GAS Ontology: an ontology for collaboration among ubiquitous computing devices

Eleni Christopoulou, Achilles Kameas

Research Academic Computer Technology Institute, Research Unit 3,

Design of Ambient Intelligent Systems Group

61 Riga Feraiou str. 26221, Patras, Greece

{hristope, kameas}@cti.gr

Abstract

The vision of ubiquitous computing is that the addition of computation and communication abilities to the artifacts that surround people will enable the users to set up their living spaces in a way that will serve them best minimising at the same time the required human intervention. The ontologies can help us to address some key issues of ubiquitous computing environments such as knowledge representation, semantic interoperability and service discovery.

The GAS Ontology is an ontology that was developed in order to describe the semantics of the basic concepts of a ubiquitous computing environment and define their inter-relations. The basic goal of this ontology is to provide a common language for the communication and collaboration among the heterogeneous devices that constitute these environments. The GAS Ontology also supports the service discovery mechanism that a ubiquitous computing environment requires.

In this paper we present the GAS Ontology as well as the design challenges that we faced and the way that we handled them. In order to select the language and the tool that we used for the development of the GAS Ontology, we designed a prototype ontology and evaluated a number of languages and tools. The ontology development tool that proved to be the most suitable from this evaluation was Protégé-2000. We also present how we use the GAS Ontology in our eGadgets project achieving semantic interoperability and service discovery. Finally, we present the GAS Ontology manager, which runs on each device, manages the device's ontology and processes the knowledge that each device acquires over time.

1. Introduction

Nowadays, people build artifact “ecologies” in their living spaces, by selecting objects and then arranging them in ways that best serve specific activities. A limitation of the current technology is that it requires human intervention, in order to enable the communication and collaboration among the artifacts. The vision of ubiquitous computing is that the addition of computation and communication abilities to the artifacts that surround people will enable the users to set up their living spaces in a way that will serve them best minimising at the same time the required human intervention.

This paper presents research that has been carried out during the eGadgets project, a research project funded in the context of EU IST/FET proactive initiative “Disappearing Computer”. The goal of this project was to deliver an architectural framework that supports the composition of ubiquitous computing systems, the Gadgetware Architectural Style (GAS). A key issue in the project domain is the heterogeneity of the devices that constitute the “ecologies”, as these devices do not form a closed and coherent set and may come from various manufacturers. Therefore a primary requirement is the description and definition of the basic concepts of these “ecologies”. This representation of the basic concepts and their inter-relations will provide a common language, which will support the semantic interoperability among the heterogeneous devices as well as the communication and collaboration among them. Another important requirement is that this common language must be flexible and extensible so that new concepts can be added and represented. The GAS Ontology was developed in order to accommodate these issues by representing and providing this common language. An earlier version of this work was presented in Christopoulou and Kameas (2003).

1.1. Basic concepts

The basic concepts encapsulated in the Gadgetware Architectural Style, according to Kameas et al. (2003), are:

- eGadgets (eGts): are everyday physical objects enhanced with sensing, acting, processing and communication abilities. Moreover, processing may lead to intelligent behavior, which can be manifested at various levels. eGts can be regarded as artifacts that can be used as building blocks to form GadgetWorlds, with the support of Gadgetware Architectural Style (GAS).

- Plugs: are software classes that make visible the eGt capabilities to people and to other eGts. They may also have tangible manifestation on the eGt's physical interface, so that users can utilize them in forming synapses.
- Synapses: are associations between two compatible plugs.
- eGadgetWorlds (eGWs): are dynamic, distinguishable, functional configurations of associated eGts, which communicate and / or collaborate in order to realize a collective behaviour. eGWs are formed purposefully by an actor (user or other) and appear as functionally unified entities.

The basic concepts defined above are illustrated in Figure 1 through a simplified scenario. Two eGts (eBook and eLamp) are connected through a synapse which is established between two plugs forming a study eGW. When the user opens the eBook, the eLamp switches on, adjusting the light conditions to a specified luminosity level in order to satisfy the user's profile. Each eGt operates independently of the other. Two plugs are being used in this scenario: open/close reflecting the state of the book and switch on/off reflecting the state of the lamp. Although not obvious, these plugs are compatible. Thus, an end-user can compose a simple eGW simply by establishing a synapse between these two plugs. Different users may have access to this "application" each one having defined his/her own study profile.

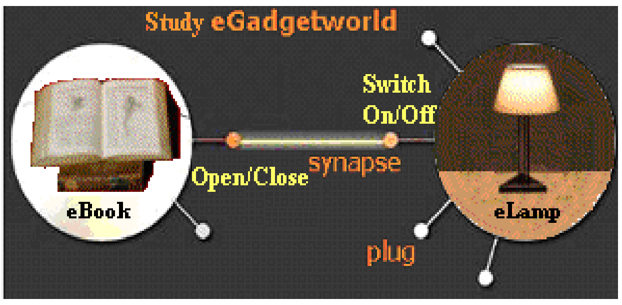


Figure 1. A study eGadgetWorld realized as a synapse between plugs

1.2. Structure of the paper

The contribution of this paper is to present the GAS Ontology that provides a common language for the communication and the collaboration among ubiquitous computing devices, mentioned as eGts, and the GAS Ontology manager that supports this collaboration. The rest of the paper is organised as follows. Section 2 outlines the reasons that led to the necessity of using the GAS Ontology into a ubiquitous computing environment. Section 3 describes the basic concepts of the GAS Ontology and presents the way that this ontology was designed regarding the design challenges that were handled. Section 4 presents the procedure of the GAS Ontology development. In this section the language and the tool that were used for the development of the GAS Ontology are presented as well as the methodology that was followed in order to select this language and tool. In this section is also presented the lifecycle of the creation of the GAS Ontology. Section 5 describes the GAS Ontology manager that was developed in order to provide the necessary mechanisms for the management of this ontology as well as to support the collaboration among eGts. Section 6 describes the use of the GAS Ontology through an example based on the aforementioned scenario of the study eGW. In section 7 related approaches for ontologies in ubiquitous computing environments are presented. The paper closes with the conclusion and an outlook on future work in section 8.

2. Need for GAS Ontology

In the eGadgets project we designed the Gadgetware Architectural Style (GAS) as an architectural framework that supports the composition of ubiquitous computing systems. During the development and deployment of ubiquitous computing environments using GAS, a number of key issues came along. The first one is the heterogeneity of the eGts and the demand for semantic interoperability among them. Another important issue is the dynamic nature of the ubiquitous computing environments, as the eGWs are not static and the eGts that constitute them are exposed to damages. The last issue is the necessity of a service discovery mechanism that is able to answer the eGts' queries for specific services provided through plugs. Following we describe these issues and explain our decision to use ontologies in order to address them as well as the necessity of conceptualising the new terms defined by the GAS.

2.1. Conceptualisation of eGadgetWorlds

The GAS constitutes a generic framework shared by both artifact designers and users for consistently describing, using and reasoning about a family of related eGWs. GAS defines the concepts and mechanisms that will allow people (eGadget users) to define, create eGWs out of eGts and use them in a consistent and intuitive way. As the eGts are everyday tangible (physical) object enhanced with sensing, acting, processing and communicating abilities the eGadget users do not have to deal with unfamiliar to them objects, but merely to view their world from a different perspective and get familiar with its enhanced concepts. This new world view is constituted of the set of basic terms, their definitions and their inter-relations, which are defined by the GAS.

