

Semantic Web Services for Intelligent Responsive Environments

Christian Alberto Noriega Guerra and Flavio Soares Correa da Silva¹

Abstract. In this work we propose a model for intelligent responsive environments based on semantic web services, in which services are located in distributed devices and the interaction between devices and the environment takes into account the dynamics of contextual information and the availability of services. In the proposed model, all functionalities in the environment are provided as structures of services, which are automatically discovered and executed to support the users in specific tasks. We illustrate our ideas through a prototype implementation in our university library, in which location aware information is provided to users to help them locate items of their interest.

1 Introduction

Intelligent responsive environments are physical environments which can sense the necessity of certain services and provide them to users at the appropriate time and location. For example, a student in a library could check from his PDA what books are available, based on the courses in which he is enrolled in that academic term. Once a book is selected, the environment could inform the student, through his PDA, about related books that can also be relevant for his studies. The environment could also determine the location of the PDA - and hence of the student - and provide directions for the student to find the desired books, as a sort of "electronic compass". Finally, the environment could search for other users who could be close to the student and share similar interests with him, and inform the student about these users.

In order to provide users with automated services that enable these functionalities, an environment must identify the required services based on the goals and the context of the users. The required infrastructure for an intelligent responsive environment includes ubiquitous computing, ubiquitous communication, adaptive user interfaces, and the capability to collect and interpret contextual information [1, 14, 32].

In the present work we propose a model for intelligent responsive environments based on semantic web services [4]. We employ the *Web Services Modeling Ontology - WSMO*, which belongs to the *Web Services Management Framework - WSMF* [13] for the semantic specification of services in the intelligent responsive environment. The WSMO is based on four elements: (1) goals, (2) services, (3) mediators and (4) ontologies. These elements relate to each other through the WSMF, which is based on a particular flavor of description logics called *F-Logic* [21].

Our model is based on the notion of *task driven computing* [39], in which the users of an information system must focus on their tasks of interest, instead of the means to accomplish these tasks.

We introduce the utilization of an *Assumption based Truth Maintenance System - ATMS* [9] to manage the provision of services, based on the relations between preconditions and effects for each service. If the context of a user changes, the updated available services provide new alternatives to reach the user goals. The ATMS identifies minimal sets of preconditions that suffice to obtain a given effect, based on the descriptions of the available services.

Any functionality in an environment based on our model is provided through web services running on mobile or fixed devices. Our prototypical implementation employs two existing middleware systems for ubiquitous computing: (1) The WSAml middleware [18] provides a protocol for the discovery and location of web services; and (2) The MoCA middleware [30] provides the means to collect contextual information about mobile devices in the environment.

In section 2 we discuss the importance of context awareness for intelligent responsive environments, and how this concept can be implemented using *task-driven computing*. In section 3 we briefly review the notion of assumption-based truth maintenance systems, and how they can be employed to help manage the services in intelligent responsive environments. In section 4 we discuss semantic web services and their relevance to intelligent responsive environments. In section 5 we introduce our model for intelligent responsive environments, based on semantic web services. In section 6 we present our prototypical implementation at a university library. Finally, in section 7 we present some conclusions and proposed future work.

2 Context-awareness and Task-driven computing

An intelligent responsive environment must be sensitive to the context of its users. In the present work we adopt as a definition of *context* what is found in [10]: context is any information that can be employed to characterize the situation of an entity. An *entity* can be a person, a location or any physical object that is relevant in the interactions between the environment and its users. In our work, we consider that people can be identified by digital devices they carry with them, such as intelligent responsive phones and PDAs.

A *context sensitive* system uses contextual information to provide relevant information and services to users [8, 39]. Once the information about the status of a device has been captured, the environment must process it and match it with a contextual model to infer contextual information. Many existing systems for intelligent responsive environments adopt a layered approach, in which sensed information is used to feed a contextual model, and services are provisioned to users based on this model [34]. The overall quality of services in these systems is bounded by the quality of the contextual model.

¹ University of Sao Paulo, Brazil, email: {cnoriega,fcs}@ime.usp.br. This work has been partially supported by CNPq and Microsoft Research. The authors wish to thank the anonymous reviewers for their suggestions and corrections to the paper.

We have implemented context awareness in our system through *task-driven computing*. When a user moves to a different location in the environment, services and available information must be updated and reconfigured accordingly. *Task-driven computing* [39] is a technique to design systems in such way that users interact with high-level descriptions of tasks they wish to see accomplished, instead of dealing with lower-level internal services provided by the system. Using task-driven computing, we have *tasks* as an intermediate level of abstraction between the goals and intentions of users and the available resources in the environment that change depending on context.

Tasks and sessions are connected through four architectural concepts: (1) Task management - management of relations between tasks and contexts; (2) Encapsulation of services - appropriate abstraction of services to be offered to the accomplishment of tasks; (3) Management of sessions and services - configuration, discovery and coordination of services; and (4) Environment management - discovery and management of information related to devices and services related to each context.

Several lower level infrastructures provide the means to implement these concepts, e.g. Jini, Universal Plug and Play - UPnP and Service Location Protocol - SLP. These infrastructures, however, address mainly the management of individual services. Task-driven computing is implemented on top of these infrastructures, to allow the management of composed services.

Other systems provide the means to implement service composition, e.g. Enterprise Java Beans, CORBA and Web Services. These systems, however, do not address *what* services should be combined. This is indeed a complex problem, presently studied by many research groups [39, 34, 28, 35].

