope of information we want to deal with, we refer to the following general definition of context proposed in [7]:
*'Context is 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, the device and application themselves'*.
We have developed a context model based on predicates. The

adopted convention has the form of *predicate(subject, value)*, in which

- *predicate* is the type of context that is being described (location, time, temperature).
- *subject* is the subject name (person name, object).
- *value* is the value of the subject (bathroom, 8 o'clock, 120 degree)

For example, *Location(Liz, Parent room)* - Liz is located in her room, *Time(New York, 08:00GMT)* - it is 8 o'clock in New York. The structure and properties of our context predicates are described in an ontology (cf. figure 3 & figure 4). Each statement is in the form of *predicate(subject, value)*, where *subject* and *value* are ontology's objects or individuals, and *predicate* is a property relation defined by the ontology.
We detail a part of the `Time` ontology (cf. figure 3), `Location` ontology (cf. figure 4) and `Action` ontology (cf. figure 5) that correspond to the Perva-Home scenario. The `Time` ontology describes semantically the notion of *morning* depending on the location. The morning in the USA is 08:00GMT in New York, which is 15:00GMT in Paris. While the morning in France is 08:00GMT in Paris (01:00GMT in New York).

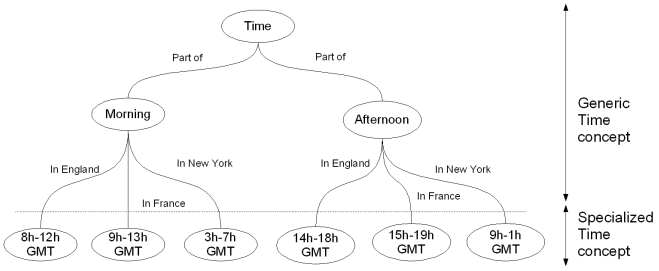


Fig. 3. Time Ontology

The `Location` ontology (cf. figure 4) describes the places Liz can occupy. No matter she is in New York or Paris Liz can be in her room at her house or in the conference room at her company.

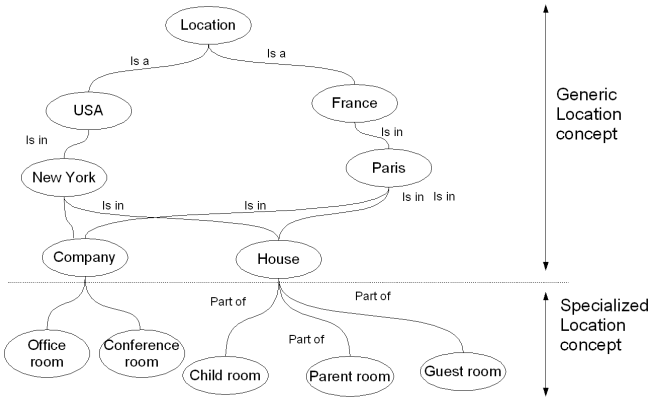


Fig. 4. Location Ontology

The `Action` ontology (cf. figure 5) describes the different actions that can be taken to awake Liz. The choice of the action depends strongly on the devices present in the context of the user, here Liz.

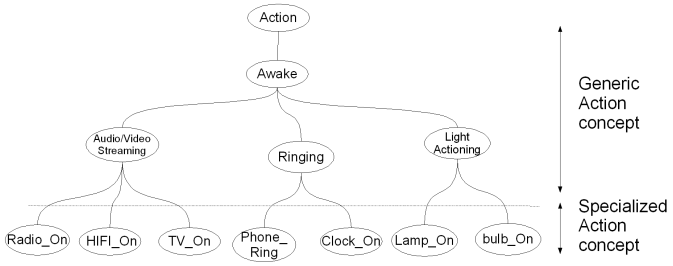


Fig. 5. Action Ontology

Context information is represented using the Resource Description Framework (RDF). RDF is a language that has originally been designed for representing information about resources in the World Wide Web (RDF-concepts). In our modeling approach, a piece of context information is an RDF fragment which relates entities (that are instances of concepts or classes from the context ontologies) to other entities or values.