The necessity of capturing and representing this knowledge is evident, as the deployment of the eGWs is based on this knowledge. As an ontology, according to Uschold and Gruninger (1996), is a tool that can conceptualise this world view by capturing general knowledge and providing basic notions and concepts for basic terms, we decided to use ontologies in order to conceptualise the terms of the eGWs. The ontology that we developed is the GAS Ontology and its first goal was the description of the semantics of the basic terms, such as eGadget, Plug, Synapse, eGadgetWorld, and the definition of the relations among them.

2.2. Semantic interoperability among eGadgets

The heterogeneity of the devices is a major issue in eGWs, as with all ubiquitous computing environments. The heterogeneity of the eGts is due to the facts that the same type of eGts, e.g. eLamps, can be created from various manufactures and that some eGts may provide different capabilities unknown to the other eGts. Nevertheless, since an eGW is a specific configuration of associated eGts, which communicate and / or collaborate in order to realize a collective behaviour, the need to support interaction among autonomous and heterogeneous eGts is evident.

In our approach and in order to present the autonomous nature and function of each eGt, we chose to base this interaction on well-defined and commonly understood concepts, so that the eGts can communicate with each other in a consistent and unambiguous way. In order to enable this interoperability the eGts have to use the same language and a common vocabulary (although each may implement a different mechanism to interpret them). The GAS Ontology can provide this necessary common language for the communication and collaboration among eGts as it describes the semantics of the basic terms and defines their inter-relations. An important aspect is also the demand that this common language must be flexible and extensible so that new concepts can be added and represented. The GAS Ontology was designed and developed in order to accommodate this issue.

2.3. Dynamic nature of eGadgetWorlds

One of the most important features of eGWs is that they are created in a dynamic way. Users are permitted to create and delete synapses between eGts whenever they want without restrictions. It may be also allowed that eGWs can form synapses between eGts' plugs automatically, without user intervention. It is evident that the creation of synapses requires some form of plugs compatibility check. Plugs compatibility is determined by several factors, e.g. the type of input that they accept and the type of output that they produce. Therefore, in order to fully utilize the automatic synapse formation, we need a representation of the formal rules that determine the compatibility of two plugs. Ontologies can be used in order to represent this kind of formal rules. Thus we need GAS Ontology in order to represent the formal rules that handle the compatibility of two plugs.

The dynamic nature of eGWs depends also on eGts mobility, as this feature can cause the dynamic disestablishment of a synapse. For example this may happen when two eGts move outside of each other's range. The same effect may appear when an eGt suddenly “disappears” due to low battery or other failure. So the critical issue that arises is what should happen when an eGt that participates in a synapse suddenly disappears. A simplistic and straightforward approach would be to ignore the eGt that its functionality fails. Since our vision refers to “smart” eGWs and eGts that exploit the knowledge that they have acquired by experience, the automatic replacement of a failed eGt is desirable. Thus a number of questions must be answered in a formal way, in order to ensure eGts replacement feasibility; e.g. when is an eGt qualified as capable to replace another one, does it suffice that the two eGts have the same physical properties or must they provide the same capabilities through their plugs, etc. So we have to know not only that an eGt can replace another one, but the degree to which the two eGts are “identical”. So we need GAS Ontology in order to contain the rules that support the eGt replacement feasibility.

2.4. Semantic service discovery

The plugs are software classes that make visible the eGts' capabilities to people and to other eGts. The term that we use for these capabilities is Services. For example the eGt “eLamp”, Figure 1, through the Plug “switch on/off” provides the Service “light”. The concept of services in eGWs is a basic one, as services can play a major role when determining eGts' replaceability and plugs' compatibility, since for example we assume that an eGt A participating in Synapse S with Plug P can be replaced by another eGt B that that provides a service P' similar to P. Furthermore, an eGadget user forms synapses seeking to achieve certain service configurations; thus a service discovery mechanism is necessary. So for a ubiquitous computing environment this mechanism must be enhanced to provide a semantic service discovery. This means that it must be possible to discover all the relevant services. Thus for the deployment of eGWs we need the semantic description of the concept Service as well as a service classification that assists semantic service discovery. The GAS Ontology contains this knowledge, too.

3. GAS Ontology design

The basic goal of the GAS Ontology is to provide the common language for the communication and / or collaboration among the eGts. Thus it must contain the semantic description of the basic terms of eGWs and their inter-relations. Additionally the GAS Ontology may support the building of eGWs by providing specific rules for plugs compatibility and eGts replacement feasibility. As a semantic service discovery mechanism is desirable the GAS Ontology must represent the suitable knowledge, such as the description of various services provided by the eGts. In the following we describe the way that we designed the GAS Ontology in order to accommodate the aforementioned issues.

3.1. Ontology layers

Each eGt must have an ontology that contains the description of the basic concepts of eGWs and their inter-relations. This knowledge must be common for all eGts in order to support the communication among them. On the other hand the ontology of an eGt must describe the way that this eGt is used and represent its acquired knowledge. Specifically an eGt's ontology must represent its description and the descriptions of its plugs and services. The knowledge emerged from the synapses that an eGt's plugs participate must be also represented into the eGt's ontology. Thus the knowledge that each eGt's ontology represent cannot be the same for all eGts, as it depends on the eGt's description and on the eGWs that the eGts has participated in the past.

The fact that each eGt's ontology is different could result to ineffective communication among eGts, since it is based on their ontologies. An awkward solution to this problem could be the merging of all existing eGts' ontologies in a global one. This solution would inevitably result into a very large knowledge base. There are two issues that make this solution undesirable. The first one is that this solution does not respect the limited memory capabilities of the eGts. The second one is that this solution in order to work properly assumes that all eGts ontologies are synchronized. Another solution could be the use of a server into which all eGt's ontologies are stored and each eGt can have access to it in order to get a new ontology. This solution conflicts with the autonomy of eGts and the eGadgets project's demand of the non-existence of a server.

The solution that we selected allows each eGt to have a different ontology with the condition that it all ontologies are based on a common vocabulary. Thus, although all eGts have different knowledge it is represented with terms and concepts common to all the eGts. According to this solution the GAS Ontology is divided into the following two layers: the GAS Core Ontology (GAS-CO) and the GAS Higher Ontology (GAS-HO). The GAS-CO provides eGts with the necessary common language that they need in order to describe their acquired knowledge represented by the GAS-HO. A representation of this is on Figure 2.

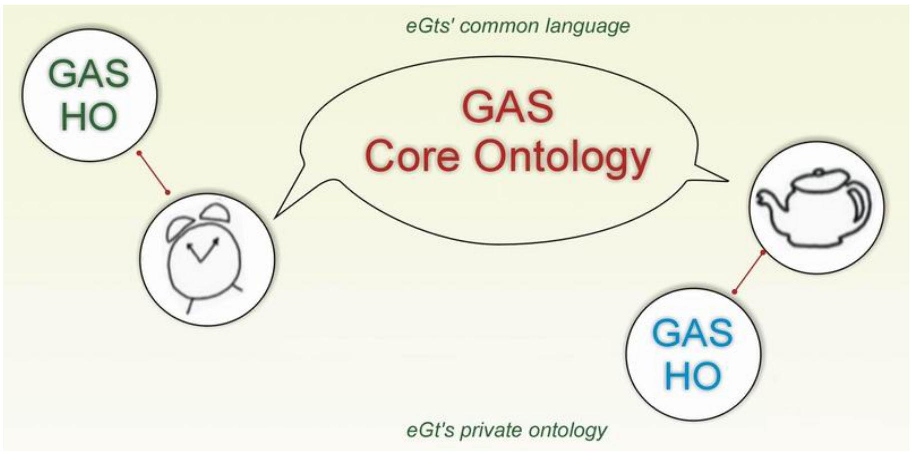


Figure 2. The GAS Ontology layers

3.2. GAS Core Ontology (GAS-CO)

The GAS-CO is the common language of eGts, so it must describe the semantics of the basic terms of eGWs and define their inter-relations. It must also contain the service classification in order to support the service discovery mechanism. Furthermore it has to define the rules for plugs compatibility and eGts replaceability.