3 Assumption-based Truth Maintenance Systems

The goal of an assumption-based truth maintenance system (ATMS) is to identify minimal sets of assumptions that can justify a conclusion. An ATMS can work in conjunction with a logical inference system, to help search a knowledge base and find useful connections between logical assertions. The details including implementation issues - about ATMS can be found in [9].

Essentially, an ATMS manages the following: (1) Premises - a premise is a justification - i.e. a logical assertion - that does not depend on other conditions to be true. Roughly speaking, it corresponds to a fact in a PROLOG program; (2) Assumptions - an assumption is a logical assertion whose own set of assumptions is empty; (3) Justifications - a justification is a Horn clause that implements a logical assertion; (4) Nogoods - a nogood is the negation of a premise; (5) Environments - an environment in an ATMS has a meaning different from the one we have used in this work. Essentially, an ATMS environment is a set of assumptions that justifies a conclusion; and (6) Labels - a label is a set of ATMS environments.

Given an arbitrary logical assertion, the ATMS makes use of a specific algorithm to search its set of justifications and find the label that corresponds to it. The label that is built by the ATMS algorithm is such that the environments are minimal, i.e. they correspond to minimal sets of assumptions that can be used as alternatives to provide logical support to the selected assertion.

We have borrowed these concepts to support the management of tasks in intelligent responsive environments. Essentially, we have built an implementation of the original ATMS, in which assumptions are directly connected to available services in the environment, and logical assertions are connected to tasks. When a task is presented

to the environment, it uses the ATMS and the currently (i.e. at the present context) valid justifications to find the services it needs to accomplish that task. This is a necessary step for service composition, which must be complemented by the appropriate resources to orchestrate the identified services.

4 Semantic Web Services

A web service is the abstraction of a functionality that should be provided by a concrete agent [38]. A web service must be well defined and self contained. Operationally, a web service is specified as a computational resource which employs SOAP (*Simple Object Access Protocol*) packages over HTTP (*Hypertext Transfer Protocol*). The description of web services is made using WSDL (*Web Service Description Language*). It comprises the public methods and parameters available for remotely calling a service.

Web services have been extended with semantic descriptions. An extension of the OWL language [4] has been proposed specifically to encode ontologies for semantic web services. This extension has been called OWL-S (*Ontology Web Language for Services*) [26].

We have adopted the *Web Services Modeling Ontology* - WSMO [29] for the semantic description of the services available in the environment. This ontology has been built for the specification of semantic web services, to allow the discovery, execution, monitoring and composition of web services. We have used this ontology to encode the services provided by the environment and how they relate to each other. The WSMO is part of a larger framework, the WSMF (*Web Service Management Framework*) [13].

The WSMO defines *service capabilities* as 4-tuples of the form assumptions, pre-conditions, effects and post-conditions (*APEP*). Based on web service APEPs, a client can find the adequate service for its goal. Assumptions and effects work as constraints on the specification of the context of the environment. Pre-conditions and post-conditions correspond to inputs and outputs of the execution of a service. As an example, the assumption

```
?someStudent [hasDepartment hasValue "IME"
               memberOf fex#StudentFenix
```

establishes that a service is provided only to students of the *IME* department, and the pre-condition

```
?someBook memberOf lib#Book
```

establishes that the service requires a *Book* as input.

The WSMO defines two visions for *service behavior*: (1) choreography - informs the clients how the web service works and (2) orchestration - informs other web services about the behavior of the service.

The behavior of the service is characterized as a *finite state machine*, in which *transition rules* determine the consequences of specified pre-conditions. If the pre-conditions of a rule are fulfilled, then the consequences are achieved. The client stores the state of the execution of services. When a rule is used, its consequences are inserted into the state. A transition rule transition has the form:

```
if (?someStudent [hasLocation hasValue
                  ?someLocation] memberOf fex#StudentFenix)
then
    add(?relatedbooks memberOf lib#RelatedBooks)
```

This rule is interpreted as: if the client state contains the student location then a list of relevant books can be found.

The WSMF defines an execution environment for semantic web services, which however until the date of the preparation of this paper had not been implemented.

Each transition rule relates to a concrete method of the WSDL interface. We have defined execution properties for each transition rule (method) of the web service. These properties are related to: (1) types of the results of the method, which can be *persistent* or *transient*; and (2) frequency of execution the method, which can be *once* or *many* times.

For example, location dependent services may need to be updated frequently and relate to the user instead of the device. Such services may be classified as *transient-many* methods.

5 Proposed Model

Our proposed model for intelligent responsive environments is based on task-driven computing. It employs an ATMS to identify, given a task, the services whose composition can accomplish it. The actual communication and exchange of services between devices is based on semantic web services.

In this section we provide a brief overview of the model. A detailed account of our model, as well as of how to access the corresponding code of its implementation, can be found in [15].

5.1 Architecture

Our proposed architecture is based on three components: (1) Environment server - this component manages the relation between components and devices, as well as the discovery and execution of services. It is located on a fixed node in a computer network; (2) Service providers - these components are the components that, when executed, offer some service to the environment. Service providers can be fixed or mobile; and (3) Environment clients - these are executed in mobile devices.

These components are depicted in Figure 1. The client sends to the environment server updated contextual information, task definitions and knowledge base facts. The environment server discovers and executes appropriate services and updates the user knowledge base with contextual information of the device and results of service executions. Figure 2 shows the components of the environment server.

