Towards Mobility in Ambient Intelligence: Component Migration and Adaptation Strategies in the HI^3 Architecture

G. Varela, A. Paz-Lopez, J. A. Becerra, S. Vazquez-Rodriguez, R. J. Duro
Grupo Integrado de Ingeniería
Universidade da Coruña
Ferrol, Spain
gervasio.varela@udc.es

***Abstract*—Ambient Intelligence (AmI) systems try to provide a seamless experience to their users by offering their functionality in a transparent way. Ubiquity, namely the capability of the system to provide its services regardless of the location and the devices available, is fundamental to this goal.**

Ubiquity is a broad field covering diverse topics from security and privacy to system interoperability. In this paper we are mainly focused in the development of solutions for two associated issues of component mobility: The fault-tolerant physical migration of components with their associated runtime state, and the adaptation of components to the destination platform and physical environment.

These solutions are being addressed within our efforts in the development of general purpose middleware for Ambient Intelligence in the framework of the HI^3 project.

***Keywords*-ambient intelligence; ubiquitous computing; mobility; service collaboration;**

I. Introduction

One of the main goals of every Ambient Intelligence (AmI) system is to provide a seamless experience to their users. This goal is achieved in AmI systems by offering their functionality to users in the most transparent way possible. This transparent operation has multiple meanings. For example, the interaction with the user should use natural interfaces as much as possible, the system should show autonomous and proactive behaviour, it should require little intervention in system management and it functionality must be ubiquitous to the users.

Ubiquity, namely the capability of the system to provide its services regardless of the location and the devices available, is an important topic in AmI, as it is a key enabler to release the user from the management of its applications and data.

Ubiquity is not an easy goal. A lot of issues must be tackled in order to provide an ubiquitous experience to the users. Interoperability between the different systems involved, fault-tolerant distributed operation, physical mobility of system components, self-organization of components, heterogeneity of devices and services available in each environment, user location tracking, and a lot of security and privacy issues, are among the most important ones.

As we are starting to support mobility in HI^3 [1], an integral Ambient Intelligence platform we have been developing for the last few years. In this paper we are mainly focused in the mobility of components and some of its associated issues, like reorganization of components and renegotiation and adaptation of behaviour after a component migration process.

Component mobility has been addressed in many different fields. The distributed computing world has provided general solutions for service oriented systems and multi-agent systems, but special emphasis must be placed on more specific solutions coming from the ubiquitous computing field.

Agent mobility has been an important research topic in the multi-agent systems field because agents gain numerous interesting capabilities with mobility [2]. In this line many multi-agent development platforms have integrated support for agent mobility [3] [4] between different instances of the same platform, being the main problem the interoperability between heterogeneous platforms. This problem has been recently addressed with some efforts for real interoperability between FIPA [5] compliant multi-agent platforms [6], specifying a common interface to manage the lifecycle of agents in a FIPA platform, as well as common agent models and data encoding mechanisms to enable interoperability between platforms that do not support the same programming languages.

Service oriented approaches are mainly represented by OSGi based platforms. As OSGi by itself does not support mobility, some efforts have been recently made in that specific space, like [7] who focused on service diffusion and state synchronization. In [8] the authors also presented an OSGi solution with support for component mobility, but enhanced with semantic information about the services and tasks involved so they can be matched and rearranged during mobility processes. Another remarkable approach is related to fluid computing frameworks, like [9], that improve OSGi with support for application data and state synchronization for its usage in dynamic service oriented environments.

Ubiquitous computing research is also a prominent source of solutions for component mobility. The Gaia OS [10] and Aura [11] projects are both good examples of component

mobility solutions, providing support for physical component migration and state synchronization, as well as capabilities to find compatible devices in the new environment that fulfil the hardware requirements of the application.

These projects are mainly focused on the distribution and migration of applications that make use of distributed devices (sensors, actuators and others) available in each environment. Therefore, even if they address important areas of application mobility, like the physical migration of the application, the discovery of compatible devices and some sort of component reorganization, they do not take into account specific topics of AmI applications, like user preference management, adaptation of behaviour to the users and the environment, or context management.

We have been working on the development of a complete component mobility solution for AmI applications and its integration in HI^3 . The final objective is to have an integral solution to the mobility issues that affect AmI systems, providing solutions in topics like physical component migration, hardware requirements negotiation and discovery, component self-organization, component self-adaptation to user and environment preferences, user data privacy and component behaviour security, as well as application interaction adaptation. This is a long-term development effort and in this paper we are presenting some of the first developments in this area of HI^3 .

Section II will be devoted to a detailed exploration of the mobility issues existing in an Ambient Intelligence environment. By using a representative use case example, the issues and benefits of supporting component mobility will be described.