As the GAS-CO describes the language that eGts use to communicate and/or collaborate, it must be the same for all eGts. The major goal of GAS-CO is to contain only the necessary information in order to be very small. Thus even eGts with limited memory capacity can store it. A key issue is that the GAS-CO cannot be changed either from the manufacturer of an eGt or from an eGadget user, so it is static.

Following we describe how the basic terms of the eGWs and their inter-relations are represented in the GAS-CO. A graphical representation is on Figure 2.

The core term of GAS is the eGt. In GAS-CO the eGt is represented as a class, which has a number of properties. As GAS-CO must provide a complete representation of eGt, it describes the eGt's properties. These properties are the following: the physical properties, which describe the eGt as a tangible object, e.g. shape, material and colour, the digital properties, which manifest the digital self of the eGt, e.g. memory configuration, processing capabilities and communication interface and the plugs that are owned by the eGt and expose its services.

The notion of plug is represented in the GAS-CO as another class, which is divided into two disjoint subclasses; the TPlug and the SPlug. The TPlug describes the physical properties of the object that is used as an eGt (i.e. dimensions, shape, colour, etc) and lists all the eGt's Plugs and Synapses; there is a cardinality restriction that an eGt must have exactly one TPlug. On the other hand an SPlug represents the eGt capabilities; an eGt can have an arbitrary number of SPlugs.

Another GAS-CO class is the synapse that represents a synapse among two plugs. A synapse may only appear among two SPlugs. The cardinality restriction that stands for a synapse is that it represents the connection among only two SPlugs. For each SPlug that participates in a synapse, a special role can be declared through GAS-CO.

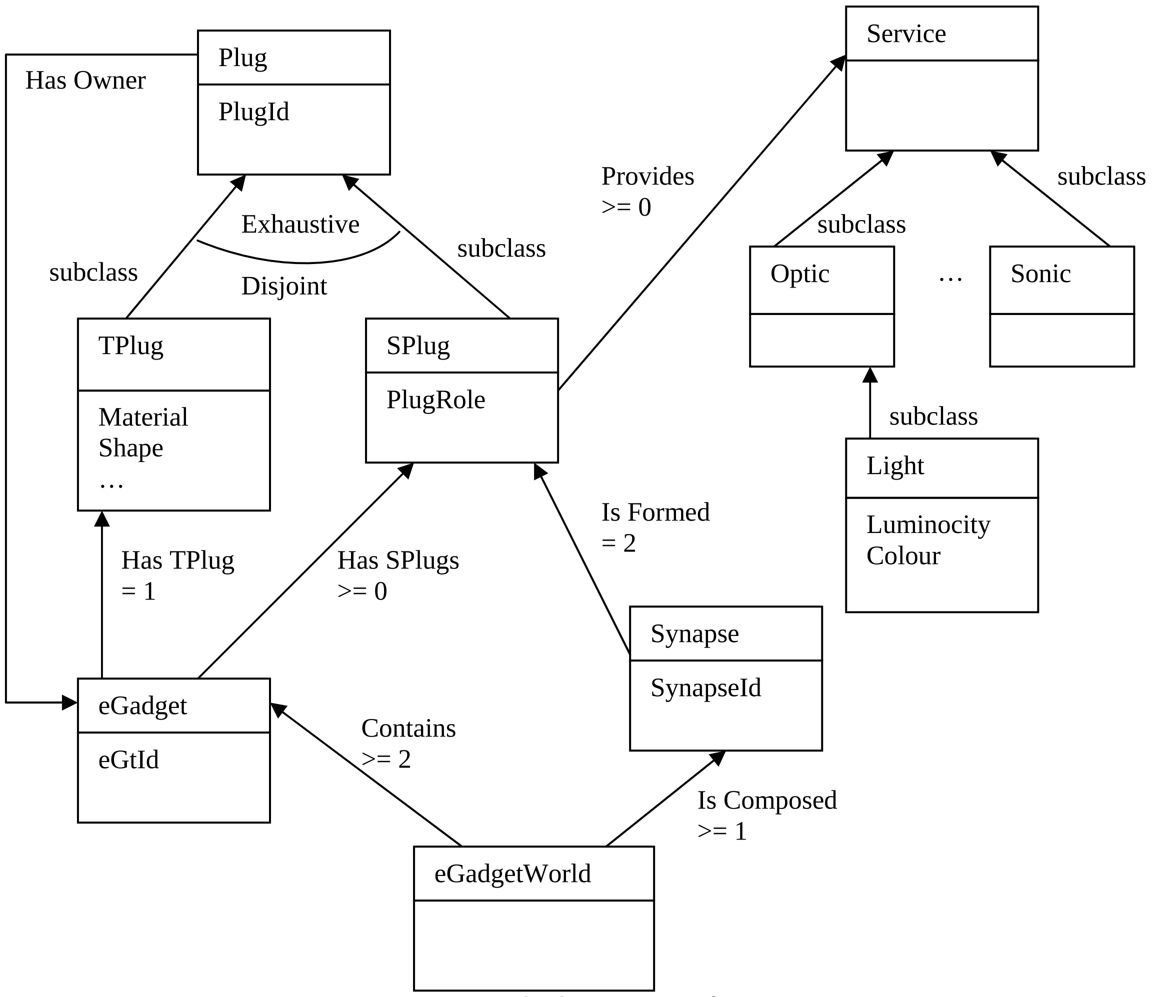


Figure 3. A graphical representation of GAS-CO

Using the class of eGW the GAS-CO can describe the eGWs that are created by the eGadget users. The eGW properties represent the eGts that an eGW contains and the synapses that compose it. In this class there are two cardinality constraints, as an eGW must contain at least two eGts and a synapse must exist among them.

This project assumes that an eGt through an Splug provides a number of services. Services are related to what an eGt's actuators/sensors can transmit/perceive. Thus our goal is to describe a service classification, which may be based on the type of the signals that an actuator/sensor transmits/perceives.

There are some elementary forms of signals like: electric, electromagnetic, gravity, kinetic, optic, thermic and sonic. The GAS-CO contains a class for the notion of service. The class service is divided into subclasses, each of which corresponds to one of the above elementary forms of signals. A service can be further refined into higher level services: e.g. the optic service can be refined into light, image, etc. Additionally a service may have a set of properties; e.g. light can have colour, luminocity, etc.

3.3. GAS Higher Ontology (GAS-HO)

The GAS-HO represents both the description of an eGt and its acquired knowledge. These descriptions follow the definitions contained in the GAS-CO. Thus the knowledge represented by GAS-HO is described as instances of the classes defined into the GAS-CO. For example the GAS-CO contains the definition of the concept SPlug, while the GAS-HO contains the description of a specific SPlug represented as an instance of the concept SPlug. Note that the GAS-HO is not a stand-alone ontology, as it does not contain the definition of its concepts and their relations.

Since the GAS-HO represents the private knowledge of each eGt, it is different for each eGt. Therefore we can envision GAS-HO as eGt' s private ontology. Contrary to GAS-CO, which size is required to be small enough, the size of GAS-HO can be "unlimited" and depending on eGt's memory capacity. Obviously GAS-HO is not static and it can be changed over time without causing problems to eGts communication. As the GAS-HO contains both static information about the artifact and dynamic information emerged from its knowledge and use, we decided to divide it into the GAS-HO-static and the GAS-HO-volatile.

