SeCoAS: An Approach to Develop Semantic and Context-Aware Available Services

ABSTRACT
The dynamic nature of pervasive environments requires that context-aware applications be supported by mechanisms that automatically provide contextual information. Semantic Web Services (SWS) demonstrate capabilities to perform automatic discovery and composition of services. In this paper, we propose an approach, called Semantic and Context-Aware Available Services (SeCoAS), that combines Semantic Context Modeling and Reasoning with Semantic Web Services. The use of the approach is presented in a case study.

Categories and Subject Descriptors
H.3.5 [Information Storage and Retrieval]: Online Information Services—Web-based services; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods

General Terms
Algorithms, Design, Management.

Keywords
Context-aware, semantic web services, rules.

1. INTRODUCTION
The Pervasive Computing (PerCom) paradigm is becoming a reality. Personal devices (PDAs, smart phones, GPS devices, etc) for example, have become part of our daily lives with new functionalities and services included at each new release. Environments empowered with personal devices allow users to perform traditional activities on smart ways. One of the most challenging aspects of PerCom is the implementation of the so called “Calm Technology”, which aims at fitting technology on our daily lives [22].

Pervasive applications need to be context-aware to operate in dynamic environments that reflect user mobility, availability of computational resources as well as constants changes in user preferences [7]. A complete definition of context is not a trivial task. Dey et al. [6] define Context as “any information that characterizes a situation related to the interaction between humans, applications and the surrounding environment” and Context-awareness as “a property of a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the users task”.

Related to the definition of context is the question of how to represent context. The literature reports that different techniques were employed to model context information. A survey of these techniques is given by Strang et al. [18], who advocate that the ontology-based approach has more advantages than other techniques. Using ontologies to represent context is promising because it is semantically defined and allows reasoning [3].

A good representation for context should enable a wider range of capabilities and a true separation of sensing context from the programmable reaction to that context [1]. In fact, Jacob et al. [9] discusses that the processing of ontologies can be exploited to provide for the propagation of context information.

The dynamism of context-aware environments needs to be treated automatically and transparently to the user. This mechanism needs to locate, invoke and compose the services on the fly and without user intervention. Such dynamism needs to impose some limitations in the definition of the system behavior at design time [20].

Considering that Semantic Web Services (SWS) enable a broad range of new automation tasks – which in some cases used to be performed by humans, including automated discovery, invocation and composition [13] – a synergetic solution may result from the the integration of the Semantic Web Services approach with the concept of context-awareness.

We propose to describe context-aware applications as Semantic Web Services in OWL-S to enable the automation of discovery, invocation and composition. As result, the changes in context-aware environments can be adapted dynamically. We combine Semantic Web Services using the OWL-S ontology with context ontologies to develop our approach called SeCoAS: Semantic and Context-Aware Available Services.

Our objective is to enable that applications in context-
aware environments can adapt to the changes occurred in the environment as far as availability is concerned. As result, we enable a dynamic environment which can discover, invoke and compose context-aware services automatically.

In this work we use an ontology based approach that uses OWL to model and reason over context information. We model the context aware services using OWL-S and test this approach in a case study, which describe some services related with an intelligent home environment.

This paper is organized as follows. Context modeling and reasoning with ontologies are presented in Section 2. Section 3 describes Semantic Web Services. Section 4 presents our approach. Section 5 presents a case study. Section 6 presents related works. Section 7 summarizes our contributions and highlights future works.

2. CONTEXT MODELING AND REASONING

Context-Awareness has evolved as a key technology to ubiquitous and pervasive computing [19]. Currently, ontology-based techniques are broadly used to model context information. The OWL language is a W3C\(^1\) (World Wide Web Consortium) recommendation to semantically express information and to reason over it. The OWL ontology contains individuals, properties, classes and restrictions.

In this work, we reuse part of other ontologies for context-aware environments as was done in CoBrA ontology [3], and SeCoM [5]. Person ontology reuses some concepts from Foaf ontology\(^2\) (FOAF Vocabulary Specification 0.9) and from Relationship ontology\(^3\) (A vocabulary for describing relations between people). The Time ontology is a generic ontology for time that we reuse in this work. Finally, the Policy and Device ontologies we have created and proposed in this work.