We describe in RDF/XML notation (cf. figure 6) the condition that stipulates that Liz is in her child room and it is the morning in New York.

```
<?xml version= «1.0 »?>
<rdf:RDF
    xmlns:rdf= « http://www.w3.org/1999/02/22-rdf-syntax-ns# »
    xmlns:rdf= « http://www.w3.org/2000/01/rdf-schema# »
    xmlns:rdf= « http://www.w3.org/2002/07/owl# »
    xmlns:rdf= « http://www.owl-ontologies.com/unnamed.owl #»
  xml:base= « http://www.owl-ontologies.com/unnamed.owl » >
  <User rdf:ID= « Liz »>
    <Location>
      <Room rdf:ID= « Child Room »/>
    </Location>
  </User>
  <User rdf:ID= « New York»>
    <Time>
      <time rdf:ID= «8:00GMT »/>
    </Time>
  </User>
</rdf:RDF>
```

Fig. 6. A context ontology example written in RDF/XML notation

The ontology is used to check the validity of context predicates. It also makes it easier to write different context predicates since we know the structure of the predicate and the types of values different arguments can take. The defined ontologies will be used to define the condition and action parts of our strategies.

### C. Specialized strategies

A rule is formed of an *event* part, a *condition* part and an *action* part. The *event* part of the rule describes the context

information such as *it is the morning* or *Liz is in her room*. The *condition* part expresses the set of events the context has to verify so that our rule can be applied. The *action* is the action that will be executed when the condition is satisfied.

Once the context of the user is defined (time, location, services available), the general strategy that expresses the *condition* and *action* in semantic needs will specialize in a strategy, expressing *condition* as context predicates and *action* as a service profile output description. In the specialization phase, the *condition* and *action* are expressed using the ontologies defined section III-B.

The *event* part of the rule is provided by the Amigo Context Management Service (CMS) [8]. A functional module provides access to local context information. It can represent sensors that are placed in the house and in each room. This module returns the representations of the real world context using the context description ontology. Amigo CMS defines two essential modules: *Context Source* and *Context Broker*. *Context Source* is the abstraction for any functional element that is able to provide context information. The common element used is the data sensor: devices that measure environmental properties, softwares entities that give information about a device or services. The *Context Broker* is responsible for the discovery of and access to Context Sources.

The *Condition* part of the rule is the expression of the events in RDF/XML notation using concepts of the ontologies. In our Perva-Home scenario the context ontology used is the time and location ontologies (cf. figure 3, 4). The general condition is then specialized and *'in the morning & at home'* is expressed using context ontology predicates, e.g. Time(New York, 08:00GMT) and Location(Liz, Child room) and expressed in RDF/XML notation (cf. figure 6).
A matching algorithm is used to find the services that execute the *action* of the strategy. The algorithm matches between the service profile description being sought (the *action* (cf. figure 5)) and the semantic description of the functionalities provided by each service. To perform semantic operation matching, our algorithm is built upon the semantic matching of web service capabilities algorithm proposed by Paolucci et al. [9]. This algorithm is based on semantic match between a declarative description of the service being sought (the request service), and a description of the service being offered (the advertisement service). The services are described in DAML-S (DARPA Agent Markup Language for Web services). The matching engine of the algorithm is based on DAML ontologies, and support flexible semantic matching. The degree of flexibility can be defined allowing flexibility in the choice of services. An advertisement service matches a request service when all the outputs of the request are matched by the outputs of the advertisement, and all the inputs of the advertisement are matched by the inputs of the request. This describes the perfect match between services. The algorithm defines a degree of match to allow more flexibility. The degree of match between two outputs (or two inputs) depends by the relation between the concepts associated with those outputs in the ontologies. The degree of match is determined by the minimal distance between concepts in the taxonomy trees. Four degrees of match are defined: *exact* matching (if the outputs are equivalent), *plug-in* matching (if the output of the advertisement is a subClass of the output of the request), *subsumes* matching (if the output of the request is a subClass of the output of the advertisement) and the *fail* matching (no subsumption relation between outputs). Degrees of match are organized along a discrete scale in which exact matches are of course preferable to any another; plug-in matches are the next best level because the output returned can probably be used instead of what the requester expects. Subsumes is the third best level since the requirements of the requester are only partially satisfied: the advertised service can provide only some specific cases of what the requester desires. Fail is the lowest level and it represents an unacceptable result. We adapt the algorithm in a way such that, the *action* of our strategy is the output of the request service, and the services present in the context are the set of advertised ones. Our generic model of service respects the web services model and the OWL-S ontology language is originally the DAML-S used by Paolucci et al. [9]. Using our OWL ontologies (cf. figure 5) and concepts matching can lead to semantic recognition despite the possible syntactic differences between services interfaces. Indeed, a phone ringing service, an audio streaming service or a switching-on-a-light service have different methods signatures and interfaces definition but a common semantic profile description corresponding to the action *awake* (cf. figure 5). We show how service is matched to the action through the radio example. A simplified version of the OWL description of the Radio service is given figure 7. The outputs the service generates is instance of the concept audio streaming that is a sub class of the concept awake (cf. figure 5). The algorithm for output matching recognizes that *Radio_On* and *awake* are plug-in match because *Radio_On* is a subClass of *Audio Streaming* which is a subClass of *awake* (cf. figure 5).

```
<!-- Service -->
+ <owl:Class rdf:ID="Radio">
  <rdfs:label>playing radio</rdfs:label>
  </owl:Class>

<!-- ServiceProfile -->
+ <owl:Class rdf:ID="AwakeningService">
  <rdfs:label>switching on the radio</rdfs:label>
  </owl:Class>

<!-- OutputDescription -->
+  <owl:ObjectProperty rdf:ID="Radio_On">
   <rdfs:subPropertyOf rdf:resource="#AudioStreaming"/>
   </owl:ObjectProperty>
```