In III a brief description of the HI^3 platform will be given, along with an explanation of the solutions developed, referencing the previously described use case example.

Finally in IV we will present some conclusions and future work.

II. Needs, challenges and opportunities

As briefly stated in section I, ubiquitous access to AmI systems is an important requirement to achieve the transparent operation goals of Ambient Intelligence. Ubiquity implies a lot of benefits on the users, but it requires some complex requirements to be fulfilled. In this section we are going to present its main benefits and its associated requirements and challenges by exploring a feasible use case scenario.

We have imagined a scenario in which an elderly person that lives in a retirement home is visiting his son or daughter for the holidays. The retirement home in which this person lives is a very modern one and it uses an Ambient Intelligence system to help the caregivers. This system autonomously manages the security (window/door opening) and comfort (central heating) facilities of the building according to the profile of each intern. It also allows the medical staff to manage the diet of the interns. An overview of this system can be seen in Figure 1.

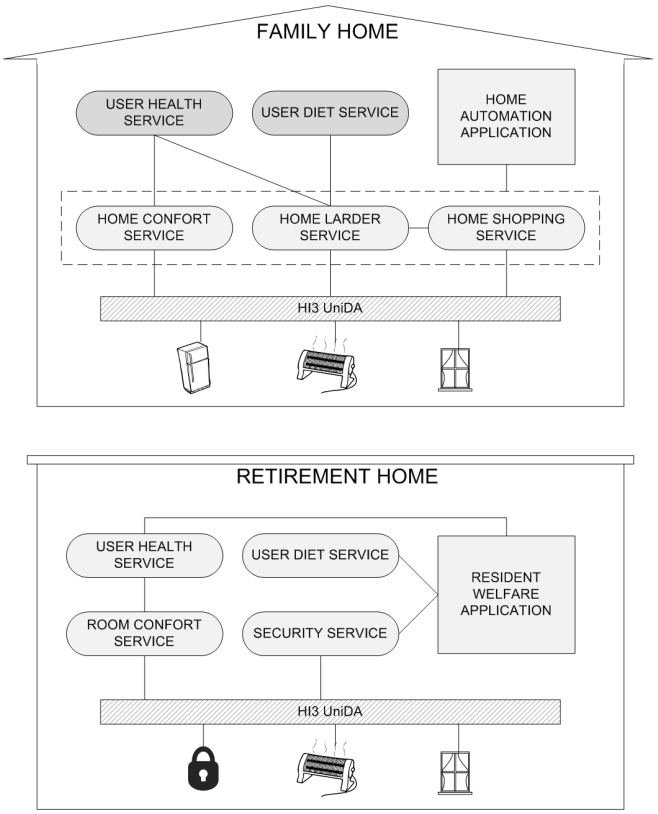

Figure 1. Usage scenario in an Ambient Intelligence context

The system has some services associated to the user which are in charge of providing information about the preferences and needs of each user (diet and room temperature requirements). A possible scenario for system mobility comes when the user leaves the retirement home to live in his son or daughter's home for some days. If the system presents mobility capabilities, the user preferences management services, *User Health Service* (UHS) and *User Diet Service* (UDS) can be migrated to the new home, so that they can interact with an existing AmI system to try to guarantee that the user's necessities and preferences are met.

The medical staff of the retirement home can use the *Resident Welfare Application* to manage care instructions for each resident. While living in the retirement home, the UHS manages the diet information of the users and allows the caregivers access to it so that they can do the shopping and prepare the food. The UDS uses other resident personal information to decide which room temperature is adequate for his health state. And the security service controls access to some places of the building or the exterior according to user permissions.

When a resident leaves the retirement home temporarily, his associated care services and their required data can be

migrated to his destination environment, in this example, a son or daughter's home. This migration can be achieved for example via the Internet or through a mobile device that temporarily stores the mobile components and deploys them in the end environment using a wireless connection. Once the components are migrated to the new home they must adapt themselves to the new environment.

For example, on one hand, the UDS, can now use a smart refrigerator, that is managed by the *Home Larder Service* (HLS), to query the available food and suggest meals with those ingredients, or even order more food from the supermarket using the shopping capabilities of the HLS. On the other hand, the UHS must use a new comfort service to manage temperature. The required comfort temperature may conflict with the temperature selected by the usual inhabitants of the home, so the UHS must negotiate the final temperature with the comfort service to achieve a consensus temperature.

As displayed by the example, there is not only the problem of physically migrating the components from one host to another, but, once migrated, the components have to be adapted to their new environment in many ways.

The benefits of migrating components instead of remotely accessing services and devices to achieve ubiquity may not be obvious sometimes, because similar functionality can be achieved while maintaining the different elements of the system distributed in their original hosts. But migration support can improve AmI systems in aspects such as network dependency, security, privacy or autonomy.

