Ambient Intelligence Applications: Introducing the Campus Framework

Amal El Fallah Seghrouchni[1], Karin Breitman[2], Nicolas Sabouret[1], Markus Endler[2], Yasmine Charif[1], Jean-Pierre Briot[1,2]

[1]UPMC-LIP6, [2] Pontifícia Universidade Católica do Rio de Janeiro
{Amal.Elfallah, Nicolas.Sabouret, jean-pierre.briot, Yasmine.Charif }@lip6.fr,
{karin,endler}@inf.puc-rio.br

Abstract

A challenge for pervasive computing is the seamless integration of computer support with users' activities in a very dynamic setting, with deep human and resource mobility. Portable devices such as notebooks, PDAs and smartphones are becoming more and more popular, as their computational power increases and their prices fall. Moreover, the quick spread of wireless networks has permitted the exchange of data in places as diverse as university campi, airports, coffee houses and family homes. In this paper we introduce CAMPUS, a framework that supports the development of multi- agent, context aware, pervasive computing applications. Campus is designed to provide the necessary infrastructure for ambience intelligence applications.

1. Introduction

Over the last few years most business processes changed on various dimensions (e.g. flexibility, interconnectivity, coordination style, autonomy) due to market conditions, organizational models, and usage scenarios of information systems. Frequently, information is relocated within geographically distributed systems, as a result of the application of unclear rules or private businessbussiness process. This fact creates a need for a software infrastructure that enables trustworthy and secure pervasive/ubiquitous mobile and collaboration systems.

The *anywhere/any time* paradigm is becoming the new challenge to the conception, design, and release of the next generation of information systems. New technologies, like Wi-fi networks and 3rd generation mobile phones, are offering the infrastructure to conceive information systems as ubiquitous, that is, systems that are accessible from anywhere, at any time, and with (almost) any electronic device. Ubiquity is not yet another buzzword pushed by emerging technologies, but rather a means of supporting new business models and encouraging new ways of working. Ubiquitous and mobile collaboration systems require new conceptualizations, models, methodologies, and support technologies to fully explore its potential.

Increasing numbers of people need to move across organizational boundaries to collaborate with others within their organization as well as among organizations and different workplaces. The ability to query the company's distributed knowledge base and to cooperate with co-workers is still a requirement, but mobility introduces new accessability scenarios and increases complexity. New issues, such as how to enable users to retain their ability to cooperate while located in different workplaces, the role of context and location in determining cooperation, the support for ad-hoc cooperation in situations where the fixed network infrastructure is absent or cannot be used are beginning to arise.

The approaches and technologies for supporting these new ways of working are still under investigation. Nevertheless, they are likely to "borrow" concepts and technologies from a variety of fields, such as work-flow systems, human computer interaction, semantic web, groupware and CSCW, event-based systems, ontology engineering, software architecture, distributed database systems, mobile computing, ubiquitous information systems, and others. A particularly interesting line of research is exploring the ambient intelligence paradigm, a multi disciplinary approach that aims at the integration of innovative technologies that empowers users through a service on demand environment. The rest of the this paper is structured as follows: in section 2 we discuss ambient intelligence, in section 3 multi agent systems and their relevance to ambient intelligence, in section 4 we introduce the architecture of the Campus Framework, in section 5 we exemplify the architecture, and in section 6 we present our final remarks.

2. Ambient Intelligence

Ambient Intelligence (AmI), i.e. « intelligent » pervasive computing, builds on three recent key technologies [1]: Ubiquitous Computing, Ubiquitous Communication and Intelligent User Interfaces. Ubiquitous Computing is the integration of microprocessors into everyday objects like furniture, clothing, white goods, toys, even paint. Ubiquitous Communication enables these objects to communicate with each other and the user by means of ad-hoc wireless networking. An Intelligent User Interface enables the inhabitants of an AmI environment to control and interact with the environment in a natural (voice, gesture) and personalized way (preferences, context).

Ambient intelligence aims at making use of those entities in order to provide users with an environment which offers services when and if needed. One great challenge of such environments is how to adequately address the heterogeneity and dynamic nature of users, services and devices. A key issue is how to identify and activate the appropriate service within a continuously changing multitude of services [2].