3.3.1. GAS Higher Ontology – static (GAS-HO-static)

The GAS-HO-static represents the description of an eGt containing information about eGt's plugs, the services that are provided through these plugs, its sensors and actuators, as well as its physical characteristics. The knowledge represented by the GAS-HO-static is static as the description of an eGt is permanent. The description of an eGt is presented as a number of instances of the concepts that are defined in GAS-CO. For example, the GAS-HO-static of the "eLamp" eGt contains the knowledge about the physical properties of "eLamp", such as its luminosity, the description of its SPlug "switch on/off" based on the definition provided by GAS-CO, as well as the declaration that the SPlug "switch on/off" provides the service "light".

3.3.2. GAS Higher Ontology – volatile (GAS-HO-volatile)

The GAS-HO-volatile contains information derived from the eGt’s acquired knowledge and its use. Specifically it describes the synapses which the eGt’s plugs are connected to, the eGWs which it takes part to, as well as information about the capabilities of other artifacts that has acquainted through communication. An eGt’s GAS-HO-volatile is updated during the eGt’s various activities. For example, when the SPlug “open/close” of the “eBook” establishes a Synapse with the SPlug “switch on/off” of the “eLamp”, both these eGts store the description of this Synapse into their GAS-HO-volatile. Although the GAS-HO-static and the GAS-HO-volatile contain different knowledge, both of them are based on the GAS-CO and contain instances of its concepts.

4. GAS Ontology development

In order to implement GAS Ontology it was necessary to select a language capable to represent both the concepts of this ontology and its rules, as well as an appropriate tool to assist its development. The methodology followed to select the language and the tool that would be used, was based on the use of a prototype ontology that served as a benchmark to evaluate each of candidate languages and tools; this methodology is described in Christopoulou (2003). This prototype ontology was a subset of GAS-CO along with a set of simple instances. The goal of this process was not to identify the best language or tool for the development of ontologies, but merely to propose the most appropriate ones for this application.

4.1. Languages evaluation

A number of criteria appropriate to evaluate ontology languages was located in bibliography and was used during the language evaluation process. The bibliography sources that were used are the Corcho and Gómez-Pérez (2000), Gil and Ratnakar (2002), Gómez-Pérez and Corcho (2002), and Ribière and Charlton (2000). The selected criteria aim to expose the capabilities of ontology languages and can be divided into two categories; the first one being the expressiveness of each language and the second one evaluates the inference mechanism that each language may provide. For example the expressiveness of a language is measured through criteria that evaluate how the concepts of an ontology and their properties are described, what types of taxonomies are allowed, whether axioms can be defined, etc. Also, there are criteria that evaluate the existence of an inference mechanism for a language, as well as criteria that evaluate whether an inference mechanism is sound and complete or if it performs constraint checking.

The prototype ontology was designed to be minimal but adequate at the same time, covering the criteria that the final GAS Ontology would cover.

The languages that were evaluated with the prototype ontology belong to various categories. For example the traditional languages of knowledge representation that were evaluated are the following: CARIN, Flogic, KIF, Loom, OCML, OKBC and Ontolingua. Additionally the following web-based ontology languages were evaluated: XML, SHOE, XOL, OML, RDF(S), OIL, DAML+OIL and OWL. Finally languages like CycL, GRAIL, NKRL and PIF, which were developed for specific application, were evaluated, too.

The language that proved to be most suitable was DAML+OIL, since it covers the most important criteria for this application. DAML+OIL, described in van Harmelen et al. (2001), is a semantic markup language for Web resources, which is suitable for the communication among agents. A key issue in agents' communication is that they exchange pieces of information exploiting the common language that they share; this is a feature that is preserved in the eGts. DAML+OIL provides a very simple notion of context, using XML namespaces. This property is important for GAS Ontology, where the same term may be defined differently in alternative contexts, e.g. a plug may provide different services to various eGadget users and eGWs. Another powerful feature of this language is its capabilities for describing concepts, their properties, as well as constraints for these properties. For example in GAS Ontology it is necessary to define the domain and the range of concepts' properties. Also, a number of cardinality constraints appears in GAS Ontology, e.g. an eGt must have exactly one Tplug. DAML+OIL has powerful taxonomy capabilities, since it enables disjoint and exhaustive decompositions, as well as multiple inheritance. In the GAS Ontology these features are necessary, as for example the decomposition of the concept plug to the concepts Splug and Tplug is both a disjoint and an exhaustive one. GAS Ontology contains various relations, e.g. an eGt has Splugs, which can be described with DAML+OIL. Furthermore with the DAML+OIL the definition of instances is feasible. Another important characteristic of the DAML+OIL is that it enables the description of first order logic axioms. This is important in the GAS Ontology, where rules, such as the id of each eGt in a specific eGW is unique, appear. Finally the fact that the DAML+OIL allows the existence of documentation into the description of concepts, properties, etc. is useful. For example in the GAS Ontology a description of the functionality of a synapse in a natural language is useful for the eGadget users.

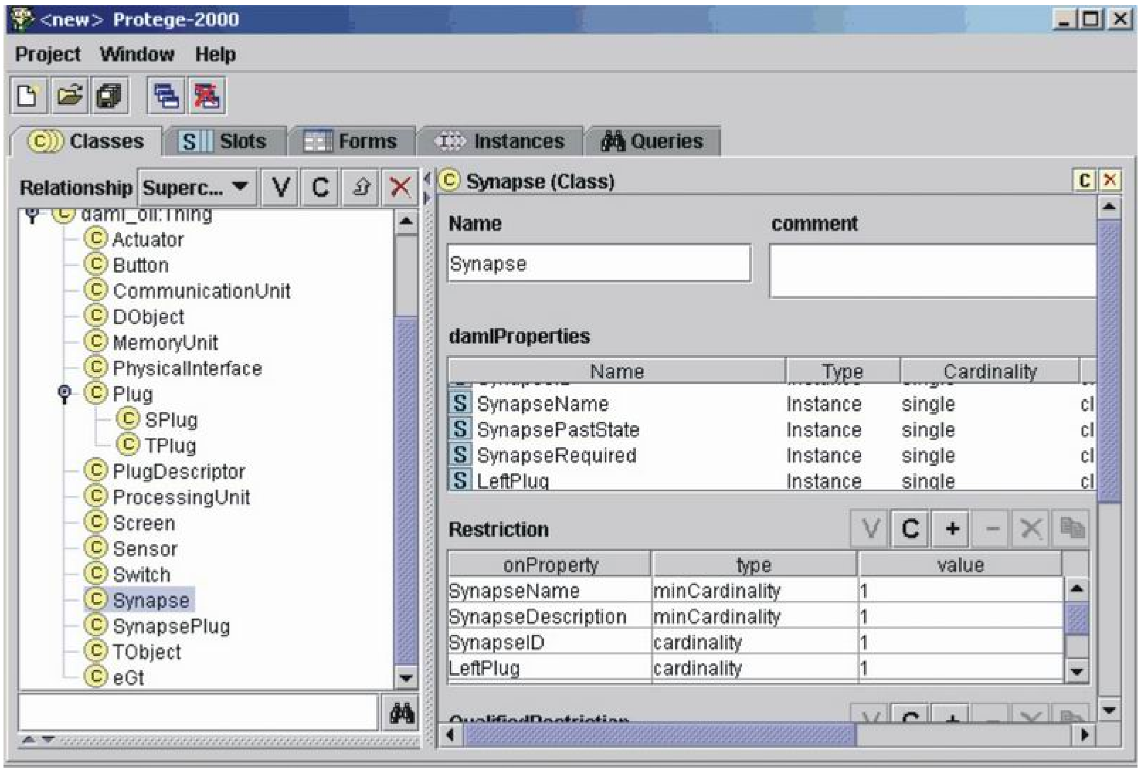
4.2. Tools evaluation

In order to find the most suitable tool for the development of GAS Ontology, some of the criteria mentioned in Gómez-Pérez (2002) and Denny (2002) were used during the tool evaluation process. The selected criteria aim to expose the capabilities of ontology development tools and can be divided into the following categories: criteria that describe the architecture of a development tool, criteria that expose the usability of a tool as well as its interoperability with other ontology development tools and languages and finally criteria that evaluate the capabilities of knowledge representation and inference of a tool. For example the criteria relevant to the architecture of a tool examine whether it is extendible and what is its storage mechanism. Additionally the criteria that evaluate the usability of a tool examine whether a graphical interface is available and if the collaborative development of an ontology is permitted. The prototype ontology was developed using a number of ontology development tools in order to select the appropriate one.