If the components are executed locally within the data sources and end devices, the network bandwidth requirements are effectively reduced and the network latency is eliminated. This last point is especially important for interactive real-time systems like AmI applications, that must control devices to interact with a human populated environment. Thanks to migration, AmI systems can even continue operating in environments that do not have network connection, or when the network fails.

Regarding privacy and security, by migrating components, instead of letting some external software access and manage user data, the user components, with their user defined security and access control measures, directly manage the devices and data. In the example we are exploring, the UHS, when migrated to a remote platform, uses local sensing data to make decisions about the temperature of the environment bearing in mind the preferences and necessities of the user. If the service was operating remotely, either local sensing data would be sent to a remote platform, which could imply a privacy problem for the rest of inhabitants, or personal information about the user would be sent to the *Home Confort Service*, in order to allow it to manage the temperature by itself, which could imply a privacy leak for the user.

However these benefits come with a lot of challenges that AmI systems developers must address:

- *Code transfer and state synchronization.* The code of the components of a system needs to be transferred from one host to another, and the receiver host must be able to execute it. Also the component state must be transferred and recomposed at the destination.
- *Resource allocation.* The destination host must have the resources (computing, sensing, interacting, etc.) required by the component to operate. Furthermore, the migrated components can enter in conflict with previous existing components about the use of a resource.
- *Security and privacy.* The system must safeguard the host environment from potentially malicious incoming components, for example, restricting the actions they can do. And vice versa, protect incoming components from malicious hosts, so that components perform the actions they are supposed to do.
- *User interaction.* The user interface components benefit most from mobility, as they need to use local devices and show low latency operation. The main challenges faced in this case are the heterogeneity of the interaction devices available in each environment, as well as the differences in interaction abilities of each user. Ubiquitous UIs must support multiple modes of interaction, as well as autonomous adaptation to the devices and interaction modes available for each environment and user.

As can be seen, Ambient intelligence platforms and middleware have in mobility an important source of challenges and opportunities to address in order to provide developers with powerful tools that facilitate the development of real ubiquitous applications.

III. Proposed solution

In this section we describe the concrete mobility solutions developed and their integration in our HI^3 architecture for AmI systems. First of all, a brief description of the HI^3 architecture will be given. This architecture provides support for the development and integration of applications and services in the framework of AmI. Additionally, this architecture provides us with the necessary concepts and tools to support other requirements such as mobility, connectivity and ubiquity.

Figure 2 shows the conceptual structure of the architecture. It is a layer based design that enables the division of system elements into levels, reducing the coupling between modules, facilitating abstraction and facilitating the distribution of responsibilities. The uniform device access layer (UniDA) is in charge of providing homogeneous and distributed access to the physical devices. The sensing and actuation layer provides components that are virtual representations of sensors and actuators in the physical environment, hiding part of the complexity of the real devices. The service layer provides tools to facilitate the development