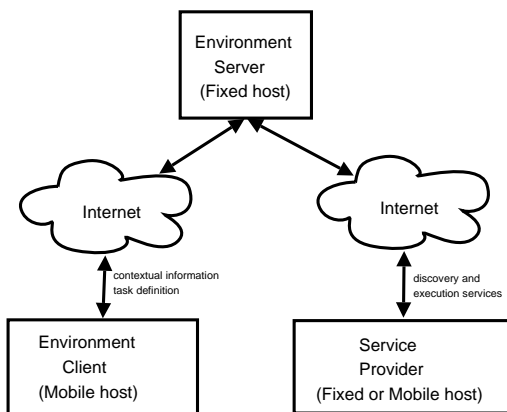


Figure 1. Basic components of the proposed architecture.

For each device in the environment are created an associated ATMS and knowledge base.

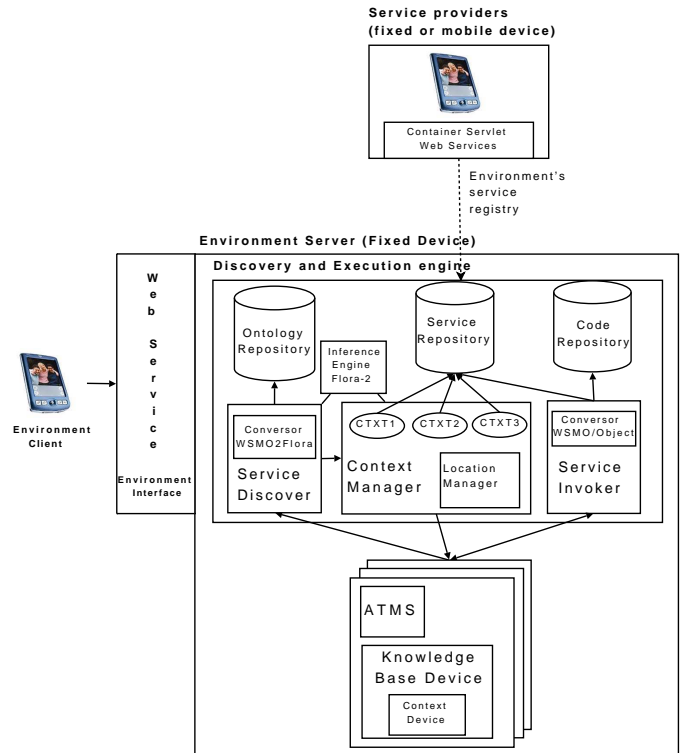


Figure 2. Components of the environment server.

The proposed architecture defines three repositories: (1) Services repository - stores the semantic specifications in WSMO of the services that are available in the environment. We have implemented access to this repository using resources available from WSAmI. The services discovery mechanism searches this repository to discover the available services; (2) Ontologies repository - stores all ontologies employed in descriptions of services in the environment. The required ontologies are retrieved from this repository to process inferences accordingly in order to build service compositions; (3) Code repository - stores Java classes dynamically generated from services, which are loaded in real time for the automatic execution of services.

A *context manager* performs the efficient management of services in the environment, based on declared connections between pairs context / service. Services that are not connected to any specific environment are generic services, considered to be available in all contexts. The composition of services is built based on inferences performed using F-logic, in order to discover implicitly available services. The context manager also monitors the context of a device and updates its set of available information accordingly. This is implemented using MoCA [30] to match locations with contexts, thus providing the required relation between location and context to the system.

The available information and services identified by the context manager feeds the *services discovery mechanism*, which then triggers the ATMS to relate services and information with potential effects that can come out of them. These effects determine what tasks can be accomplished.

Once the required services to accomplish a task are determined, they are invoked using a *Dynamic Invocation Interface* - DII for *Jax-RPC*. DII requires the following concrete pieces of information to be triggered: the service execution end-point, the name of the spe-

cific method to be triggered, and the required objects that function as parameters of the method.

The same methods that implement the invocation of services capture the results of the corresponding services and update the information available to each device accordingly.

Each device has a set of pieces of information that characterize its context, which includes information about location. This is called the *knowledge base* of the device. The knowledge bases are constantly updated by the context manager.

The sequence of steps for discovery and execution of services is as follows:

Registry: A device is registered in the environment. The environment builds a knowledge base, an ATMS and a context for the device.

Specification of a task: A user sends to the environment the description of a task. This can be done explicitly and manually by the user, or automatically by a device carried by the user.

Identification of available services: The context manager retrieves the set of available services.

Matching of task and services: Using the ATMS, minimal sets of services are identified in order to accomplish the desired task.

Execution of services: The appropriate services are invoked, and the results of their execution are stored in the appropriate knowledge bases. If the task cannot be accomplished, pro-active help is provided to the user, suggesting nearby locations where the necessary services can be found.

The context manager updates the user's knowledge base with contextual information, including the user location. The context manager also stores service availability information in context as pairs such as:

```
< { ?someStudent[hasDepartment hasValue "IME"
    memberOf fex#StudentFenix ],
    {Library} >
```

This pair, for example, defines that the `Library` service is available for students of the `IME` department. This association allows the reduction of the search space and the management of service availability in the environment.

The implementation of this module makes use of `CIS` and `LIS` services of the `MoCA` middleware [30] for contextual and location dependent information.

Facts in the knowledge base are stored as pairs of `WSMO` expressions and Java objects. The expressions describe the Java objects, and the objects are reified as `WSMO` expressions. Consider, for example, the pair:

```
< (?someStudent[hasLocation hasValue
    ?someLocation] memberOf fex#StudentFenix),
    {Student@1232fasf, Student@3g3122} >
```

In this example we find the definition of which objects have an attribute associated to the `hasLocation` ontology property. This association allows the retrieval of the appropriate objects in a `WSMO` expression for the execution of services.