The tools that were evaluated through this process are the following: Apollo, LinkFactory, OilEd, OntoEdit, Ontolingua, Ontosaurus, Protégé-2000, WebODE and WebOnto. These tools support merely the development of an ontology and do not provide services for merging, integrating, annotating or querying ontologies. We selected to focus on this kind of tools as our goal was to develop GAS Ontology and we were not interested to merging and annotating services.

The ontology development tool that proved to be most suitable was Protégé-2000. Protégé-2000 is an ontology editor and a knowledge-base editor that provides an easy to use graphical and interactive ontology-design environment, which is described in Protégé-2000 tool - User’s guide. An important criterion for this selection was the fact that the Stanford Medical Informatics group (SMI) continuously releases new versions of this tool, which support new services. Furthermore its standalone architecture that does not oblige the user to depend on a web browser, as well as the fact that it is an open source application played a major role to this decision. In addition to its highly usable interface, the two other important features that distinguish Protégé-2000 from the other ontology development tools are its scalability and its extensibility. Its component-based architecture, described in Musen et al. (2000), which enables system builders to add new functionality by creating appropriate plug-ins, was a key factor for this selection. The Protégé’s plug-ins are divided into the following three categories: backends, which enable users to store and import knowledge bases in various formats, slot widgets, which are used to display and edit slot values in domain-specific and task-specific ways, and tab plugins, which are knowledge-based applications usually tightly linked with Protégé knowledge bases. The fact that the Protégé-2000 has a backend plug-in that supports storing and importing ontologies in DAML+OIL, as described in Noy et al. (2001), was a trump for the selection of Protégé-2000. Also, the existence of tab plug-ins, which provide capabilities for advanced visualisation, is a very important feature of this tool.

Furthermore the abilities of Protégé-2000 regarding the knowledge representation and the inference mechanism were determinative factors that lead to the selection of Protégé-2000. The Protégé-2000 can support the description of both the basic concepts and the taxonomies that are described in the GAS Ontology. Also, this tool permits the specification of pre-defined and arbitrary constraints for concepts' properties, which include cardinality restrictions, default values and inverse properties. Another important factor is that Protégé-2000 allows the definition of axioms, using the PAL axiom language. Therefore the Protégé-2000 provides a built-in inference engine through PAL, supporting constraint and consistency checking for an ontology. Finally the Protégé-2000 can interoperate with tools like Jess and FaCT, providing different inference mechanisms. A snapshot of the GAS Ontology developed with Protégé-2000 is shown in Figure 4.



Name	comment
Synapse	

damlProperties

Name	Type	Cardinality	
SynapseName	Instance	single	cl
SynapsePastState	Instance	single	cl
SynapseRequired	Instance	single	cl
LeftPlug	Instance	single	cl

Restriction

onProperty	type	value
SynapseName	minCardinality	1
SynapseDescription	minCardinality	1
SynapseID	cardinality	1
LeftPlug	cardinality	1

Figure 4. GAS Ontology developed with Protégé-2000

4.3. Lifecycle of the GAS Ontology

In this section we present how the GAS Ontology is evolved and who participates in this evolution. A graphical representation of the lifecycle of the GAS Ontology is presented in Figure 5. As we have already mentioned the GAS Ontology is divided into two layers: the GAS-CO and the GAS-HO. As the GAS-CO provides the eGts with the necessary common language for their communication, it must be the same for all eGts. Thus it is a static ontology and it is not affected during the deployment of eGWs. We designed and developed the GAS-CO in order to meet the requirements of ubiquitous computing environments. We selected to represent the GAS-CO with the DAML+OIL language and we used the Protégé-2000 in order to develop it. New versions of the GAS-CO that are developed from the eGadgets project team can be used by the eGts, but it cannot be changed either by the eGts' manufacturers or by the eGts themselves.

The development of the GAS-CO using the Protégé-2000 was fairly easy especially because of the Protégé's easy to use graphical interface. The facts that an existing ontology is easily imported from Protégé-2000 and that the addition of new concepts and slots into an existing ontology is very simple supported the step-by-step creation of the GAS-CO as well as the development of various versions. Particularly assistant was the use of a number of visualization plug-ins, such as the OntoViz tab widget and the TGViz tab widget that provided to us a graphical representation of the GAS-CO. Another feature of the Protégé-2000 that facilitated the development of the GAS-CO is its standalone architecture as we did not depend on a web browser. Additionally the DAML+OIL backend plug-in that we used in order to store and import the GAS-CO ontology was very helpful as we have selected the DAML+OIL as the language representation for the GAS Ontology. Furthermore the fact that the Protégé-2000 permits the specification of pre-defined and arbitrary constraints for concepts' properties, like cardinality restrictions, default values and inverse properties, helped us to describe the constraints of the GAS-CO.

The GAS-HO-static represents the description of an eGt containing information about eGt's plugs, the services that are provided through these plugs, its sensors and actuators, as well as its physical characteristics. The developer and / or manufacturer of an eGt know this kind of information for the eGt. Thus they are responsible for the development of the eGt's GAS-HO-static. The knowledge into the GAS-HO-static is represented as a number of instances of the concepts defined into GAS-CO. So the developer of an eGt can use any ontology development tool that imports GAS-CO, in order to develop the GAS-HO-static for an eGt. Changes to the GAS-HO-static are permitted when the developer of an eGt make a change to it. Note that the Protégé-2000 and the JBuilder constitutes the development environment of eGts.