The integration of these competences produced several results in the composition of ambient services. Most are centered in the orchestration between the user and other entities in the process of achieving one or more user goals. Gárate et al [3] developed a network of domestic appliances, e.g, refrigerators, television, blenders, that were interconnected and managed by a central control. In this network the user could open a natural language dialog with the appliances and pose his or her needs. Phillips Research HomeLab [4] proposed the intelligent bathroom concept. It comprises an interactive mirror, that plays a central role in the coordination of the interaction with the other devices in the room, e.g., evaluating the results of the scale and recommending a health program. In both the services were defined statically, i.e., based on a given scenario.

Vallée et al. propose the use of semantic web techniques to aid the composition of services offered by ambient objects [5]. In this approach, an abstract plan is provided for each task that can be requested by the user. An example is the kitchen surveillance task, that is associated to a plan to be accomplished by a set of subtasks, such as verifying whether the oven was switched off and the taps closed, before exiting the kitchen environment Their composition mechanism connects every sub-task to some service provided, while taking care that context parameters are respected and general coherence of the performed sub-task is achieved. The decomposition of the user needs is done *a priori*, to be distributed among the set of discovered services [6]. To the best of our knowledge, most approaches in literature proposed centralized control for ambient services, reducing the problem to the orchestration of sub tasks. To realize the vision presented in section 1 we must be able to organize tasks in a decentralized way, thus achieving coordination rather than orchestration of services. In the next section we describe how we intend to use multi- agent systems to this purpose.

3. Multi Agent Systems

To shift from centralized service control (orchestration) to decentralized (choreography), we propose to adopt the paradigm of Multi-Agent Systems (MAS) as an architectural basis for the proposed Campus framework. Multi-agent systems (MAS) materialize both human users as well as artificial agents (assistants, problem solvers…), and their various types of interaction and collaboration in a single software system. The MAS paradigm is suitable to deal with AmI and ubiquitous intelligence (UI) for several reasons:

At the cognitive level, Multi-Agent Systems (MAS) are composed of intelligent agents and a real or simulated environment. MAS offer a well established paradigm to implement computational intelligence as a collective body of knowledge, to be distributed among intelligent agents. Hence, intelligent agents constitute concrete implementations of embedded digital intelligence, that could be in everyday objects, workplaces, homes, clothes, etc.

Interaction between agents throughout the environment has the potential to provide trustworthy and relevant services to users. In addition, these agents can also interact efficiently with users and other agents thanks to sophisticated features of ACLs (agent communication languages) and interaction protocols.
Depending on the system to be designed, the agents may be endowed with different levels of intelligence, e.g., context-aware, active, proactive, interactive, reactive, adaptive, and cognitive. Moreover, the notions of agents and organizations and their decentralized and pro-active nature match well the requirements of large-scale pervasive computing environments. In the next section we introduce Campus, a framework to support the development of multi- agent, context aware, pervasive computing applications.

3.1 Definitions

Before presenting the proposed framework, we define a few key concepts:

Context is any information which characterizes the state of a resource (of a device), of a user or of an environment. In the scope of the Campus framework, we handle the following types of context information:

- System context: data about the mobile device's and the network resources, ,including device capabilitie, currently free memory, CPU utilization, battery level, connectivity status, connectivity parameters (such as IP address, mask, current wireless (Wifi) access point, etc.
- Physical context: data about a device's or an environment's symbolic location (as opposed to its geographic position), city, country, and data collected from sensors, such as temperature, noise, luminosity, acceleration, etc.
- Time context: information such as hour of the day, day of the week, week of the year, month, year, etc.
- User context: data that indicate the user's role, profile, preferences, activity, etc. Compared to the other types of context information, this is the only kind of information which usually cannot be probed or inferred automatically. Hence, in the Campus framework, agents may try to infer their user context, but the user will always be requested to confirm or complement/correct her user context values.

A service is an operation that can be invoked remotely by any software component or by a user through a graphical user interface. In the Campus framework, we will distinguish two kinds of services : environment services that are deployed on servers using WSDL [7] descriptions and that can be invoked through the SOAP protocol; and agent services which are offered by agents and that can be invoked using ACL messages.

An agent is a software process with the following properties:

- It is situated, which means that it can communicate with an environment to invoke its services.
- It is interactive, which means that it can communicate with other agents (either directly or through the environment), to invoke their services.
- It is autonomous, which means that it makes use of local decision process to select its operations at runtime. One key implication of autonomy is that an agent can refuse to perform a service when requested, even if it has the capacity to perform it.