The `WSMO` expression describes the attributes which have the Java objects. For example, in the previous case, the expression indicates which the `Student` objects have the attribute `location`, for the property `hasLocation`.

In this mapping we use the convention for ontologies, which uses `has` notation for named concept attributes; and the convention for classes, which uses `get-set` notation to access class properties.

5.2 Service discovery mechanism

The process for service discovery is divided in two phases: (1) logical service discovery, which uses service capabilities to match services with the user goals; and (2) ATMS based services composition, in which rule transitions are propagated and an ATMS is used to identify valid environments that characterize minimal sets of preconditions for services.

The ATMS manages the context of the environment and the knowledge base manages the context of the user and the device.

Services and ontologies are rewritten as F-Logic [21] assertions. A service must satisfy the following condition:

$$\exists Ser \bullet O, Ser_{pos}, KB_{disp} \models_{F-Logic} G \wedge Ser_{hip}$$

where Ser is the service, O are the ontologies, Ser_{pos} are the service post-conditions, KB_{disp} in the device knowledge base of the user, G is the user goal and Ser_{hip} are the assumptions. The services that satisfy the goal are selected for the next phase.

The transition rules of the services described in the previous discovery phase are rewritten as ATMS justifications. An ATMS justification has the form:

$$\langle effect_i, \{\{condition_1, condition_2, \dots\}, \{\dots\}, \dots\} \rangle$$

where $effect_i$ and $condition_i$ are the effect and conditions of a transition rule. For example, in the following transition rule:

```
if(?someStudent[hasLocation hasValue
    ?someLocation] memberOf fex#StudentFenix)
then
add(?relatedbooks memberOf lib#RelatedBooks)
```

the ATMS justification is:

```
< ?relatedbooks memberOf lib#RelatedBooks,
{ {?someStudent[hasLocation hasValue
    ?someLocation] memberOf fex#StudentFenix}},
{ ?someStudent[hasLocation hasValue
    ?someLocation] memberOf fex#StudentFenix}
=> ?relatedbooks memberOf lib#RelatedBooks >
```

This justification is propagated in the ATMS. The conditions of the transition rule are added to the ATMS as assumptions. An ATMS assumption has the form:

$$\langle cond_i, \{\{condition_i\}\} \rangle$$

Justifications are propagated in the ATMS to identify appropriate ATMS environments.

When there is a change in the availability of services, the ATMS must update its sets of assumptions. Each environment contains the minimal and consistent assumptions required to accomplish a task that is relevant for a user.

The ATMS of a device is updated with the services that are available in a context and useful for the user task. Pre-conditions of transition rules of services that are no longer available are withdrawn from the ATMS. A service is not available if we have (1) communication problems, e.g. the end-point of the web service is not accessible or (2) the service does not satisfy the constraints of the context environment, e.g. when the user moves to an area where the service cannot be executed.

The knowledge base manages the context of the device, and the ATMS manages the context of the environment and the availability of services required for the user task.

When adequate ATMS environments cannot be found, new services are searched to be placed as assumptions in ATMS environments. In other words, assumptions become sub-goals and discovery is restarted to look for further assumptions that support these sub-goals.

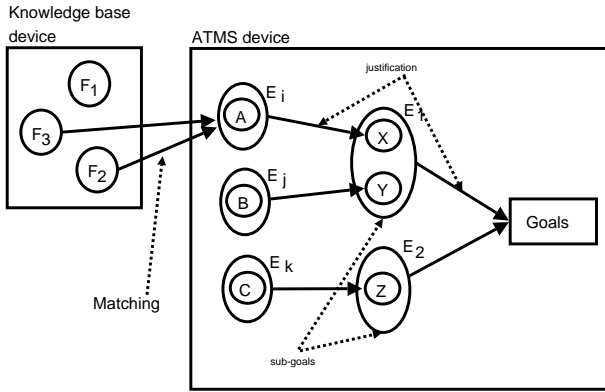


Figure 3. Matching ATMS and the knowledge base.

For example, in Figure 3, X and Y are the sub-goals and the environments E_i , E_j and E_k are the new environments found for this sub-goals. Thus, the previous ATMS state is:

`< effect, {{X, Y}, {Z}} >`

Propagation of new justifications for the environments X , Y and Z produce the environments E_i , E_j and E_k . The new ATMS state is:

`< effect, {{X, Y}, {X, B}, {Y, A}, {A, B}, {Z}, {C}} >`

Proactive support is provided to the user: when it is not possible to add new rule transitions to find an environment that can support the user task, recommendations are provided so that the user can move to a better location where perhaps the required conditions can be found. For example, if the environment has the assumption:

`?somestudent[hasLocation hasValue "areal"]
memberOf fex#StudentFenix`

then, in order to satisfy this environment is recommended to the user to move to *areal*.

6 A Prototypical Implementation: The math library at the University of Sao Paulo (USP)

We have built an experiment to test our ideas at the Library in our Institute. In order to build this experiment, we have implemented an environment server on a high-end portable computer using J2EE and a generic client on a PDA using the Microsoft .NET platform.

The server has a fixed network node and an IP identifier. We have employed a JBoss applications server as a container of web services, and CORBA to manage the communication with the client.