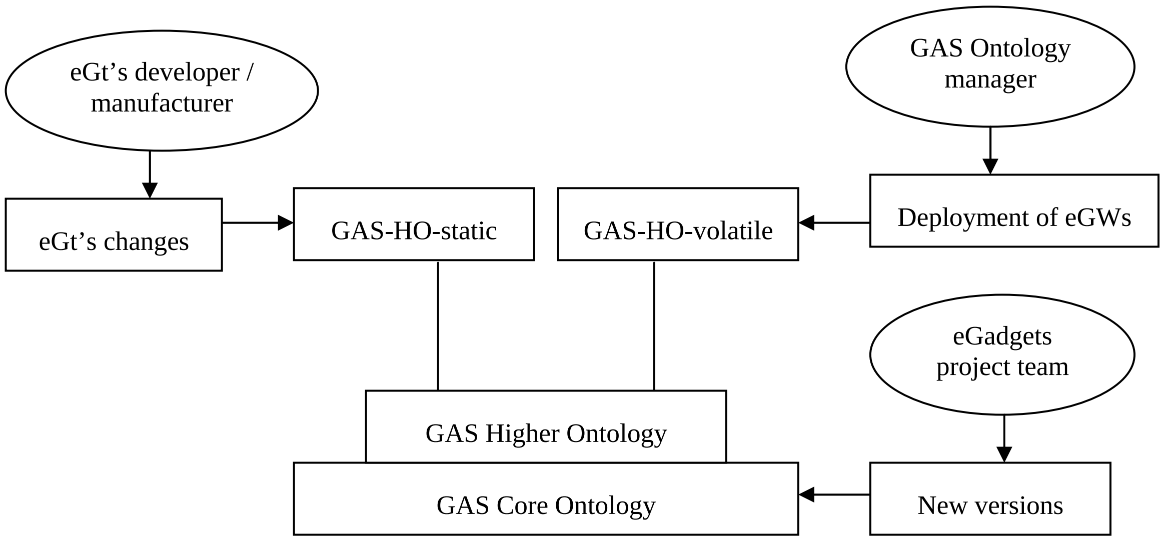


Figure 5. The lifecycle of the GAS Ontology

The GAS-HO-volatile contains information derived from the eGt's acquired knowledge and its use. Specifically it describes the synapses which the eGt's plugs are connected to, the eGWs which it takes part to, as well as information about the capabilities of other eGts that has acquainted through communication. The knowledge represented by the GAS-HO-volatile is dynamic and each eGt is responsible for its GAS-HO-volatile. The changes into the GAS-HO-volatile are made from the GAS Ontology Manager, an eGt's mechanism for the management of GAS Ontology. Note that the changes made by the GAS Ontology Manager are based on the GAS-CO. An eGt's GAS-HO-volatile is updated during various activities of the eGt, such as the exchange of knowledge among eGts, the creation of synapses and the participation of the eGt into an eGW.

5. GAS Ontology manager

The GAS-Operating System (GAS-OS) is the minimum set of modules and functionalities that every device must have, in order to be an eGt and participate in eGWs. The GAS-OS kernel encompasses a communication module, responsible for communication among eGts, a process manager that coordinates all the modules of GAS-OS, a state variable manager, which reflects the state of an eGt's hardware like sensors and actuators, a memory manager, responsible for handling the memory resources of the eGt, and the GAS Ontology manager. The GAS Ontology manager is the mechanism that supports the interaction of the eGt with its stored ontology. The Figure 6 demonstrates the interaction among the GAS Ontology manager and the GAS-OS Kernel.

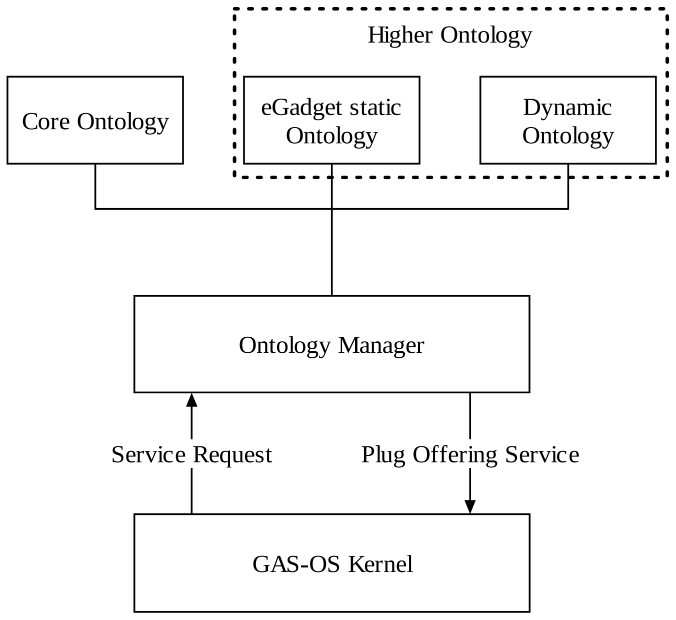


Figure 6. The GAS Ontology manager

We selected to develop the GAS Ontology manager and not to use an existing tool for ontology management because of the following two reasons. The first reason is that the mechanism for the ontology management must have limited requirements, since the eGts have limited processing capabilities. Thus the use of an existing tool, like the Protégé-2000, is not feasible, since these tools cannot run on our platform. The second reason depends on the fact that the GAS-OS needs specific knowledge from the GAS Ontology, so the development of the GAS Ontology manager is more efficient than the use of an existing tool for ontology management. It is in our plans to develop an interface between the GAS Ontology manager and the Protégé-2000, in order to provide to the most capable eGts the abilities of Protégé-2000. For the development of this interface we will use the Protégé API. Some benefits of using the Protégé-2000 for the management of the GAS Ontology are the constraint and consistency checking for the GAS Ontology and the use of the Protégé-2000's query mechanism.

5.1. Role of the GAS Ontology manager

The basic functionality of the GAS Ontology manager is that it provides the necessary mechanism for the interaction of the eGt with its stored ontology. So it can manipulate GAS Ontology, by reading and querying both GAS-CO and GAS-HO and by updating only GAS-HO. Being a module of GAS-OS, the GAS Ontology Manager can provide to the other modules of GAS-OS any knowledge that they need from GAS Ontology. So it supports all the functionalities that demand knowledge represented by the GAS Ontology. Particularly, when two eGts communicate and the exchange of knowledge is necessary, the GAS Ontology Manager enables it by sending parts of an eGt’s GAS-HO to another’s. Also the GAS Ontology Manager provides to the GAS-OS modules information about the known and available services of an eGt assisting the service discovery mechanism.

5.2. Design of the GAS Ontology manager

In this section we present some important features of the GAS Ontology manager, as well as the decisions that we took for its design.

5.2.1. A level of abstraction between the GAS-OS and the GAS Ontology

One of the most important features of the GAS Ontology manager is that it adds a level of abstraction between GAS-OS and the GAS Ontology. This means that only the GAS Ontology manager needs to understand and manipulate the GAS Ontology; the GAS-OS can simply query the GAS Ontology manager for information stored into the GAS Ontology and it does not need to have knowledge about the ontology language or its structure. Therefore any changes that may be done to the GAS Ontology affect only the GAS Ontology Manager and the rest of the GAS-OS is isolated from them.

5.2.2. The GAS Ontology manager and eGt’s ontologies

The GAS Ontology manager is allowed only to query the GAS-CO of an eGt, since it must be common for all the eGts and it cannot be changed during the deployment of eGWs. So the GAS Ontology manager provides methods that query the GAS-CO for knowledge such as the definitions of specific concepts like eGadget and Plug and knowledge relevant to the service classification.

On the other hand the GAS Ontology Manager can both read and write to the GAS-HO of an eGt. Specifically the GAS Ontology manager can only query the GAS-HO-static of an eGt, but it is responsible for keeping up to date the GAS-HO-volatile of an eGt. Note that as into the GAS-HO appear only instances of the concepts defined into GAS-CO, the basic methods of GAS Ontology manager relevant to the GAS-HO can query for an instance and add new ones. The new instances of a concept that the GAS Ontology manager adds to an eGt’s GAS-HO are based on the concept’s definition represented in the GAS-CO. So an important feature of the GAS Ontology manager is that it enforces the integrity of the instances stored in the GAS-HO with respect to the definitions and structures described in the GAS-CO.

5.2.3. Communication among eGadgets

The GAS Ontology facilitates the communication among eGts by providing them with a common language. The communication among eGts is initially established using eGts' GAS-HO; if the differences in these ontologies obscure the communication, then the GAS Ontology manager is responsible for the interpretation of GAS-HO based on the common GAS-CO. Therefore the communication among eGts is ensured. Apart from assisting the communication among eGts, the GAS Ontology manager enables knowledge exchange among them, sending parts of an eGt's GAS-HO to another's.