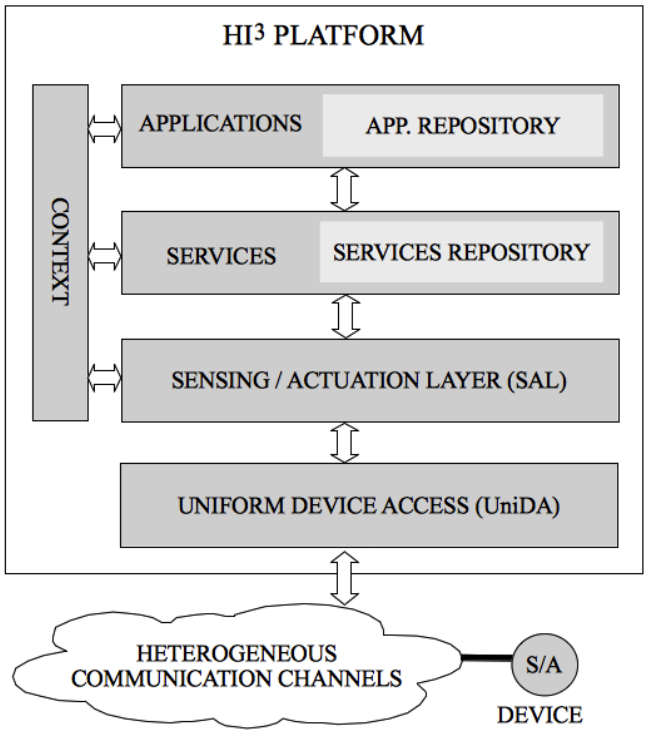

Figure 2. HI^3 software architecture

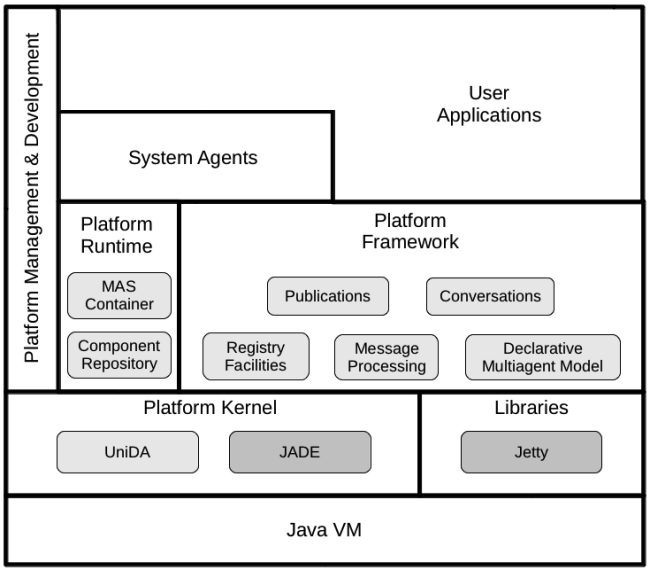

Figure 3. HI^3 multi-agent platform

and execution of services, including a service repository, semantic service definition capabilities, service composition facilities or service discovery mechanisms. The applications layer is the highest level layer and the one that hosts the elements representing and implementing applications that provide particular functionalities a user expects from the system. Finally, the context is a common access component for the three higher level layers of the architecture. Its main objective is to represent the current state of the environment.

In order to implement the previously presented conceptual model, we have chosen a multi-agent technology based approach, supported by the JADE agent framework. The main elements of the higher level layers in the HI^3 architecture are implemented as multi-agent systems (MAS) that collaborate for distributed task resolution.

Multi-agent system based platforms, in general, are a natural way to distribute intelligence and provide for component mobility, autonomy, scalability and fault tolerance. JADE, in particular, was designed as a general purpose multi-agent platform and not for the particular demands of AmI systems. Consequently, we developed some extensions in order to improve and adapt its capabilities. Specifically, we designed and developed a platform that is structured as displayed in Figure 3. It facilitates a container for AmI services and applications providing advanced features such as high level inter-agent communication facilities, multi-agent models with support for the declarative definition of components, development utilities or distributed management tools. This middleware promotes the division of large and complex AmI systems into highly decoupled components that dynamically and autonomously collaborate to solve complex tasks.

A more detailed description of this architecture and multi-agent platform can be found in [1] [12].

A. HI^3 Mobility Subsystem

As indicated above, we are implementing the proposed mobility solutions on top of the HI^3 architecture and multi-agent platform. In the HI^3 platform every system component is a multi-agent system (MAS), thus, as a starting point, we are considering MAS as atomic components that can migrate. This is a different approach from what is usually done in generic multi-agent platforms, as they often take agents as their atomic components. We have made this decision because in HI^3 every component (application, service or sensor/actuator) is a highly coupled collection of agents that collaborate to offer a very particular functionality, so it will generally be a good idea to keep them together in the same physical platform. Nevertheless we plan to support component fragmentation in the future.

Figure 4 shows a simplified class diagram of the structure of the HI^3 mobility subsystem. It shows the main elements of the subsystem and their relations in the component migration process. This procedure is divided into two main subprocesses. On one hand there is the physical migration of the component from one instance of the HI^3 platform to a new one, and on the other, the adaptation of the migrated component to the new platform and physical environment.

The physical component migration process is mainly managed by the *Component Mobility Manager* (CMoM) and the *Component Migration Manager* (CMiM). The CMoM