In Figure 4 we show the main classes used in the implementation of the environment server:

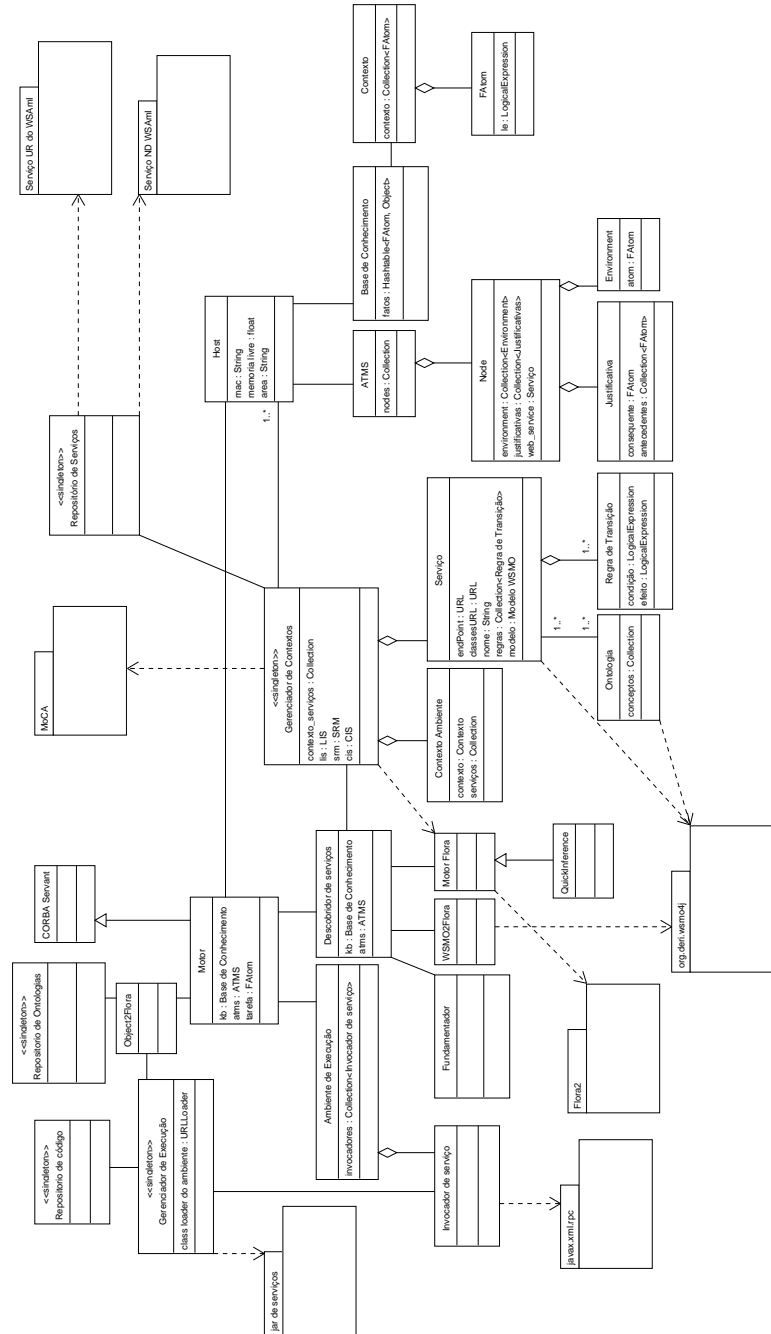


Figure 4. Main classes in the environment server.

Device engine: This class encapsulates the objects and methods required for the discovery and execution of services. Each instance of this class contains an ATMS and a knowledge base connected to a specific device. This class also implements CORBA objects to manage the connection between the server and the clients.

Host: Manages the information about a mobile device in the environment. The contextual information collected by MoCA is updated in this class.

Knowledge base: Manages the knowledge base of a device.

ATMS: Implements the ATMS corresponding to each device.

Service: Manages the information related to a service in the environment: execution end-point of the web service, an instance of the WSMO model that describes the service, the URL where the implementation of the service can be found, and a service unique identifier.

F-logic engine: Manages and performs inferences in F-Logic.

WSMO2F-logic: Translates WSMO service specifications into F-logic notation.

Services repository: This class is based on the services *UR* and *ND* found in WSAmI. It retrieves services registered in the WSAmI universal repository (*UR*) and searches for specific services that match with the service identifier *ND*, based on the semantic specifications of services declared in WSMO.

Context manager: Manages the relations between services and contexts.

Services discovery: Performs all required steps to identify sets of services given a task.

Service identifier: Connects each set of services given by the services discovery to the corresponding methods whose execution corresponds to those services.

Execution manager: Orchestrates the execution of the other classes.

Execution environment: Implements service invocation objects for each method to be executed.

Service invocation: Identifies the required parameters for the execution of methods that implement a service. Queries the knowledge base of a device to identify JAVA objects whose semantic description matches with the required parameters. When necessary, this class also updates the knowledge base.

Object2F-logic: Translates F-logic expressions into JAVA objects.