5.2.4. Service Discovery

One of the GAS Ontology goals is to describe the services that the eGts provide and assist the service discovery mechanism. In order to support this functionality the GAS Ontology manager provides methods that query both the GAS-HO-static and the GAS-HO-volatile for the Services that an SPlug provides as well as for the SPlug that provide a specific Service. So the GAS Ontology manager provides to the GAS-OS the necessary knowledge stored in an eGt's ontology relevant to the eGt's services, in order to support the service discovery mechanism. Similarly the GAS Ontology manager can answer queries for plugs compatibility and eGts replaceability.

5.3. Implementation of the GAS Ontology manager

The GAS Ontology manager was developed using the Java programming language. The requirement that the GAS Ontology manager should run on the Java personal edition runtime environment was given by the fact that GAS-OS runs on resource constraint devices. Keeping in mind that eGadgets is a research project and the software components developed may change over time, in order to use different technologies, the GAS Ontology manager was defined as an interface that is offered to all other GAS-OS components and thus it is easily replaceable.

6. Using the GAS Ontology in a ubiquitous computing environment

In this section we present an example of the use of the GAS Ontology in a ubiquitous computing environment and the role of the GAS Ontology manager. This example is based on a simple scenario for service discovery at the study eGW. According to this scenario a user creates its own study eGW using two eGts, an eBook and an eLamp. Both these eGts share the same GAS-CO, but they have different GAS-HO-static. For example, the eLamp's GAS-HO-static contains information about eLamp's SPlug "switch on/off" and the eBook's GAS-HO-static contains the description of SPlug "open/close". These two eGts are connected through a synapse which is established between the aforementioned plugs forming the study eGW. So when the user opens the eBook, the eLamp switches on, adjusting the light conditions to a specified luminosity level in order to satisfy the user's profile. The knowledge emerged from this synapse is stored in the GAS-HO-volatile of both the eGts that participate to it. So the eBook "knows" that it's SPlug "open/close" participates to a synapse with an SPlug that provides the service "light" with a specific luminosity.

If this synapse is broken, for example because of a failure at the eLamp, a new eGt having an SPlug that provides the service "light" must be found. The eBook's GAS-OS needs to find another eGt with an SPlug that provides the service "light". The eBook's GAS-OS is responsible to send a message for service discovery to the other eGts' GAS-OS that participate to the same eGW. This type of message is predefined and contains the type of the requested service and the service's attributes.

When the GAS-OS of an eGt receives a service discovery message, forwards it to the eGt's GAS Ontology manager. Assume that the eGt "eDeskLamp" participates to the study eGW and that this is the first eGt that gets the message for service discovery. The eDeskLamp's GAS Ontology Manager first queries GAS-HO-static of eDeskLamp in order to find if this eGt has an SPlug that provides the service "light". If we assume that the eDeskLamp has the SPlug "LampDimmer" that provides the service light, the GAS Ontology manager will send to the eDeskLamp's GAS-OS a message with the description of this SPlug. If such an SPlug is not provided by the eDeskLamp, the GAS Ontology Manager queries the eDeskLamp's GAS-HO-volatile in order to find if another eGt, with which the eDeskLamp has previously collaborated, provides such an SPlug. If the GAS Ontology Manager finds into GAS-HO-volatile such an SPlug it sends to the eGt's GAS-OS the description of this SPlug. If the queried eGt, in our example the eDeskLamp, has no information about an SPlug that provides the requested service, the control is sent back to GAS-OS, which is responsible to send the query message for the service discovery to another eGadget. Note that all the eGadgets have the same service classification, which is stored into the GAS-CO; thus the messages for service discovery are based on this classification.

7. Related work

Since the eGts, the artefacts that constitute a ubiquitous computing environment, can be perceived as agents that communicate and collaborate, our work is closely related to the field of agent communities. It is widely acknowledged that without some shared or common knowledge the members of a multi-agent system have little hope of effective communication. According to Huhns and Singh (1997), ontologies can support the inter-agent communication by providing a definition of the agents' world as well as by providing terms that can be used in communication. In the agent communities the most common way to enable software agents to communicate, is to give each of them the same ontology. In our work we preserve this solution, as we demand all eGts to have the same GAS Core Ontology.

The issue that the agents of a multi-agent system may have different ontologies is not rare in the agent field. A number of approaches have been proposed in order to handle this issue. A generic one was made by Steels (1998), who proposed to let shared ontologies to evolve within the multi-agent system. Another approach used by Stephens and Huhns (2001) proposes to merge the existing ontologies in order to result in a common one that includes all the aspects of the individual ontologies. The most used approach, described by Hendler (2001), depends on the definition of mapping between concepts of different ontologies. The solution that we propose is based on the idea that all eGts have a common ontology, the GAS-CO, and each eGt have also a different, "private" ontology, the GAS-HO that is based on the GAS-CO. This idea is similar to the one presented by Stuckenschmidt and Timm (2002), where the communication among the agents relies on partially shared ontologies. Another important feature of our work is that we enable the exchange of knowledge among eGadget by permitting the exchange of parts of their ontologies. The result of this procedure is that the eGts can incorporate new knowledge to their ontologies and that they can learn through their communication. A relevant idea is presented by Soh (2003).

As we have already mentioned the ontologies can handle a number of issues emerged from the composition of ubiquitous computing environments. The goal of the GAS Ontology, which was presented in this paper, is to address issues such as the semantic interoperability among heterogeneous eGts and the semantic service discovery. Ontologies have been also used in a number of ubiquitous computing infrastructures in order to address the same issues.

A known use case is presented in Maffioletti and Hirsbrunner (2002), where UbiDev, a homogeneous middleware that allows definition and coordination of services in interactive environment scenarios, is described. In this middleware, according to Schubiger et al. (2000), resource classification relies on a set of abstract concepts collected in an ontology and the meaning of these concepts is implicitly given by classifiers. The main advantage of this approach in facing resource management problem is that resources selection is based on their semantics that is given by the context. Since every application may have its ontology the application structure results separated from the implementation. This approach is different than the one that we have followed, because whereas Maffioletti and Hirsbrunner (2002) use an ontology for each application that includes several devices, our goal is to provide an ontology that facilitates the use of devices in various ad hoc eGWs.

Ontologies are also integrated in the Smart Spaces framework GAIA presented by McGrath et al. (2003) and Ranganathan et al. (2003b). In this work the ontologies have been used in order to overcome a number of problems in the GAIA Ubiquitous computing framework, such as the interoperability between different entities, the discovery and matching and the context-awareness. The approach that the GAIA framework follows is fairly different to the one that we have proposed for the eGadgets project. Specifically in the GAIA framework there is an Ontology Server that maintains the ontologies and there are different kinds of ontologies, such as ontologies that have meta-data about the environment's entities and ontologies that describe the environment's contextual information. The ontologies in GAIA are also used in order to support the deployment of context-aware ubiquitous environments, more on Ranganathan and Campbell (2003a). Chen et al. (2004) have also described the COBRA-ONT, an ontology for context-aware pervasive computing environments.

Finally a very interesting work is the one made by the Semantic Web in UbiComp Special Interest Group (SW-UbiComp-SIG). The basic goal of this group is to define an ontology to support knowledge representation and communication interoperability in building pervasive computing applications. This project's goal is not aimed to construct a comprehensive ontology library that would provide vocabularies for all possible pervasive applications, but to construct a set of generic ontologies that allow developers to define vocabularies for their individual applications. The Standard Ontology for Ubiquitous and Pervasive Applications includes the following domain ontologies: the spatial ontology, the temporal ontology, the person profile/user model, the event ontology, the device profile, the digital document and the security and privacy policy.