4. Campus Architecture

The Campus architecture is intended as a configurable framework in which users can decide what services they want to enable in their environments, rather then a monolithic application. It is composed of three levels: the context-provisioning layer, the communication and coordination layer, and the ambient services' layer, as illustrated by Figure 1. In a nutshell, the bottom level, implemented using the MoCA service-oriented middleware is responsible for offering basic middleware services and functionality, such as providing context and positioning information and device discovery. Overall communication and coordination is provided by the middle layer that integrates the Campus ontological model to mediation services combined to the coordination protocol proposed by Charif et al. [8]. This protocol provides a communication and orchestration layer with software agents in the environment. Finally, the topmost layer provides application specific and ambient services and acts as a hotspot, i.e., allows users to extend the framework by plugging in specific services required by a particular user, environment, type of collaboration, of interest to their environment. In this paper we elaborate the bottom and middle layers as follows. This architecture is largely inspired by the NIST/ECMA "toaster model" architecture [9].

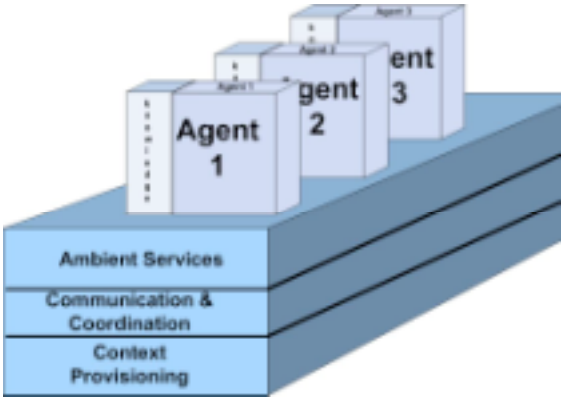

Figure 1 – Campus Architecture

4.1 Context Layer

In the Campus framework, the context layer is implemented using the MoCA middleware. MoCA (*Mobile Collaboration Architecture*) [10] is a service-oriented architecture that supports the development of context-aware applications and services for mobile computing. Besides a small and simple set of APIs to build such systems, MoCA provides efficient services

to collect, store and distribute context information associated with mobile devices. This information comprises not only raw data related to the device's resources and the wireless links (currently, only IEEE 802.11), but also the symbolic location of each device, which is inferred from the RSSI values measured at the device with respect to all the Wi-Fi Access Points in its vicinity.

Among the core services of MoCA is CIS (*Context Information Service*), which is responsible for storing, processing and providing synchronous (i.e. request-reply) and asynchronous (i.e. event-based) access of context data probed to applications (or other services). Examples the context data handled by CIS are the device's CPU utilization, free memory, battery level, current IP address, quality of the wireless connectivity, capabilities of the mobile device (e.g. screen size, built-in camera), among others.

Another of MoCA's core service is the LIS (Location Inference Service) [11] which is responsible for inferring the symbolic location of a mobile device by comparing the current pattern of RSSI (Radio Signal Strength Indicator) with the RSSI pattern previously probed at specific (indoor or outdoor) Reference Points, such as the entrance of a building, a meeting room, or a section of an auditorium. A service called Monitor (currently available for Windows XP/Mobile, Linux and SymbianOS) has to be executed on each mobile device for the purpose of collecting this context data and send it to CIS. Other MoCA core services include a discovery service (UDS), which allows devices to discover and connect to services at each visited symbolic location (including services executing on other nearby mobile devices), and a configuration service (CS), which is used to bootstrap the device's connections to UDS and CIS.
Although MoCA's original APIs are only for Java, recently we have added also multi-platform support through gateways (a.k.a "MoCA Personalities") for JADE, Web Services and Corba's IIOP.