Access to the environment server is made through a web service which abstracts the relevant processes in the environment to the clients. The interaction between the clients and the environment server is implemented by the following functions:

```
void register(HostData, ContextData): This function registers a device in the environment. It takes the MAC address of the device encapsulated in HostData and an initial context in ContextData. Contextual information is encoded as sentences such as crhistian[hasNusp hasValue "5055668"] memberOf StudentFenix.
```

```
void setContextForHost(HostData, ContextData): This function updates the context of a device using the information contained in ContextData.
```

```
boolean hasException(HostData, GoalData): This function starts up the process of discovery and execution of services. It takes the specification of a task in GoalData and returns true if the process is successful, or false if some exception is triggered, e.g. requiring pro-active behavior.
```

```
WebServiceData[] getServiceForGoal(HostData): This function publishes the services and corresponding methods
```

that were executed under normal conditions (i.e. when the previous function returns true).

```
Exception getLastProActiven(HostData): This function publishes the services and methods that were executed under abnormal conditions (e.g. when pro-active behavior is required).
```

```
Exception getLastMultipleObject(HostData): This function returns a list of potential objects related to a concept.
```

```
PositionData getPosition(HostData): This function returns the location of the device.
```

```
String[] getKB(HostData): This function returns the knowledge base of the device as a collection of WSMO expressions.
```

```
void addFactToKB(HostData, String): This function adds a new entry to a knowledge base.
```

The client was implemented using C# and Microsoft Visual Studio .NET 2005. It runs on Windows CE 5.0 in an HP iPAQ Pocket PC. It is essentially a client for the web services provided by the environment server. Web services are the means to allow seamless interoperability between .NET and J2EE.

We have implemented a generic graphical user interface, so that users can interact with the environment using WSMO sentences. A user friendly interface should be built on top of this interface to make this system truly available to end users. In our specific prototype, the WSMO sentences represent books in the library, their bibliographical information and physical location.

We had to build a map of the library to configure the location service provided by MoCA. The library was organized in 104 areas.

We have built the following ontologies for this example:

Library: In this ontology we have built concepts related to books in a library, such as *Book*, *ListBook*, *BookTheme*, *BookPlace* and *RelatedBooks*.

Fenix: In this ontology we have built concepts related to academic records of university students². It contains concepts such as *StudentFenix*, *TeacherFenix*, *CourseFenix* and *Department*.

Ambient: This ontology contains concepts related to the environment, such as *Device*, *User* and *Context*.

Location: This ontology contains concepts related to locations in the environment. The base concept is *Location*, with corresponding attributes *x*, *y* and *z*. It also contains the concept *Region* that refers to a composite area in the environment. For example, a book can be associated with a *Location* that belongs to a *Region*.

People: This ontology contains concepts that characterize the presence of people in the environment.

We have implemented the following services in our prototype:

Fenix: With this service the user can check the academic status of a student. The service provides the following information:

1. Relevant books according to the courses in which the student is enrolled.
2. List of courses in which the student is enrolled.
3. List of books that are available in this library and can be relevant to the student.

Library: This service encapsulates all functionalities provided by the library. It allows the following operations:

² The name *Fenix* relates to the information system used at the University of Sao Paulo for that purpose.

1. Enter a queue for the withdrawal of a book.
2. Find books that relate to a book of interest.
3. Find books that relate to a specific topic.
4. Find books that are physically close to the present location of the user.

Location: This service guides the user to find a book within the library. It takes a book ID and a student ID and location and returns directions to find the book.

All interactions between the user and the environment run through WSMO sentences expressing terms of the environment ontologies. A generic mobile client was implemented on a Pocket PC under Windows CE. This client implementation allows (1) to send goals definition to the environment server; (2) to add and remove fact of the knowledge base, including context information; and (3) to consult facts in the knowledge base and transient information.

Figure 5 shows the generic interface of the client application, and Figure 6 shows the interface for library environment. It shows the directions presented to a user to find a selected book.



Figure 5. Generic client application running in environment device (Pocket PC).

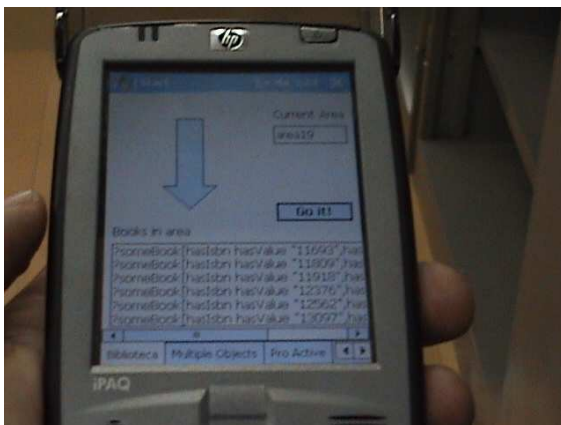


Figure 6. Library environment interface.

7 Conclusions and Further Work

In [12] some scenarios are presented in which intelligent responsive environments can be useful. These scenarios also pose challenges for

research in ambient intelligence. We suggest three scenarios to show the potential of our proposed model:

1. *An environment that reacts to the presence of people:* In this scenario services are discovered according to the user location. For the user, it means that the environment reacts to her presence when she moves e.g. into the "room1" area.

This scenario is based on a sensor-actuator model. Sensors are devices on the environment in charge of capturing contextual information and inform the actuators which handle physical changes in the environment. In our proposed model sensors and actuators are controlled through services.

For example, consider a thermometer in area room1 (sensor) and a heater in the same area (actuator). The thermometer can have the specification:

```
if(?user[hasLocation hasValue "room1"]
    memberOf User)
    then
    add(?temperature memberOf Temperature)
```

This transition rule defines that the sensor returns the temperature in area room1. The heater can have the specification:

```
if(?temperature[hasValueCelsius
    hasValue ?somevalue]
    and ?user memberOf User)
    then
    add(?temperature
    memberOf EnvironmentTemperature)
```

This transition rule defines that the heater returns the desired temperature for a given user. The implementation of the heater can contain further rules that specify what to do to move from Temperature to EnvironmentTemperature. The discovery mechanism allows sensors and actuators dynamically.

2. *A location aware recommendation system for a wide area:* In this scenario we consider a wireless network area such as a metropolitan area and a location system such as GPS or Placelab [23] for this type of environment.