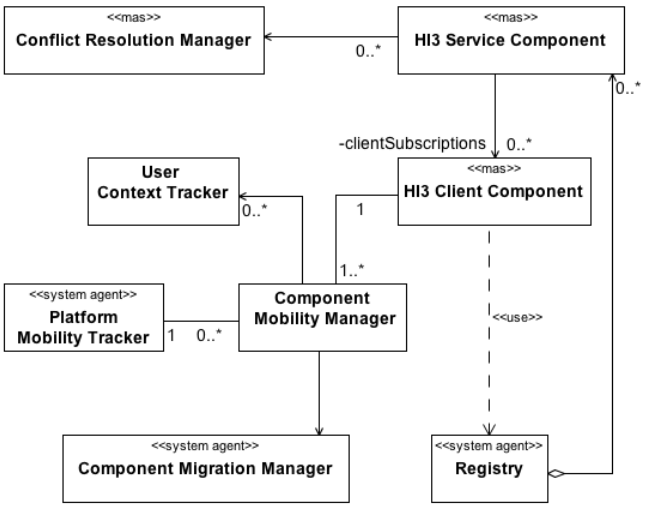

Figure 4. HI^3 mobility subsystem simplified class diagram

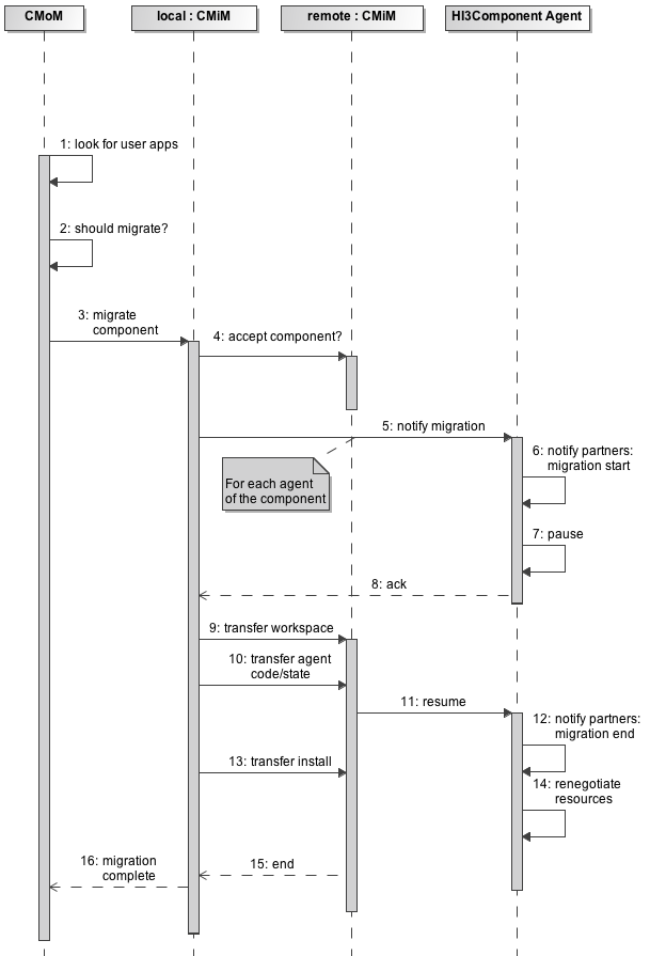

Figure 5. HI^3 component migration process

is in charge of monitoring the state of the users of the component, as well as the platform itself, and deciding when it is required or recommendable to migrate the component to another platform. This may occur, for instance, because the main user of the component is now in another physical location and a reduction of network latency is required, or it can be that the current platform is going to shut down because its battery is running low. The CMiM is in charge of managing the communications with the other physical platform and transferring the component code and state.

The adaptation of the migrated components to the new environment is mainly managed by the components themselves, but the system provides them with some helper capabilities. Service components are catalogued in the system registry according to their functionality, so when a component arrives in a new platform, it can use the registry to find components in the new local platform that provide the same functionality it was previously using. Furthermore, components can parameterize the functionality to request some specific behaviour, and the registry will provide the service that best matches the desired behaviour.

The next two subsections describe in detail these main two aspects of the mobility subsystem.

B. Component Migration Process

The migration process of an HI^3 component can start in two ways. The component can decide to move by itself because some internal logic says that it would be better to execute the component in a different platform (for example to have more computing resources). Or the CMoM can decide to move the component because some external changes in the environment, the users, or the platform, make it recommendable. This last case is shown in Figure 5.

When a CMoM (there can be many CMoMs to preserve scalability) detects a context change that may require the migration of one of its managed components, it decides whether it is required or not to move the component and, if affirmative, it requests the migration of the component to the CMiM (there is one for each HI^3 platform). To migrate the component, the CMiM must transfer the three elements that form an HI^3 component to the new platform at runtime. The *workspace*, a place where the component can store persistent data, the *install*, the component code, required libraries and configuration files, and the component *runtime state*.

The JADE agent platform, on which the HI^3 multi-agent platform is based, has integrated support for agent mobility. A JADE platform is divided into agent containers, so that one platform can have multiple containers in physically different hosts, and it supports agent mobility between containers. Unfortunately all containers depend on services provided by a main container, so it is a central point of failure. Furthermore, every container must be managed by the same administrators, so it is not possible to migrate components between platforms managed by different entities, which is a basic requirement for ubiquitous computing.