2.1 Person Ontology

Person ontology has relevant information about a person. A family may be modeled using the Person ontology. Ricardo and Mariela are married and are parents of Alex and Andrea. Carlos is a friend of Ricardo and Carol works with Ricardo.

Figure 1 illustrates the use of the Person ontology. It shows two types of properties - Data Type Properties and Object Properties. An example of Data Type Property is the name of the person, which is a string. Object Properties are parentOf and childOf, which are inverse between them. We declare that Ricardo (Person_1) is parent of Alex (Person_3) (line 7 in Document 1) and that Andrea (Person_4) is child of Ricardo. Document 2 presents a query returning that Alex and Andrea are children of Ricardo. This return is possible because we use the Pellet DL reasoner [17], which deals with incomplete information. It derives that Ricardo is also parent of Andrea because the property parentOf is an inverse property of childOf.

2.2 Time Ontology

All things occurring in an intelligent environments can be related to spatial and temporal relations. OWL-Time\(^4\) ontology is a complete specification of time that is required for Semantic Web applications [8]. We use this ontology to define temporal concepts such as Instant, Interval, InstantEvent, and IntervalEvent. A temporal concept could be divided into instants such as the actual time, the exact time that a person enters in a space, or intervals as morning, afternoon, and the time interval in which a person was in a local or made an activity.

2.3 Policy Ontology

Context-Aware information should be suggested policies to protect person privacy and to control the access to information and to services. The class personAllowed_EnterHome (Document 3) is formed by the people that belong to the union of family members (property familyMember) and authorized people (property authorizedPerson), who are the people that are not family members but are authorized to enter in home by a family member. This class serves to restrict entrance to home only to people that belong this class.

---

1. http://www.w3.org/
2. http://xmlns.com/foaf/0.1/
A context-aware environment contains a diversity of pervasive devices. The Device ontology allows the descriptions of these devices. The device categories are: Sensors collect information from the environment; Actuators that perform actions and allow the interaction with the environment, and Appliances that have specific function and could be programmed.

For example, device FrontDoor can be an actuator that gives access to a person to pass from a space to another. Specifically, this device allows someone to enter a home when the action OpenDoor. The behavior of this device could be represented by the rule performAction_OpenDoor (Document 4). The device performs the action OpenDoor if its state is Locked. This state is updated to UnLocked and access is given to a person to pass from a space to another.

```
0 <!-- Document 4 -->
1 <!-- A rule example of a device behavior -->
2
3 [performAction_OpenDoor:
4  (?door dev:name 'FrontDoor')
5  (?door dev:state 'Locked') ->
6  (?dev dev:performAction 'OpenDoor')
7  (?door dev:state 'UnLocked')
8 ]
```

### 3. SEMANTIC WEB SERVICES IN OWL-S

We adopt the definition of the W3C about Web Services: “Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL)

Web services with explicit semantic annotation are called Semantic Web Services (SWS).

#### 3.1 OWL-S

The Ontology Web Language for Services (OWL-S)

- allows explicit semantic specifications to Web Services. Explicit semantics can enable flexible automation of service provision and usage, and support the construction of powerful tools and methodologies

The OWL-S provides three essential types of knowledge about a service (Figure 2): the ServiceProfile, the ServiceModel and the ServiceGrounding.

![Figure 2: OWL-S Ontology for Services.](http://www.w3.org/Submission/OWL-S/)

#### 3.1.1 Service Profile

The class ServiceProfile provides a superclass of every type of high-level service description. This class has several properties including serviceName, textDescription, contactInformation, categoryName, hasInput, hasOutput, hasPrecondition, and hasEffect.

The functionality description of a service is the specification of what functionality the service provides and the specification of the conditions that must be satisfied for a successful result. OWL-S Profile has focus on two aspects of the service functionality that forms the IOPE (inputs, outputs, preconditions and effects): i) inputs and outputs represent the information transformation; ii) preconditions and effects represent the state change produced by the service execution.