Services are associated to specified areas in the metropolitan area. When a user moves about, appropriate services are discovered and executed. For example, consider a recommendation system for consumer products. A user can add to his knowledge base the intention to buy some product, e.g. a refrigerator:

```
fridge[hasModel hasValue "LG"]
    memberOf Product.
preference[hasElements hasValue fridge]
    memberOf Preference.
```

Environment services can be implemented for a shopping mall in the area under consideration, containing rules like e.g.:

```
if(?someprefer[hasElements hasValue
    ?someproduct] memberOf Preference
    and ?user[hasLocation hasValue ?somearea]
    memberOf User
)
    then
    add(?someInfo memberOf ProductInformation)
```

In this scenario, users can access their architectural components (knowledge base, ATMS, discovery and execution mechanisms) through the Internet without losing their states.

Additional service capabilities can be provided by adding semantic descriptions for those functionalities. For example, to buy selected products, service implements that functionality in a WSDL method associated to the next transition rule:

```

if(?someprefer[hasElements hasValue
    ?someproduct] memberOf Preference
and
    ?creditcard[hasUser hasValue ?someuser,
        hasType hasValue ?someType]
        memberOf CreditCard
)
then
    add(?sometatus[hasStatus hasValue buyOK]
        memberOf StatusPayment)

```

3. An environment capable of providing composite services:

In this scenario we can illustrate the implementation of architectural services with the purpose of accessing components (knowledge bases) of other devices in the environment, thus looking for alternative means to accomplish a task. Sophisticated planning and collaborative systems can be used to build compositions of services.

These three scenarios illustrate some possibilities in which intelligent responsive environments can be effectively employed to provide good services to users. Any implementation of an intelligent responsive environment based on our model requires the definition of: (1) a collection of ontologies, in which the semantic specification of services is built; (2) semantic definition of services in the environment and (3) service implementation with web services technologies.

In the case of a reactive environment, the main feature to be considered is the dynamic discovery and execution of services. In this scenario we can, for example, be interested in services that control the behavior of devices distributed across the environment, so that for example a printer, a digital display or an air cooling device is started when it senses the presence of a mobile device such as a PDA or a smartphone.

A location sensitive recommendation system for a wide area - e.g. a university campus or a whole urban area - requires that location information is provided by appropriate technologies - such as, for example, GPS. A user can inform the environment through his/her mobile device the interest in buying a specific product - say, a refrigerator. This information could be added to the device's knowledge base, which would then interact with different services and devices as the user moved about the environment. Interesting lower level implementation challenges become relevant in this scenario, such as the need to replicate the information and services provided by the environment server in geographically distributed workstations across the environment.

The composition of basic services can greatly enrich the capabilities of a intelligent responsive environment. In order to fully implement compositions, we would need to enrich our model with explicit composition capabilities, provided for example by automated planning systems. We envisage difficulties, however, to provide appropriate composition capabilities that perform well under the stringent real time requirements that characterize intelligent responsive environments and location aware systems in general.

All in all, we believe that our proposed model for intelligent responsive environments based on semantic web services and task-based computing has great potential in many application areas. The flexibility provided by web services and task-based computing is use-

ful to support scalability and the seamless utilization of heterogeneous devices. The semantic specifications granted by semantic web services and semantic web technology is useful to design sophisticated interaction mechanisms. The utilization of ATMS to constrain the sets of relevant services to accomplish a given task has proven to be a useful tool to improve the computational efficiency of the model. Indeed, our experiments have indicated that this model can be implemented efficiently, to abide by the real time requirements that characterize intelligent responsive environments and location aware systems, at least for systems in which the composition of services does not require the utilization of highly sophisticated planning systems.

In the prototype system like the one described in this paper, it is very important to conduct user evaluation. This will reveal flaws in the system and should provide directions for future work. Currently, without any user evaluation data, it is difficult to assess how easy and useful the system is going to be in actual practice and what limitations it presents. We plan in the future to develop some user tests to assess and improve the system.

Further empirical analysis must also be done to evaluate how the system scales to large numbers of services and complex coordination tasks.