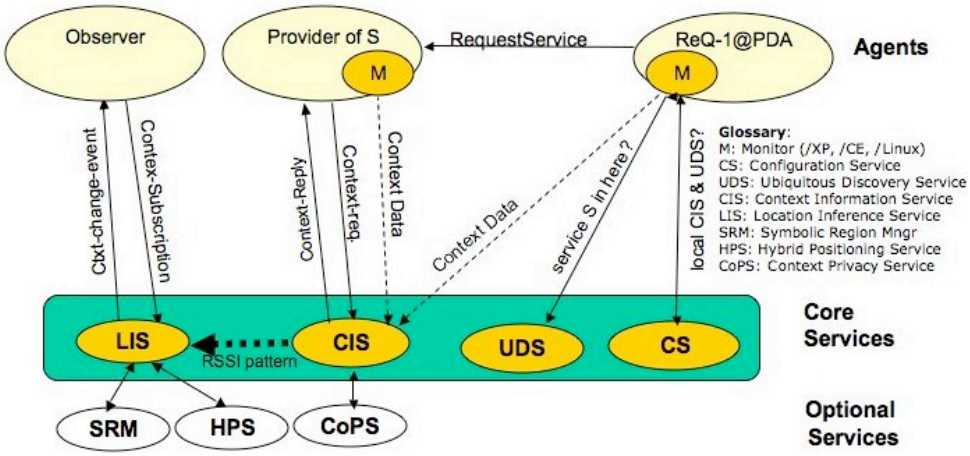

Figure 2 – MoCA Layer Architecture

4.2 Communication and Coordination Layer

This layer is composed of several infrastructure agents, an agent communication and coordination protocol, as well as a base ontology. We illustrate the architecture of this layer in Figure 3, as follows.

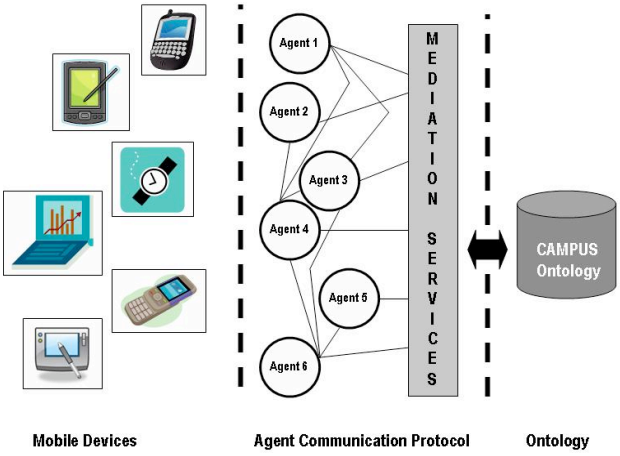

Figure 3 – Communication and Coordination Layer Architecture

4.2.1 Multi-Agent Infrastructure

The multi-agent system relies on an environment (I.e. the MAS infrastructure) which is responsible for providing the following services:

- The Agent Management Service (AMS), which is responsible for loading and unloading agents on the platform;
- The Directory Facilitator Service (DF), aka Yellow Pages, which allows agents do discover each other
- The Communication Layer, which is responsible for ACL message transportation. Since ubiquitous environments are essentially distributed, the communication layer will use brokering services which allow agents to communicate from one server to another.
- The Mediation Service which deals with semantic heterogeneity (see section 4.2.2).

Other coordination mechanisms can be offered by the environment (typically using coordination artifacts). We do not present them in this paper.

Each physical component of the ambient environment (mobile device, home appliance...) is then represented by an agent in the system. The agent publishes in the DF the list of services it offers. The user can communicate with any agent to express its needs. Agents interact with each following the coordination protocol to fulfill the users requirements.

4.2.2 Campus Ontology

The Campus ontology serves as a knowledge base for the framework implementation, i.e., provides the necessary semantics to allow high level exchanges, including brokering, negotiation and coordination amongst software agents. It contains precise definitions for every relevant concept in the framework (for each instance of the framework, that is), e.g, it defines that a service is described by a tuple containing its name, a parameter list, a capability list, the communication port number, and protocol. Of course the concepts of name, parameter list, capabilities list, port and protocol are also defined in the ontology. This ontology serves as a static model of our domain and will be used as a basis upon which the mediation services will try to reason and understand the information provided by agents in the environment. Ontology was chosen over other conceptual models, e.g. glossaries and acronyms; a taxonomy is a set of terms arranged in a generalization-specialization (parent-child) hierarchy because they are much more expressive [12]. A controlled vocabulary is simply lists a set of terms and definitions,. A taxonomy may or may not define attributes of these terms nor does it specify other relationships between terms, e.g. RosettaNet and ebXML; a relational database schema defines a set of terms through classes, attributes and a limited set of relationships among those classes; an OO software model defines a set of concepts and terms through a hierarchy of classes and attributes and a broad set of binary relationships among classes. Constraints and other behavioral may be specified through methods on the classes (or objects). An ontology can express all of the preceding relationships, models and diagrams as well as, n-ary relations, a rich set of constraints, rules relevant to usage or related processes and other differentiators including negation and disjunction [13]. Table I summarizes the benefits of the adoption of ontology.