The Profile ontology does not provide a schema to describe IOPE’s instances. However, such a schema is defined in the Process ontology.

#### 3.1.2 Service Process

In order to give a detailed perspective on how a service operates, it can be viewed as a process. Specifically, OWL-S defines a subclass of ServiceModel, the ProcessModel.

A process, in OWL-S, has two functions. First, to produce a data transformation from a set of inputs to a set of outputs. Second, to produce a transition in the world from one state to another. This transition is described by the preconditions and effects of the process [12].

The class Process has three types of processes: Atomic, Composite and Simple. The atomic processes are directly invocable (via appropriate message passing). Atomic processes have no sub-processes and are executed in a single step, from the service requester perspective. Simple processes are not invocable and are not associated with a grounding. Simple processes are used as elements of abstraction; a simple process may be used either to provide a view of (a specialized way of using) some atomic process, or a simplified representation of some composite process. Composite processes are decomposable into another (non-composite or composite) processes; their decomposition can be specified by using control constructs such as Sequence and If-Then-Else, Split, Split-Joint. [12].

#### 3.1.3 Service Grounding

The grounding of a service specifies the details of how to access the service – details having mainly to do with protocol and message formats, serialization, transport, and addressing. Grounding can be thought of as a mapping from an abstract to a concrete specification of these service description elements that are required for interacting with the service – in particular, the inputs and outputs of atomic processes. The grounding defines mapping rules to link OWL-S atomic processes to WSDL operations.

#### 3.2 Discovery

Discovery is the process of finding Web services with a given capability. Most of the current service discovery approaches perform syntactic matching, which searches for service descriptions that contain particular keywords from the users query and this often leads to poor discovery results [2].

In semantic service discovery, a domain specific ontology is used to describe service providers and requesters. The results from the discovery process tend to be more accu-
rate than the results from simple keyword matching discovery [13]. The degree of matching is decided by whether the matching between concepts is exact or subsumes with respect to some similarity metric(s). It allows a ranking of discovered services such as: exact, plugin, subsumes, subsumed-by, fail. Where exact means that matches exactly and fail means that does not matches.

4. SECOAS
SeCoAS, Semantic and Context-Aware Available Services, combines some context modeling ontologies (Section 2) with the OWL-S service ontology (Section 3). As result, we can model the services in our context-aware environment as a service semantically described.

In a dynamic environment, it may be necessary to discover what are the devices available for the provision of a given service. Semantic Discovery permits to locate the services and their functionalities. In order to locate services, SeCoAS uses semantic matching. Once a service is located, the environment adapts to use that configuration.

4.1 Defining the Contextual Parameters
Contextual information is modeled using the ontologies described in Section 2. This information, in SeCoAS, serve as Input for the Profile and Process. The inputs are modeled as parameters.

A parameter has a parameterType property, which permits the specification of the type of a parameter. SeCoAS uses this property to define that a parameter has a type given by context ontology. An Input is a subclass of Parameter. By using this property, we can define an Input having a type of context ontology. Lines 1 to 3 in Document 5 show an example of an Input instance. This instance (PersonName) has the type Name, which is defined in the Person ontology (see Section 2.1).

```xml
0 <!-- Document 5 -->
1 2 <!-- Contextual Parameters as Input for the Process -->
2 3 <!-- A Input instance in Process -->
4 <process:Input rdf:ID="PersonName"/>
5 <process:parameterType rdf:resource="#person#Name"/>
6 </process:Input>
```

All parameters in OWL-S are defined in the Process ontology and can be used by the Profile ontology (as IOPE’s) to advertise the service. SeCoAS is according with this rule. We define all parameters in the processes of our context-aware services and we use some of these parameters to advertise the service by the service profile.

4.2 Defining the Service Workflow
IOPE’s are defined as contextual parameters for the SeCoAS as described in the previous section (Section 4.1). These parameters are used in the execution of some context-aware services. These services, in SeCoAS, are modeled as an OWL-S process.