REFERENCES

- [1] Alcaiz M. e Rey B., *New Technologies For Ambient Intelligence*, Ambient Intelligence, IOS Press 2005.
- [2] Angele J. e Lausen G., *Ontologies in F-Logic*, Handbook on Ontologies in Information Systems. International Handbooks on Information Systems 29-50, Springer Verlag, 2004.
- [3] Burbey I., *Ubiquitous Internet Computing*, WWW Beyond the Basics (<http://ei.cs.vt.edu/book/index.html>), Prentice Hall, 1998.
- [4] Bussler C., Maedche A. e Fensel D., *A Conceptual Architecture for Semantic Web Enabled Web Services*, ACM Special Interest Group on Management of Data: Volume 31, Number 4, 2002.
- [5] Bussler C., Maedche A. e Fensel D., *Web Services: Quo Vadis*, IEEE Intelligent Systems, 2003.
- [6] Chen G. e Kotz D., *A Survey of Context-Aware Mobile Computing Research*. Dartmouth Computer, Science Technical Report TR2000-381. Department of Computer Science - Dartmouth College, 2000.
- [7] Chen H., Finin T. e Joshi A., *An Intelligent Broker for Context-Aware Systems*, Adjunct Proceedings of Ubicomp 2003, USA, October, 2003.
- [8] Cortese G., Lunghi M. e Davide F., *Context-Awareness for Physical Service Environments Ambient Intelligence*, IOS Press, 2005.
- [9] de Kleer J., *An Assumption-Based TMS*, Artificial Intelligence, Volume 28, Issue 2, 127-162, 1986.
- [10] Dey A. K. e Abowd G. D., *Toward a better understanding of context and context-awareness*, GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999.
- [11] Davies J., Fensel D., e Van Harmelen F., *Towards the Semantic Web: Ontology-Driven Knowledge Management*, John Wiley Sons, 2003.
- [12] Ducatel K., Bogdanowicz M., Scapolo F., Leijten J. e Burgelman J-C. *Scenarios for Ambient Intelligence in 2010, ISTAG*, February 2001.
- [13] Fensel D. e Bussler C., *The Web Service Modeling Framework WSMF*, Electronic Commerce: Research and Applications, 113-137, 2002.
- [14] Gaggioli A., *Optimal Experience in Ambient Intelligence*, Ambient Intelligence, IOS Press 2005.
- [15] Guerra N. A. Crhistian, *Um modelo para ambientes inteligentes baseado em servios web semnticos*, Tese de Mestrado IME-USP, Agosto 2007.
- [16] Helal A., Mann W., Elzabadiani H., King J., Kaddourah Y. e Jansen E., *Gator Tech Intelligent responsive House: A Programmable Pervasive Space*, IEEE Computer magazine, March 2005.
- [17] Hansmann U., Merk L., Nicklous M. e Stober T., *Pervasive Computing*, Springer Second Edition, 2003.
- [18] Issarny V., Sacchetti D., Tartanoglu F., Sailhan F., Chibout R., Levy N. e Talamona A. *Developing Ambient Intelligence Systems: A Solution based on Web Services*, In Journal of Automated Software Engineering. Vol 12. 2005.

- [19] IST Advisory Group (ISTAG), *Ambient Intelligence: from Vision to Reality*, Ambient Intelligence, IOS Press 2005.
- [20] Kifer M., Lara R., Polleres A. e Zhao C., *A Logical Framework for Web Services Discovery*, ICWS 2004.
- [21] Kifer M. e Lausen G., *F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme*, International Conference on Management of Data archive ACM SIGMOD, 1989.
- [22] Kleiner A., *Game AI: The Possible Bridge between Ambient and Artificial Intelligence, Ambient Intelligence*, IOS Press, 2005.
- [23] LaMarca A., Chawathe Y., Consolvo S., Hightower J., Smith I., Scott J., Sohn T., Howard J., Hughes J., Potter F., Tabert J., Powledge P., Borriello G. e Schilit B., *Place Lab: Device Positioning Using Radio Beacons in the Wild*, In proceedings of Pervasive 2005, Munich, Germany.
- [24] Masuoka R., Parsia B. e Labrou Y., *Task Computing - The Semantic Web meets Pervasive Computing*, <http://www.flacp.fujitsulabs.com>, Fujitsu Laboratories of America, Inc., 2004.
- [25] OASIS - Organization for the Advancement of Structured Information Standards, *OASIS Reference Model for Service Oriented Architecture V1.0*, Official Committee Specification approved Aug 2, 2006.
- [26] OWL Services Coalition, *OWL-S: Semantic Markup for Web Services*, <http://www.daml.org/services/owl-s/1.0/>.
- [27] Paolucci M., Kawamura T., Payne T. R. e Sycara K., *Semantic Matching of Web Services Capabilities*, 2004.
- [28] Project Oxygen, <http://www.oxygen.lcs.mit.edu/>, site acessado pela ltima vez 01/06/2007.
- [29] Roman D., Lausen H. e Keller U., *Web Service Modeling Ontology (WSMO)*, <http://www.wsmo.org/TR/d2/v1.2/>, Working Draft D2v1.2, April 2005.
- [30] Sacramento V., Endler M., Rubinsztein H. K., Lima L.S., Goncalves K., do Nascimento F.N. e Bueno G., *MoCA: A Middleware for Developing Collaborative Applications for Mobile Users*, ACM/IFIP/USENIX International Middleware Conference, Toronto, October, 2004.
- [31] Schmidt A., *Interactive Context-Aware Systems Interacting with Ambient Intelligence*, Ambient Intelligence, IOS Press, 2005.
- [32] Schilit N., *A System Architecture for Context-Aware Mobile Computing*, Phd Teses, Columbia University, 1995.
- [33] Singh M. e Huhns M., *Service-Oriented Computing: Semantics, processes and agents*, John Wiley Sons, 2005.
- [34] Sousa J. P. e Garlan D., *Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments*, Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, August 25-31, 2002.
- [35] Srivastava B. e Koehler J., *Web Service Composition: Current Solutions and Open Problems*, ICAPS 2003 Workshop on Planning for Web Services, 2003.
- [36] Strang T. e Linnhoff-Popien C., *A Context Modeling Survey*, Workshop on Advanced Context Modeling, Reasoning and Management as part of UbiComp 2004.
- [37] Xavier E. e Correa da Silva. F. S., *Expressing Systems Capabilities for Knowledge Coordination*, AAMAS'2002.
- [38] W3C Working Group, *Web Services Architecture*, <http://www.w3.org/TR/ws-arch/>, W3C Working Group Note 11 February 2004.
- [39] Wang Z. e Garlan D., *Task-Driven Computing*, Technical Report, School of Computer Science, Carnegie Mellon University, May 2000.
- [40] WSMO Group, *The Web Service Modeling Language WSML*, <http://www.wsmo.org/TR/d16/d16.1/v0.2/>, WSML Final Draft 20 March 2005.