Table I – Benefits of adopting formal ontology to model ambient knowledge in Campus.

Ontologies are semantically richer (greater expression power than taxonomies, entity relationships or OO models)
Conceptual knowledge is maintained through complex and accurate representations above and beyond hierarchical approaches
Ontologies are formal - OWL DL ontologies map directly to Description Logic (a dialect of first order logics)
Formal ontologies in the OWL DL standard can verified/classified with the aid of Inference Mechanisms, e.g. RACER and FaCT: - consistency checks - classification

- new information discovery
OWL ontologies use a XML/RDF syntax that allows them to be automatically manipulated and understood by most resources on the Internet
Capture and represent finely granulated knowledge
Ontologies can be used to reduce ambiguity so as to provide a model over which information can be freely shared and acted upon by autonomic managers
Ontologies are modular, reusable and code independent - ontology driven applications are specified separately from the ontology itself. Changes to the ontology should not impact the code or vice versa
Ontologies can be combined with emerging rule languages, such as SWRL

4.2.3 Mediation Services

Multi-agent systems are based on the interactions among autonomous entities to provide a desired global behaviour. The coherence of the system as whole depends on the capacity of the agents to interpret messages received from their pairs. In this scenario it is necessary to distinguish between two types of communication:

- Communication level per se – it is the level where we define message management (communication protocols). Multi- agent systems usually make use of standards, such as KQML [14] and the FIPA-ACL [15] that define a series of performatives or message categories. We have adopted the VDL-ACL [16] for that purpose, described in section 4.2.3
- Contention/Interpretation level – it is the level where we define what the agent must interpret. To assure correct interpretation of the messages we must guarantee that the former carry some associated semantics.

We assume that every agent in the environment possesses some associated information. It can vary from very basic information, such as a list of services and capabilities provided, to very sophisticated ontological descriptions of itself, i.e. of the roles and preferences of the user it is representing.. We cannot make any assumption on the format the internal agent knowledge is encoded either. It can vary from an ontology to a database or even a string of characters. Because we cannot guarantee that the agents will provide semantically compatible descriptions with the environment's (encoded in the Campus ontology, see section 4.2.1), we must provide mediation services that aim at aligning, mapping, negociating and reconciling different knowledge representations used by the software agents. By mediation services, we mean the problem of designing a *mediator*, a software service that is able to translate user queries, formulated in terms of a common schema M (the Campus ontology, in our case), into queries that can be handled by the knowledge representation of the agent. The mediator must therefore match each agent knowledge representation with the Campus ontology. The problem of query mediation becomes a challenge in the context of open MAS environments, where the number of different software agents may be enormous and, moreover, the mediator does not have much control over the agent's knowledge representation, which may join or leave the mediated environment at will.
Those services include an ontology mapping service, CATO [12, 17], similarity measurement among concepts [18] and mediators [19, 20, 21]. Figure 3 shows how the Campus ontology, communication protocol and mediation services components interact. Please note that for each device in the environment, e.g. mobile phone, PDApda, a virtual counterpart is created and materialized as a software agent. The software agents communicate among themselves and the framework using Charif et al. coordination protocol [8]. Further interaction can be obtained from the mediation between the framework's knowledge (captured by the Campus ontology) and the knowledge imbedded in each agent individually. This mediation process is assisted by the mediation services component.

4.2.4 ACL and Agent Coordination Protocol

An ACL is designed to enable software (agents) to exchange information, knowledge or services. In our architecture, agents are provided with an ACL, called VDL-ACL [16], allowing them to communicate with software or human users' needs and build messages expressing information about their competences.

Indeed, ACLs traditionally used in agent communication, such as KQML and FIPA-ACL, typically assume that agents of the system are artificial and that their main objective is knowledge exchange.
However, if we consider ambient intelligence case studies, involving mixed communities (human users and devices), then artificial agents require a new conversation language which enriches their communicational abilities so that they may participate in exchanges of ideas, bargaining sessions or planning.