There exists an extension of the JADE platform to support inter-platform agent mobility [6], but we couldn't use their implementation, as it relies in the system class loader to obtain the code of the agents, and it does not support the MAS concept. In HI^3 we have decided to use private class loaders for each MAS, instead of having only one global class loader like default JADE, as a way to enforce code access privileges. We consider it important, not only due to security reasons, but because it allow components (MASes) to use different versions of the same code (for example libraries) without conflicting, which is specially important in open systems like those of AmI, even more when mobility is supported and components from different authors and users should coexists.

The migration process is mainly implemented by the CMiM. As can be seen in Figure 5, the CMiM starts by requesting the remote platform to accept the new component. If the remote platform accepts the component, it answers with metadata about the platform, so that the local platform can check if it matches the basic requirements of the component (mainly computing resources). Then it notifies the start of the migration to every agent of the specified component. Each agent notifies its partners that it is going to be temporarily paused due to a migration, then it prepares its workspace for the migration and pauses its execution.

When all component agents are paused, the CMiM compresses the workspace of the agent and transfers it to the new platform, that deploys it. Next it uses JAVA serialization to serialize the state of each agent, and packs this state with the code of classes directly used by the agent (extracted by introspection of the agent fields and declared superclasses and interfaces). The state and code of each agent is sent to the new platform, that resumes the execution of each agent.

Once the agents are already running in the new platform, the local platform compresses and sends the component installation directory, that contains the complete code of the component, as well as its required libraries. This transfer is carried out asynchronously, so that, thanks to the classes included with the agent state, agents can start running before having the complete component code.

While the few classes sent with the agent state can be enough to execute some simple agents, complex ones would need access to more code even before the component installation transfer is complete. For this, the class loader of the moved agents is able to dynamically request classes from a remote CMiM (the CMiM of their previous local platform). The CMiM will check if the requesting agent is one of the migrated agents of the component holder of the requested code, and will pack and send the class code.

Once the component install transfer is finished, the remote CMiM informs the local CMiM of the successful deployment of the new component and the end of the process. Next, the local CMiM stops the execution of the moved component and removes its agents (previously paused) from the platform.

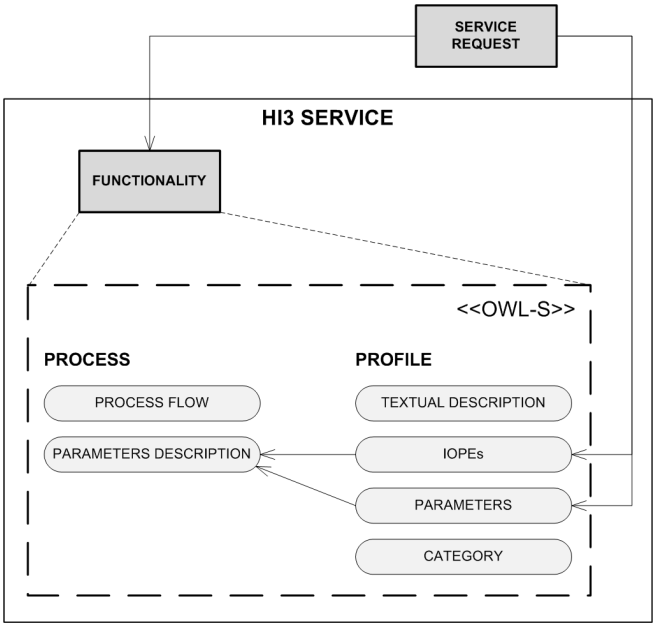

Figure 6. HI^3 service semantic description model

C. Negotiation and Conflict Resolution Strategy

As we have introduced before, the HI^3 mobility subsystem involves two main processes: the physical migration of the components and the adaptation of the migrated components to the new environment. Now, we focus on the second process.

Taking a look at the overall migration process, we immediately appreciate the convenience of automating and generalizing the process of restarting the execution of the migrated components on the target platform. Thus, the system developed here provides a common model and a set of facilities the migrated components need in order to adapt their configuration and tasks to the new execution conditions. Specifically, the mobility system manages the negotiation process in charge of discovering the most appropriate local services that provide the functionalities the migrated component needs to continue its execution. This whole process is supported by the semantic description capabilities provided by the HI^3 architecture, which facilitates the development of self-describing modular components, that can be published, automatically located and invoked across the platform.

The HI^3 platform incorporates a description model that every HI^3 service must comply with. It is used in order to specify the service capabilities to the system. Particularly, this model allows developers to describe components according to the functionality provided, their external communications interface and their internal structure in terms of agent instances.