In the SeCoAS, the context-aware service workflows are modeled as a process, which may be atomic or composite. The control constructors of the Process ontology are used in SeCoAS to model the service behavior. In such a way, we define the service workflow by using control constructs like Sequence, If-Then-Else, Split, Split-Join, While, Choice, and so on.

4.3 Matching in SeCoAS
The matching algorithm used in SeCoAS compares services that have similar contextual parameters. The algorithm executes the matching in the service providers (P) description against the service requester (R) descriptions. The matching is performed over four types of concepts: service category, service outputs, service inputs and contextual parameters.

The matching algorithm has three phases: 1) matching of service category: filter out those services that are not the desired service type provided in the user request; 2) matching of services outputs: filter out all service descriptions that do not have the desired service output; 3) matching of services inputs: use all inputs explicitly informed by the user and all inputs that may be obtained from context.

SeCoAS matching is based on Description Logics (DL) described by Li and Horrocks [10]. For each phase is performed a DL subsumption matching. A DL reasoner can check whether two concepts subsume each other.

4.4 Implementation
We use Jena Semantic Web API\(^7\), which is a Java framework for building Semantic Web applications, and Sparql\(^8\), which is query language for RDF.

The tool used to create the OWL documents was Protége\(^9\). The OWL-S editor [15] was implemented as a Protége plugin, that we used to write the service descriptions in OWL-S.

The reasoner used is the Pellet DL reasoner [17]. The SWRL (Semantic Web Rule Language)\(^9\) was proposed by W3C to express these rules. At the time of implementation of this work, the Pellet DL reasoner did not offer support to SWRL. For this reason, we performed rule reasoning using Jena Generic Rule Reasoner.

5. A CASE STUDY
In this section we describe a case study using SeCoAS and the Context Ontologies described in Section 2. Both, the services and the context information are modeled with ontologies. This enables composition and discovery of the services on the fly. These services evaluate the current context in order to verify if changes are needed to provide the service as the user requires.

5.1 Scenario Description
The scenario is Home Domain. It is supposed to be equipped with networked ubiquitous devices. The front door is equipped with devices to recognize the presence of people and a home intercom. The home intercom provides multimedia information with sound and video and it is able to find a home resident and establishes a communication with him with the device that the home resident is using (PDA, cellphone, notebook, etc) and with application that support this device.

A possible scenario is: "Alex goes to visit his friend Carlos. When Alex is in front of Carlos’ Home the service Check Authorization verifies if he is authorized to enter. The system verifies that Alex is not a family member and asks him the name of a person who can authorize his entry. Using the service Request Permission, the system obtains information about Carlos and finds out that Carlos is going home,

\(^7\) http://jena.sourceforge.net/documentation.html
\(^8\) http://www.w3.org/TR/rdf-sparql-query/
\(^9\) http://www.w3.org/Submission/SWRL/
driving his car and carrying in his cell phone). A communication is automatically established (by the home intercom) between Alex and Carlos. Finally, Alex obtains the permission and waits for Carlos arrival. The door is opened and a welcome message is sent. Finally, the system chooses the music he likes, his TV favorite programs, the temperature and illumination pleasant to him.

The next sections explain the first two services: Check Authorization (Service 1) and Request Permission (Service 2). Set Home Environment (Service 3) has a similar structure.

5.1.1 Service 1: Check Authorization

This service has the precondition that someone be near the front of home and has as result that this person is authorized. When the person is authorized the front door can be opened.

When a person wants to enter home, this person is identified by a device attached in front of the door such as a camera or a RFID reader that returns and ID: the IDPerson.

"When the system recognizes that exists someone near the front door, it verifies if the person is authorized to enter home by getting the context parameters Person, Space, and Actual Time, which are the inputs required to check the authorization. As result, there are three options: i) The person is an authorized person: the system sends to the user a welcome message with the person’s name and the door is automatically opened; ii) The person is not an authorized person: she can request a permission from a resident, activating the service Request Permission; iii) The person is not authorized and a message is sent to leave the home”.