The ACL used in Campus' communication layer structures each message as shown on figure 4. Thus, each message encompasses its sender and receiver agents' IDs, an identifier (reply-with), the identifier of the message to which the concerned one answers (in-reply-to), a conversation identifier (conv-id) assigned to all the messages involved to solve a common goal. Moreover, a message's content can be composed of one or several requests.

```
m = [   sender: AgentID;
        receiver:       AgentID;
        conv-id,        String;
        reply-with:     String
        in-reply-to:    String;
        req-id:         int[];
        content:        {r1, r2, …}   ]
```

Figure 4 – Message Structure in the ACL

Each request r in the ACL is a pair $\langle \alpha, \delta \rangle$ where α is the performative and δ a propositional content. The request's performative α determines its range. A request can then be a question on data, an assertion, an action command, an execution acknowledgement, an answer stating that some parameters are missing for the agent to execute an action, an answer stating that the agent ignore the asked data or is not capable to perform an action, etc.

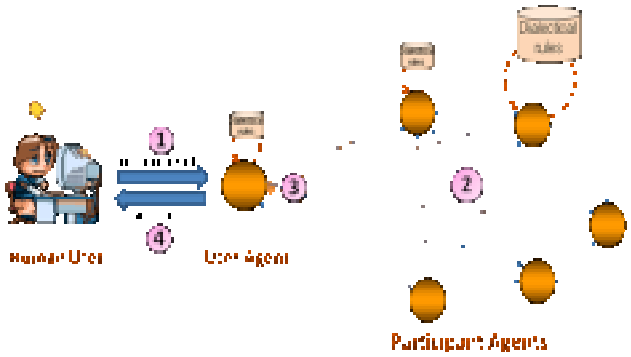

Figure 5 – Overview of the Coordination Protocol

The coordination protocol [8] used by the agents assumes that each agent that has to perform a task (asked by a human user or planned, etc) tries first to do it by its own. In figure 5, which represents an overview of the coordination protocol, this is illustrated as the first step. The requested agent has then the role of the user agent, which is in charge of returning the final answer to the user. If an agent fails to perform (integrally) a task, then it coordinates with its acquaintances to achieve, decompose, and cover it. This is represented by the second step in figure 5, where all the agents involved in the coordination have the role of participant agents. At the end of the coordination, each participant agent is supposed to have built an answer for the user requirements, i.e. whether an assertion, an acknowledgment, an answer stating that it cannot process the request, etc. All the answers are then returned to the user agent. This is illustrated as the third step in figure 5. Finally, in the fourth step, the user agent sorts the received answers, eventually proceeds to a selection regarding QoS parameters (resolution, sound, etc) if any, and returns the final answer to the user.
The coordination protocol assumes that agents are provided with a history table to store all the messages exchanged to reach their goal.

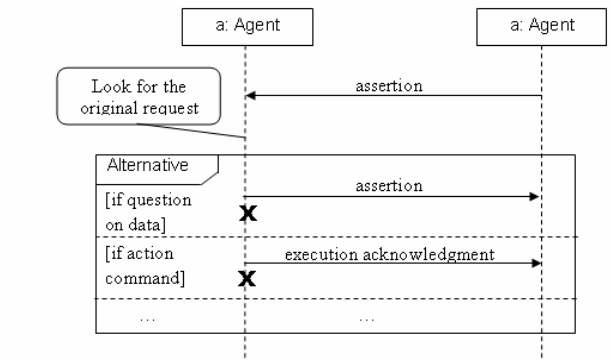

Figure 6 – Assertions Processing by the Coordination Protocol

To achieve agents' coordination, the protocol defines dialectical rules that agents use for messages processing. Each dialectical rule considers the requests contained in a message. Following the requests ranges, the rule specifies the appropriate processing. For example, figure 6 illustrates the AUML diagram of the dialectical rule that processes assertions, i.e. answers to questions. This figure shows that, following this rule, an agent receiving an assertion has first to check in its history table whether the assertion is expected (did the agent send a question to the received assertion's sender?). If so, the agent retrieves, from its history table, the original request for which it sent a question. The original request in hand, which could be a question on data, an action command, etc, the agent tries to process it again given the received assertion.

Therefore, through the coordination protocol, the agents make use of a collection of dialectical rules to process the exchanged messages and build message answers. This processing takes into account both the agent competences and their previous dialogues.

then retrieves the original request for which it sent its question, and processes it again using the information received by the television. Finally, it returns to the telephone an action execution acknowledgment which is then forwarded to the user.

Figure 7 – Agents' Coordination in the Illustrative

5. Case Study

In this section, we illustrate the Campus approach through a simple but real life case study. In our example, a human user, who is at work, asks his home telephone (using for instance his mobile phone) to *"Record the football match Real Madrid-Bayern München broadcasted on TV"*. We suppose that the user's house is provided with home appliances like a television, a DVD recorder, a Hi-Fi system, etc.