Figure 6 shows a simplified view of the semantic de-

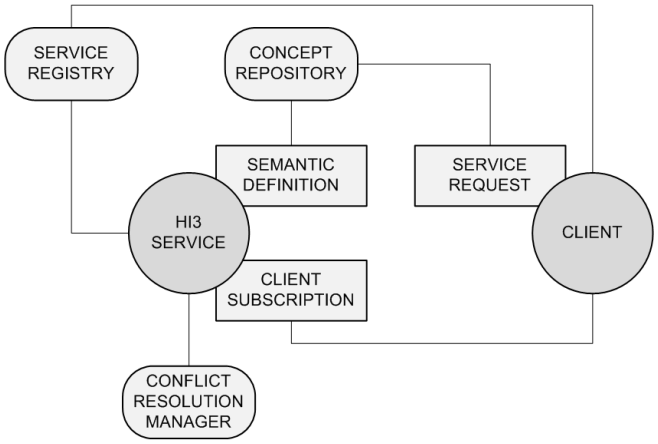

Figure 7. HI^3 service request negotiation model

scription of the functionalities provided by HI^3 services. We propose a solution based on the utilization of current semantic web service (SWS) technologies and initiatives [13]. Currently there exist two main initiatives in the field of SWS description, OWL-S [14] and Web Service Modeling Ontology (WSMO) [15]. We have chosen OWL-S, an OWL-based webservice ontology for the specification of what the service provides for prospective clients, how it is used and how one interacts with it.

We use the OWL-S Profile model to specify two different aspects of the functionality of a service: the information transformation (represented by inputs and outputs) and the state change produced in the environment by the execution of the service (represented by preconditions and effects). Additionally we use the OWL-S Process model to define the semantic concepts used by the Profile model. This specification can be used by a client component to create parameterized *service requests* to require some desired behaviour from the system.

Using these capabilities we designed the service request negotiation model presented in Figure 7. It shows the main elements of the mobility subsystem that are in charge of managing the adaptation of the migrated components to the new platform and environment. In this process, when a migrated component arrives at a new platform, it can use the service registry of the platform to find services that provide the same functionality it was using before the migration process. For this purpose, the migrated component creates parametrized *service requests* semantically describing the desired functionalities. Additionally, each service has a collection of clients and its previous parameterized requests (suscribers) and incorporates capabilities to manage conflicts in the access to its functionality, so, when a new component requests an action and there can be a conflict with some resources managed by the service, they have integrated capabilities to negotiate the action with other currently active clients subscribed to the service. The detailed sequence of actions for this negotiation process is shown in Figure 8.

To better illustrate the service negotiation process we can imagine an example in the scenario from section II. The UHS of the elderly user sends a parameterized request to the *Home Confort Service* (HCS) requesting a temperature of 21 C. The request also includes a parameter that indicates the flexibility of the result accepted by the requesting component, for this example it would be low, because the temperature of the home is critical for the health of the user. As shown in Figure 8, the client component sends the request to the service registry of its local platform. The registry searches in its database of available services for those that match the semantic description specified in the request. Once found, it stores them in a cache to accelerate future requests, and sends the paramaterized request to all the services found. Each service uses its *Conflict Resolution Manager* (currently supporting a basic fuzzy logic strategy) to calculate an estimated result after processing the request, and sends that as a response to the registry. The registry finally sends a response to the client component indicating the available services and their estimated results, so that the client can select the one that better suits its needs. The negotiation process can also be carried out directly by the client. In that case, the request sent to the registry would not be parameterized, so the registry will only find those available services that match the semantic description of the request and return them to the client, that would be in charge of sending the parameterized request to the services to negotiate the results and confirm the request.

These strategies constitute a first step towards the autonomous adaptation of systems to the changing environments envisioned in fields like ubiquitous computing, ambient assisted living, ambient intelligence or human-centered computing. The proposed negotiation and conflict resolution strategy provides HI^3 components with a series of mechanisms that allow them to adapt to new environments, with the restriction that every environment must share a common semantic description model. Nevertheless, to achieve a complete solution it is necessary to improve these strategies with characteristics such as interoperability among heterogeneous systems, computational load balancing, user and environment data privacy safeguard measures or capabilities to preserve the functional integrity of the whole system.

IV. Conclusions

Application mobility, as a key enabler to achieve ubiquity, is an important characteristic of Ambient Intelligence systems, because, as we have highlighted in this paper, it provides important benefits in areas such as system operation latency, system autonomy or user data privacy. However, this particular topic is usually overlooked by AmI projects, that often rely exclusively on distributed operation to achieve ubiquity.