5.1.2 Service 2: Request Permission

The Request Permission service has the precondition that someone requests a permission to enter home. The result is that the person obtains or do not obtains this permission.

“The system obtains further information, which include the current location of a resident (a person that can give an authorization), the status of this person (if she is working, driving a car, etc.) and the device that she is using. According to these information, if the conditions to establish the communication are satisfied, the person that requested authorization talks with the person that can authorize her. As result, the system receives the authorization with some polices about what spaces this person can enter, the devices that she can use and the time interval which can be resident”.

5.2 Context-Aware Services

Section 5.1 describes a scenario of a context-aware environment that is a home. In this scenario we define two services (Sections 5.1.1 and 5.1.2). In this section we describe the service CheckAuthorization that is defined in Section 5.1.1.

Document 6 shows the service CheckAuthorization described in OWL-S. The service is modeled as proposed in SeCoAS, Section 4, where we define a service having a profile, a process and a grounding.

Figure 4: Service 2: Request Permission. This service allows that someone, which arrives to home front door, gets a permission to enter

Figure 4 shows the service description Request Permission. This service has a main process Get Permission that is composed of tree process: i) Get Context Parameters, which obtains information about the person that can authorize, such as the person location, the status and devices that she is using; ii) Establish Communication, which enables a communication between the person that requests authorization and the person that can authorize; iv) Send Message, which sends a message to the Service 1 (Check Authorization) about the result of the authorization.
The CheckAuthorization service description shown in Document 6 has the three parts of the OWL-S: ProfileCheckAuthorization (line 5), AuthorizePerson (line 8) and GroundingCheckAuthorization (line 11).

The following sections detail how we integrate contextual parameters to the service CheckAuthorization (Section 5.2.1) and how we describe the service workflow using the concept of process in OWL-S (Section 5.2.2).

5.2.1 Contextual Parameters: Check Authorization Service

The CheckAuthorization service needs to know (informed by the context-aware environment) about the person (who wants to change of space), the space (where the person wants to go), the time (when the person is trying to enter the space) and the time interval (when the person has permission to enter in the space). This information is about the context dimensions: what, who, where and when [1].

Document 7 has the profile (ProfileCheckAuthorization) of the CheckAuthorization service. This profile is formed by four inputs, one precondition, two effects and one output.

The Profile ontology does not provide a schema to describe IOPE instances. However, such a schema is defined in the Process ontology and used by the Profile ontology.

OWL-S can use domain ontologies to define the inputs, outputs, preconditions and effects of a service. In SeCoAS (Section 4.1), we define that all contextual parameters are defined as Input parameters in the process by using the property parameterType.

Document 8 shows that: the parameter type of the input PersonAllowed_EnterHome is Person (see line 4), which is defined in the Person ontology.

The inputs AccessTime and AllowedIntervalTime in Document 8 need some special attention. These inputs were created using the Time ontology (see Section 2.2) with a restriction that we defined and that is shown in Document 9 at lines 6 to 9. This restriction was made because the AccessTime needs to be contained in the AllowedIntervalTime. In other words, the instant that a person wants to enter in a space needs to be an instant that this person has permission to enter in that space.

We have defined the inputs, outputs, preconditions and effects for the CheckAuthorization service. However, to simplify our case study in this paper, Document 9 shows only the inputs that uses external concepts from contextual ontologies.

5.2.2 Service Workflow: Check Authorization Service

As shown in Document 6, CheckAuthorization service workflow is described by a process named AuthorizePerson. Figure 3 shows the AuthorizePerson process and their components that compose the CheckAuthorization service. In this section we describe the workflow (as proposed in SeCoAS, Section 5.2.2) of the CheckAuthorization service by detailing their composing processes.