Let's assume that the DVD recorder is capable to performing the recording action, on the condition that the broadcast channel, hour and duration are provided. Suppose also that the DVD recorder uses the action name "Save" to appoint the program recording action.

Figure 7 depicts, through a sequence diagram, the home appliances coordination to respond to the user's request. This figure shows that the telephone sends the user's request to the DVD recorder. Indeed, it is not able to perform it alone, and it additionally knows, from the service directory, that the recording action belongs to the DVD recorder competences. Using its ontology, the DVD recorder matches the "Record" action asked by the user with its "Save" action. But then, it answers that it misses the broadcast channel, hour and duration of the required program. Following the coordination protocol, it builds the request: *"What is the broadcast channel, hour and duration of the program, given that its title is football match Real Madrid-Bayern München?"*. The television receives this request. Using its teletext, it is able to build an answer. It then sends an assertion to the DVD recorder stating the required information. The DVD recorder who receives this assertion uses one of the protocol dialectical rules to process the received message. It

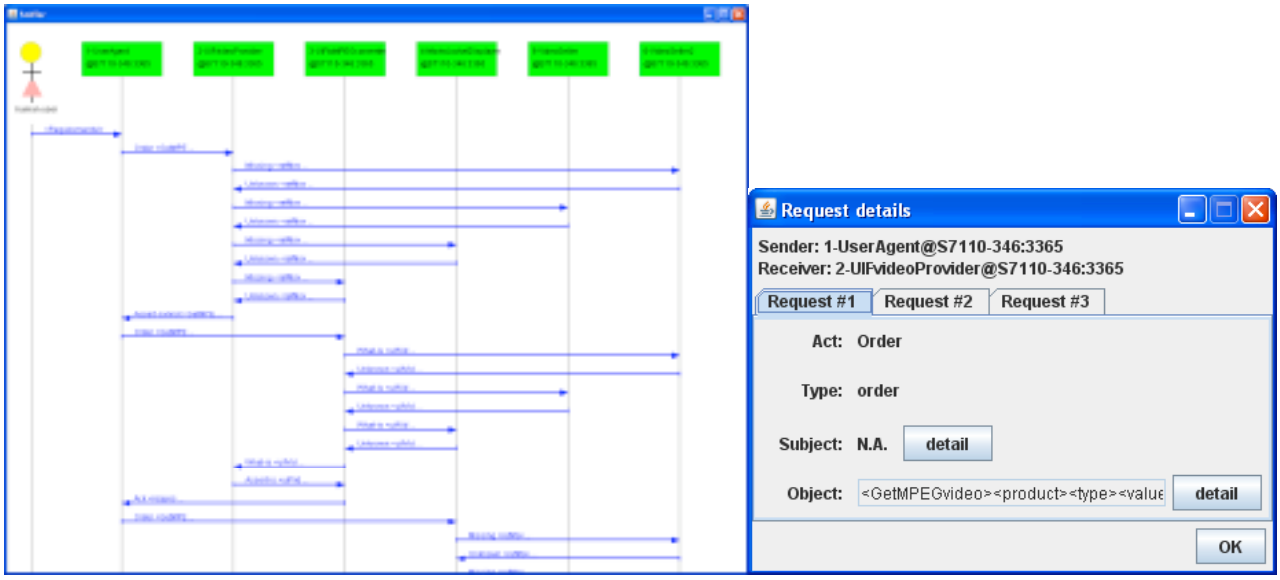

Figure 8 – Traces of agent interactions

Figure 8 shows, on the left side, a screenshot of the traces, captured by our application; of the agents' interactions, and on the right, a message sent composed of three requests.

6 Conclusions

In this paper we introduced Campus, a framework for the development of ambient intelligence applications. Based on multi agent system technology, Campus provides an infrastructure to develop innovative context aware applications that accommodate an ample spectrum of mobile and fixed devices. The support for semantic interoperability, offered by the communication and coordination layer, allows for the discovery, exchange and collaboration among hybrid devices, regardless of proprietary representations of information. We illustrate our approach with a real life example, in which we show the interaction among several software agents to record a football match. Future work includes a full, open software release of the framework and further experimentation.

Acknowledgements

This work was partly funded by the CNPq under grant no. 472010/2007-7, UPMC-LIP6 Professeur Invité Program, and the IBM Faculty Awards Program

7 References

[1] J. Ahola. Ambient Intelligence. ISTAG Scenarios for Ambient Intelligence in 2010, Feb 2001, Final report.