Fig. 7. The radio semantic description service written in OWL

In the Perva-Home scenario the general strategy : *'In the morning & at home, awake Liz'* will become:

- *IF Time(New York, 08:00GMT) U Location(Liz, Parent*

*room) THEN (Radio, Radio_On)*

- *IF Time(New York, 08:00GMT) U Location(Liz, Guest room) THEN (lamp, Lamp_On)*
- *IF Time(Paris, 08:00GMT) U Location(Liz, Parent room) THEN (Phone, Phone_Ring)*

The Perva-Home chooses the service that plays the radio as the service to execute in Liz room. The semantic description of this service fits the semantic description of the *awake* action. The execution is done by a method call to the service interface. When Liz is in the guest room the Perva-Home search for the services available in the new environment and having the semantic description `awake`. Liz will be awoken by the light switching on. In her room in Paris the strategy will be different (ringing the phone) and adapted to the new environment.
The strength of the semantic ontologies model is also in the possible equivalence between concepts. In our Perva-Home scenario, if Liz goes to Japan for professional reasons, she would like to be awoken in the morning like at her house in New York or Paris. The general strategy *'In the morning & at home, awake Liz'* must be also adapted in the hotel in Tokyo. The hotel and house concepts in our ontology should be treated as home for Liz concerning the awakening strategy.

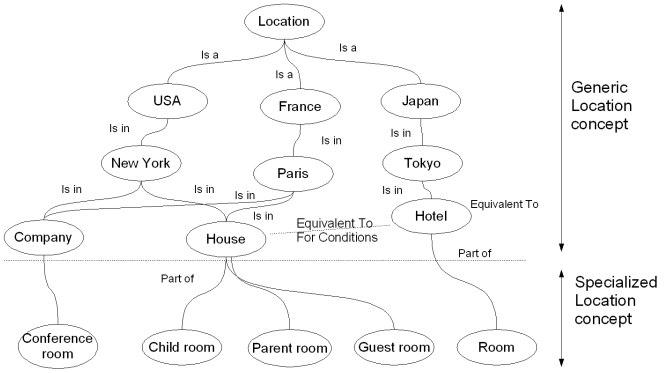


Fig. 8. Extended Location Ontology

The two concepts Hotel and House are tied together using a property equivalentClass to indicate that in a certain context and for some strategies, the hotel can be taken as home for the user. For example, in our strategy that awaken Liz every day, if Liz is in a hotel in Tokyo, she would like to be awoken, so the strategy also work in the hotel that is considered in our ontology (figure 8) equivalent to house.