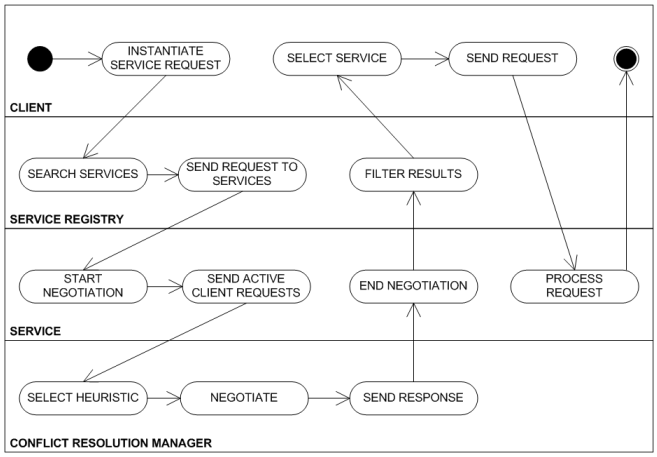

Figure 8. HI^3 service request negotiation process

Mobility is a complex topic by itself, but when applied in an AmI context, its complexity increases even more, as new issues related with interoperability, system autonomy, device access or security and privacy are introduced. In this paper we have described our first efforts in the integration of mobility support within a general purpose ambient intelligence development platform. In particular, we have addressed mobility by starting with two basic areas:

On one hand, the physical migration of components. As it has been previously addressed by other authors in the literature in many different ways, we only had to adapt existing approximations to the particularities of an AmI multi-agent based platform, paying special attention to code access privileges inside the platform, as well as latency and fault-tolerance issues during the mobility process.

On the other hand, component self-adaptation and reorganization, which is also an important and studied topic, especially in the service oriented world and the ubiquitous computing field has been considered. In this case we have applied solutions coming from those worlds, like OWL-S, adapting them to some particular issues that appear during the migration process of components within an AmI system.

Nevertheless, a lot of work is still required in order to obtain a complete mobility solution for AmI. From extending the current developments, for example with a semantic abstraction layer to decouple the solution from OWL-S and achieve better interoperability, to new developments in areas like protection of the mobile components from their hosts and viceversa, user interface adaptation after a migration process or data privacy and synchronization.

ACKNOWLEDGMENT

This work was partially funded by the Xunta de Galicia and European Regional Development Funds through projects 09DPI012166PR and 10SEC019E.

REFERENCES

[1] A. Paz-Lopez, G. Varela, J. Monroy, S. Vazquez-Rodriguez, and R. J. Duro, "HI3 Project: General Purpose Ambient Intelligence Architecture," *Proceedings of the 3rd Workshop on Artificial Intelligence Techniques for ambient Intelligence, AITAmI08*, pp. 77–81, 2008.

[2] D. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*, vol. 42, no. 3, pp. 88–89, 1999.

[3] "Jade. java agent development framework." [Online]. Available: http://jade.tilab.com

[4] "Aglets. java mobile agent platform and library." [Online]. Available: http://aglets.sourceforge.net

[5] "Fipa. foundation for intelligent physical agents." [Online]. Available: http://www.fipa.org

[6] J. Cucurull, R. Martí, G. Navarro-Arribas, S. Robles, and J. Borrell, "Full mobile agent interoperability in an IEEE-FIPA context," *Journal of Systems and Software*, vol. 82, no. 12, pp. 1927–1940, Dec. 2009.

[7] D. Preuveneers and Y. Berbers, "Pervasive services on the move: Smart service diffusion on the OSGi framework," *Lecture Notes in Computer Science*, vol. 5061, pp. 46–60, 2008.

[8] G. Pan, Y. Xu, Z. Wu, L. Yang, M. Lin, and S. Li, "Task Follow-me: Towards Seamless Task Migration Across Smart Environments," *IEEE Intelligent Systems*, 2010.

[9] J. S. Rellermeyer, "flowSGI A Framework for Dynamnic Fluid Applications," Ph.D. dissertation, 2006.

[10] M. Román, C. Hess, R. Cerqueira, R. Campbell, and K. Nahrstedt, "Gaia : A Middleware Infrastructure to Enable Active Spaces," in *IEEE Pervasive Computing*, vol. 20. Citeseer, 2002.

[11] J. Sousa and D. Garlan, "Aura: an architectural framework for user mobility in ubiquitous computing environments," *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, no. August, pp. 29–43, 2002.

[12] A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. Becerra, and R. Duro, "Some Issues and Extensions of JADE to Cope with Multi-agent Operation in the Context of Ambient Intelligence," *Trends in Practical Applications of Agents and Multiagent Systems*, pp. 607–614, 2010.

[13] B. Dutta, "Semantic Web Services: A Study of Existing Technologies, Tools and Projects," *DESIDOC Journal of Library & Information Technology*, vol. 28, no. 3, pp. 47–55, 2008.

[14] "Owl-s: Semantic markup for web services." [Online]. Available: http://www.w3.org/Submission/OWL-S/

[15] "Wsmo: Web service modeling ontology." [Online]. Available: http://www.wsmo.org