The AuthorizePerson process, which is shown in Document 10, is a composite process that is composed by a sequence of two process: GetContextParameters and SendMessage. The AuthorizePerson process has inputs that are contextual parameters defined in Section 5.2.1.
The processes that compose AuthorizePerson are an atomic process and a composite process. The atomic process SendMessage is a process that has some message as an output. This message can be interpreted, for example, as an input for another service that presents the message as a sound. The composite process GetContextParameters is the core of the CheckAuthorization service.

Document 11 shows the OWL-S descriptions for the composite process GetContextParameters. This process is composed by three atomic processes within the OWL-S constructor Split-Join (lines 9 to 17). Split-Join construct permits that the three atomic processes are executed concurrently and, at the end, join into a sequence for the SendMessage process (see Figure 3).

The concurrent processes CheckPerson, CheckSpace and CheckTime check if a person has authorization to enter in a space in the current time.

6. RELATED WORKS

The Context Broker Architecture (CoBrA) [3] is an agent architecture that uses ontologies specified in OWL to model and to reason in intelligent environments. Their context model includes vocabularies for person, places, time, device and privacy policies.

The CONtext ONtology (CONON) [21] provides an upper context ontology that captures general concepts about basic context, and a domain-specific ontology. A similar approach is taken by Bulcão et al. [5], who described a domain-independent semantic model and exploit the following context dimensions: identity (who), location (where), time (when), activity (what) and devices profiles (how).

Broens et al. [2] present an approach to use SWS and context information. Their approach focuses on the description of a matching algorithm that uses contextual attributes: the authors do not discuss the integration of SWS with context information nor the modeling of services related to Pervasive Environments.

The Context Aware Service Oriented Architecture (CASSOA) [4] presents an approach to the description of services, but their context information modeling is restricted to the description of service requesters and the service providers.

Mokhtar et al. [14] model services and user tasks as semantic Web services in OWL-S extended with context information. The authors present an approach to the dynamic, context-aware composition of services to perform user modeled tasks.

Works of the aforementioned authors can be divided into two groups: one that is concerned with modeling contextual information and reasoning with contextual information [3, 21, 5]; and another that is concerned with the use of SWS applied to the development of Pervasive Environments [2, 4, 14]. However, these authors do not investigate the problem of integrating contextual information with semantic web services, which is the main contribution of the work reported in this paper.

In a previous work [16], some authors of this paper have proposed supporting remote experiments as Semantic Web Services. They use ontologies to formalize experiments as Semantic Web Service. The aim is to promote the standardization of tasks such as experiment discovery, invocation, composition and monitoring as well to automate these tasks.

7. FINAL REMARKS

We present an approach to the development of Semantic and Context-Aware Available Services: SeCoAS. We have outlined the results and applications of Semantic Web Services Modeling and Reasoning for Context-Aware Environments. SeCoAS integrates context-related ontologies, such as Time, Person, Space, and Device, with the OWL-S ontology (Semantic Markup for Web Services). These ontologies were used as parameters to the context-aware services.

OWL-S provides three types of knowledge about the service. Service Process gives details about how the service operates. One of the three types of Service Process is the Composite Process, which defines some control constructors as Sequence and If-Then-Else, Split, Split-Joint. We use these constructors on the definition of complex behaviors (workflows) of Pervasive Environments.

The approach proposed and presented in this paper was discussed in a scenario of a context-aware environment that describes some services about an intelligent home. We detailed the integration of semantic web services with context ontologies. SeCoAS combines the efforts on modeling context information with SWS modeling in OWL-S: the aim is to allow tasks such as automatic matching and composition.

The matching algorithm we present makes use of contextual parameters. As future work, we plan to implement the semantic matching and composition of services in context-aware environments. We also plan to develop an adaptation mechanism that allows the modification of the rules that define the behavior of the environment.

Acknowledgments

We would like to thank FAPESP for supporting the authors in the context of the TIDIA-Ae Project. We also thank the following organizations for their support: CAPES and CNPq, for supporting the Graduate Program in Computer Science at the Universidade de Sao Paulo and Edgardo Escobedo (CNPq), Cássio Prazeres (CAPES) and Maria da Graça Pimentel (CNPq).

8. REFERENCES