## IV. Implementation

We demonstrated the feasibility of our specialization model by implementing a first prototype. We choose to implement it using Java platform and OSGi technology. The choice of Java was motivated by its portability and its capability to provide a strong separation between the APIs and their implementations. OSGi [10] was chosen for its facility to provide the Inversion Of Control (IOC) [11]. OSGi simplifies the development, deployment and management of services by decoupling the service's specification from its implementation. A simplified version of Amigo CMS [8] implements the context broker and context sources on OSGi to collect the context information and returns the *event* part of the strategy. The specialization is done following the `Strategy Pattern` [12] (cf. figure 9). The discovery process of the services in the context is not explained in this article. For now, we rely on existing system, UPnP [13].

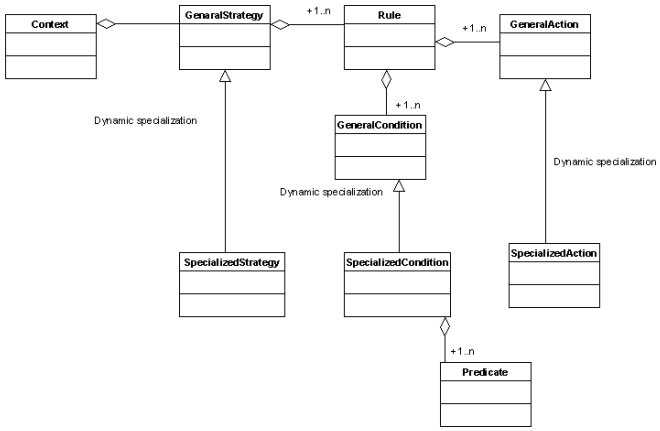


Fig. 9. Specialized Strategy generation

This work is integrated to the *ANIS: Automatic Negotiated Integration of Services* platform [14]. *ANIS* provides a framework for integrating services with different techniques of integration (composition, weaving, and deployment by downloading/uploading) and an OSGi Toolkit. The decision mechanism in *ANIS* will use the strategies specialization in order to decide which services to integrate, and how to integrate them, in a pervasive environment. The decision service of *ANIS* creates a general integration strategy for integrating services within applications. An example of a general strategy defines the *event* part: applications and services available in the context, the *condition* part: *when compatible services* and the *action* part: *integrate*. The *action* will be specialized in *compose*, *weave* or *deploy/upload* depending on the context information, the semantic services descriptions and matching algorithm results. Possible specialization can be: *'when semantic inputs of a service matches semantic outputs of another service, compose services'* or *'when semantic inputs of a service matches semantic inputs of another service, weave services'*. The specialization of integration strategies will be the subject of another article.

## V. Related work

Context-aware strategies for using services have emerged with the development of pervasive environments. Service-oriented platforms and Multi-Agent systems are the domains in which these concepts are most dealt with.
[15] proposes a middleware that facilitates the development of context-aware agents. This Middleware for Context-Aware Agents in Ubiquitous Computing Environments allows agents

to acquire contextual information easily, using about it different logics and then adapting themselves to changing contexts. The middleware allows autonomous, heterogeneous agents to have a common semantic understanding of contextual information. Ontologies are used to define different types of contextual information. Agents either use rules or machine learning approaches to decide their behavior in different contexts.

In service oriented platforms, existing approaches to identifying and choosing services can be classified into three cases [16]: Task based approach, choosing a service is based on the functionality the service provide with respect to a user task [17]; semantic web based approach, choosing a service is based on compatible concepts of the service's inputs, outputs, pre-conditions, and/or effects (IOPEs) with respect to semantic relations defined in ontologies [18], [19]; sub-typing based approach, choosing a service is based on matching the signatures (arguments and types) of the services with the respect to various sub-typing rules [20]. We will present two existing frameworks: XVMF and SOCAM.

XVMF [16] is an extensible and versatile matchmaking framework, which supports various service substitution mechanisms in dynamic application adaptation for ubiquitous mechanism. It allows an application or a user to select the most appropriate substitution mechanism with respect to its requirements by providing various substitution policies.

SOCAM [21], a service-oriented middleware for building context-aware services proposes a middleware for building and rapid prototyping of context-aware services. It provides support for acquiring, discovering, interpreting and accessing various contexts to build context-aware services. The context model is based on OWL. SOCAM is a distributed middleware that converts various physical spaces from which contexts are acquired into a semantic space where contexts can be easily shared and accessed by context-aware services.