8. Conclusions and Future work

In this paper we have presented research and development work that have been conducted in the context of the eGadgets project, see eGadgets project website, under the proactive initiative "Disappearing Computer" of the EU IST/FET research program. Our approach attempts to resolve the complexity of the ambient applications by introducing an architectural framework that supports the composition of ubiquitous computing systems by people who can use artifacts as building blocks, the Gadgetware Architectural Style (GAS). A part of this work was the development of GAS Ontology, in order to facilitate a common understanding among these artifacts. Furthermore the GAS Ontology manager was implemented as a component of GAS-OS, in order to manipulate GAS Ontology.

So far GAS-OS exploits a number of functionalities that the GAS Ontology offers like service discovery and eGts replaceability. The current version of the GAS Ontology assumes that all eGts know the same service classification. This is a limitation that we intent to eliminate by adding to GAS Ontology manager the capability to map a service description to another one, using the knowledge that eGts have acquired from their collaboration. Another goal is to create a graphical interface for the GAS Ontology manager through which the eGadget users can add into GAS Ontology information about the context of their environment, as well as semantic information about the synapses that they create and their functionality. Our endmost goal is to move from the current version of GAS-OS that is based on hard-coded descriptions of the basic terms of GAS, to a version that the GAS Ontology will provide an abstraction even for the basic terms.

Our future work has also another different direction, which is related to the Protégé-2000. One of our imminent plans is to develop an interface between the GAS Ontology manager and the Protégé-2000, by using the Protégé API. We believe that the use of Protégé-2000 will facilitate the query mechanism and support an inference mechanism. We also intend to develop a plug-in for the Protégé-2000 that will support the generation of ontologies compliant to the terms of ubiquitous computing environments from Protégé projects.

Finally, this work will carry on in the EU IST/FET funded project PLANTS, see PLANTS project website. In this project plants are enhanced with sensing, actuating and communication abilities, in order to be considered as a different form of artifacts that they have plugs through which they are able to communicate with other artifacts in digital space and form mixed interacting communities of eGts and ePlants called bioGadgetWorlds.

References

Borst, W.N., 1997. Construction of Engineering Ontologies. University of Twente, Enschede, NL-Centre for Telematica and Information Technology.

Chen, H., Finin, T., Joshi, A., 2004. An ontology for context aware pervasive computing environments, To appear, Knowledge Engineering Review - Special Issue on Ontologies for Distributed Systems, Cambridge University Press.

Christopoulou, E., 2003. An evaluation framework for ontology languages and tools. Master thesis. Department of Computer Engineering and Informatics, University of Patras, Greece.

Christopoulou, E., Kameas, A., 2003. GAS Ontology: Conceptualising Gadgetware Architectural Style. In the Proceeding of Tales of the Disappearing Computer, Santorini, Greece.

Corcho, O., Gómez-Pérez, A., 2000. A roadmap to ontology specification languages. 12th Int'l Conference Knowledge Engineering and Knowledge Management (EKAW00), Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 80–96. Berlin. http://delicias.dia.fi.upm.es/articulos/ocorcho/ekaw2000-corcho.pdf

Denny, M., 2002. Ontology building: A survey of editing tools. XML.com http://xml.coverpages.org/Denny-OntologyEditorSurveyText20021111.html

Disappearing Computer initiative http://www.disappearing-computer.net/

eGadgets project website http://www.extrovert-gadgets.net

Gil, Y., Ratnakar, V., 2002. A Comparison of (Semantic) Markup Languages. In the Proceedings of the 15th International FLAIRS Conference, Pensacola Beach, Florida, USA.

Gómez-Pérez, A., 2002. Deliverable 1.3: A survey on ontology tools. OntoWeb http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D13_v1-0.zip

Gómez-Pérez, A., Corcho, O., 2002. Ontology languages for the semantic web. IEEE Intelligent Systems, Vol. 17, No1, pp 54-60.

Van Harmelen, F., Patel-Schneider, P., Horrocks, I., 2001. Reference description of the DAML+OIL (March 2001) ontology markup language. The DARPA Agent Markup Language Program. http://www.daml.org/2001/03/reference

Hendler, J., 2001. Agents and the semantic web. IEEE Intelligent Systems, pages 30–37, March/April

Huhns, M., Singh, M., 1997. Ontologies for agents. IEEE Internet Computing, pages 81–83, November/October

Kameas, A., Bellis, S., Mavrommati, I., Delaney, K., Colley, M., Pounds-Cornish, A., 2003. An Architecture that Treats Everyday Objects as Communicating Tangible Components. In Proceedings of the 1st IEEE International conference on Pervasive Computing and Communications (PerCom03). Forth Worth, USA.

Maffioletti, S., Hirsbrunner, B., 2002. UbiDev: A Homogeneous Environment for Ubiquitous Interactive Devices. Short paper in Pervasive 2002 - International Conference on Pervasive Computing. pp. 28-40. Zurich, Switzerland.

McGrath, R., Ranganathan, A., Campbell, R., Mickunas, D., 2003. Use of Ontologies in Pervasive Computing Environments. Report number: UIUCDCS-R-2003-2332 UILU-ENG-2003-1719 Department of Computer Science, University of Illinois

Musen, M., Fergerson, R., Grosso, W., Noy, N., Crubezy, M., Gennari, J., 2000. Component-Based Support for Building Knowledge-Acquisition Systems. Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000). Beijing.

Noy, N., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., Musen, M., 2001. Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems, 16(2), pp 60-71.

PLANTS project website http://www.edenproject/PLANTS

Protégé-2000 tool - User's Guide. http://protege.stanford.edu/publications/UserGuide.pdf

Ranganathan, A., Campbell, R., 2003. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil.

Ranganathan, A., McGrath, R., Campbell, R., Mickunas, D., 2003. Ontologies in a Pervasive Computing Environment. Workshop on Ontologies in Distributed Systems at IJCAI, Acapulco, Mexico.

Ribière, M., Charlton, P., 2000. Ontology overview, Motorola Labs. http://www.fipa.org/docs/input/f-in-00045/f-in-00045.pdf

Schubiger, S., Maffioletti, S., Tafat-Bouzid, A., Hirsbrunner, B., 2000. Providing Service in a Changing Ubiquitous Computing Environment. Proceedings of the Workshop on Infrastructure for Smart Devices - How to Make Ubiquity an Actuality, HUC.

Semantic Web in UbiComp Special Interest Group. http://pervasive.semanticweb.org

Soh, L.-K., 2003. Collaborative Understanding of Distributed Ontologies in a Multiagent Framework: Design and Experiments. In the Proceedings of the Third International Workshop on Ontologies in Agent Systems (OAS). Melbourne, Australia.

Steels, L., 1998. The origins of ontologies and communication conventions in multi-agent systems. Autonomous Agents and Multi-Agent Systems, 1(1):169–194.

Stephens, L., Huhns, M., 2001. Consensus ontologies - reconciling the semantics of web pages and agents. IEEE Internet Computing, pages 92–95, September/October.

Stuckenschmidt, H., Timm, I. J., 2002. Adapting Communication Vocabularies using Shared Ontologies. In the Proceedings of the Second International Workshop on Ontologies in Agent Systems (OAS). Bologna, Italy.

Uschold, M., Gruninger, M., 1996. Ontologies: principles, methods and applications. Knowledge Engineering Review, Vol. 11 No. 2, pp. 93-155.