[2] K. Kranenborg, J. Stegeman, J. Lindenberg, W. Pasman and M.A. Neerincx. Improving service matching and selection in ubiquitous computing environments: a user study. Personal and Ubiquitous Computing 11(1):59-68, Springer Verlag, 2007.

[3] A. Gárate, N. Herrasti, and A. López. GENIO : an Ambient Intelligence Application in Home Automation and Entertainment environment. In Proc. Joint Conference on Smart Objects and Ambient Intelligence, pages 241§256, 2005.

[4] T.A. Lashina. Intelligent Bathroom. In European Symposium on Ambient Intelligence (EUSAI 2004), 2004.

[5] M. Vallée, F. Ramparany, and L. Vercouter. Composition exible de services d'objets communicants. In Proc. Ubimob'05, volume 120, pages 85§192, 2005.

[6] V. Ermolayev, N. Keberle, and S. Plaksin. Towards Agent-Based Rational Service Composition RACING Approach. In M. Jeckle and L-J. Zang, editors, Proc. of the International Conference on Web Services Europe, volume LNCS 2853, pages 167§182, Erfurt, Germany, September 2003. Springer-Verlag.

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl, 2001. specs/fipa00025/XC00025E.pdf, 2001.

[8]Y. Charif, N. Sabouret. An Agent Interaction Protocol for Ambient Intelligence, Proc. 2nd

International Conference on Intelligent Environments (IE'06), pp. 275-284, 2006.
[9] - B. Boehm, S. Wolf. "An open architecture for software process asset reuse," *ispw*, p. 2, 10th International Software Process Workshop (ISPW '96), 1996.
[10] V. Sacramento, M. Endler, H.K. Rubinsztejn, L.S. Lima, K. Gonçalves, F.N.do Nascimento, G. Bueno, MoCA: A Middleware for Developing Collaborative Applications for Mobile Users_IEEE Distributed Systems Online, ISSN 1541-4922, vol. 5, no. 10, October, 2004.
[11] H.K. Rubinsztejn, M. Endler, V. Sacramento, K. Gonçalves, F.N. do Nascimento, Support for Context-aware Collaboration, First International Workshop on Mobility Aware Technologies and Applications (MATA 2004), Florianópolis, LNCS no. 3284, pages 37-47, October, 2004.
[12] K. K. Breitman, M. A. Casanova and W. Truszkowski. Semantic Web: Concepts, Technologies and Applications. Springer Verlag, 2007
[13] Gómez-Pérez, A.; Fernadéz-Peréz, M.; Corcho, O. - Ontological Engineering - Springer Verlag - 2004.
[14] T. Finin, R. Fritzson, and D. McKay. An overview of KQML: A Knowledge Query and Manipulation Language. Technical report, University of Maryland Baltimore Country, 1992.
[15] FIPA Interaction Protocol Library Specification. http://www.fipa.org/
[16] Y. Charif and N. Sabouret. A Model of Interactions about Actions for Active and Semantic Web Services. In Proc. Semantic Web Service workshop at 3rd International Semantic Web Conference (ISWC'04), pages 31-46, 2004.
[17] Breitman, K. K.; Perazolo, M. . Using Formal Ontology Representation and Alignment Strategies to Enhance Resource Integration in Multi Vendor Autonomic Environments. Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems EASe 2007. Los Alamitos : IEEE Computer Society Press, 2007. p. 117-126.
[18] L. Mazuel, N. Sabouret. Degré de relation sémantique dans une ontologie pour la commande en langue naturelle, In Proc. 18th Journées Francophones d'Ingénierie des Connaissances (IC'07), pp. 73-83, 2007.
[19] Casanova, M. A. ; Breitman, K.; Brauner, D. F. Marins, A. . Database Conceptual Schema Matching. Computer (Long Beach), v. 40, p. 102-104, 2007.
[20] Brauner, Daniela F ; Casanova, M. A. ; Milidiu, Ruy . Mediation as Recommendation: An Approach to Design Mediators for Object Catalogs. In: 5th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2006), 2006, Montpellier.
[21] Breitman, K. K.; Brauner, D., Casanova, M.A.; Milidiú, R.; Gazola, A. - Instance-Based Ontology Mapping In: Fifth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems EASe 2008, Belfast, Ireland. IEEE Computer Society Press, 2008 – to appear.