Our approach combines all the three classified approaches, our task is defined by the general strategy, the semantic is used for its specialization and the sub-typing when calling the interfaces services. Our proposal is innovating not because it adapts the strategy to the context using a classic rule-based approach but because it introduces a dynamic, context-aware strategy specialization concept. The strategy is specialized depending on the context at runtime. Our approach uses existing works such as the Amigo CMS model and the semantic matching algorithm proposed by Paolucci et al. [9] and adapts them to our needs.

## VI. Conclusion and Perspectives

In this paper we have proposed a novel approach for specializing strategies for choosing, and then using services in a pervasive environment. Usually in other approaches, strategies for the use of services in a pervasive environment are dynamic but simply defined during the development phase. Our approach is to provide a context-aware mechanism which is able to adapt dynamically the *condition* part and *action* part of the strategy rule to the context without the need to specify how, during the development of applications. Only a general strategy is defined in the development phase. At run time, a dynamic specialization provides a specialized, context-aware rule from the general rule strategy. We define context predicates to model our specialized *condition* and ontologies concepts for our specialized `action`.

We are improving our rule-based engine in our prototype and also aim to test the performance of our prototype for instance, its reactivity to service availability in real pervasive environments by moving from simulated context to real sensed context (home lab).

## References


[1] N. Georgantas, Ed., *Specification of the Amigo Abstract Middleware Architecture*, ser. Delivrable D2.1, IST Amigo project, 2005.

[2] OSGIalliance, "About the OSGI service platform," OSGI alliance, Tech. Rep., 2004, revision 3.0.

[3] W. Iverson, *Real Web services*. O'Reilly, Oct. 2004.

[4] A. E. Walsh, *UDDI, SOAP and WSDL: the web services specification Reference book*. Pearson Education, Apr. 2002.

[5] W. Members, "OWL web ontology language," W3C, Tech. Rep., 2004.

[6] T. O. S. Coalition, *OWL-S: Semantic Markup for Web Services*, OWL Services Coalition, 2003, white paper.

[7] A. Dey, D. Salber, and G. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, vol. 16, pp. 97–166, 2001.

[8] M. D. Janse, Ed., *Report on Specification and Description of Interfaces and Services*, ser. Delivrable D4.1, IST Amigo project, 2005.

[9] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," *Lecture Notes in Computer Science*, 2002.

[10] O. Alliance, "OSGi Service Platform, Core Specification Release 4," Draft, 07 2005.

[11] B. Loritsch, "Developing with apache avalon," Apache Software Foundation, Tech. Rep., 2001.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*. ADDISON-WESLEY, 1995.

[13] M. Corporation, "Understanding UPnP : A white paper," UPnP Forum, Tech. Rep., 2000.

[14] N. Ibrahim and F. L. Mouël, "ANIS: A Negotiated Integration of Services in Distributed Environments," in *8th International Symposium on Distributed Objects and Applications (DOA)*, Oct. 2006, montpellier, France.

[15] A. Ranganathan and R. H. Campbell, "A middleware for context-aware agents in ubiquitous computing environments," in *Middleware Conference*, 2003, pp. 143–161.

[16] K. Lee, D. Lee, I. Park, and S. Han, "XVMF: An Extensible and Versatile Matchmaking Framework for Supporting Dynamic Application Adaptation in Ubiquitous Computing Environments," in *Proceedings of Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops(PERCOMW'06)*. IEEE Computer Society, 13-17 March 2006.

[17] J. P. Sousa, "Scaling task management in space and time: Reducing user overhead in ubiquitous-computing environments," Ph.D. dissertation, Carnegie Mellon University, 2005.

[18] S. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," in *IEEE Intelligent Systems*, 2001, pp. 46–53.

[19] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing Semantics to Web Services: The OWL-S Approach," in *Proceedings of First International Workshop on Semantic Web Services and Web Process Composition(SWSWPC 2004)*, 2004.

[20] A. Zaremski, "Signature and specification matching," Ph.D. dissertation, Carnegie Mellon University, 1996.

[21] T. Gu, H. keng Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, vol. 28, pp. 1–18, 2005.