Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 529–536

ISBN 978-83-60810-22-4

Services Composition Model for Home-Automation peer-to-peer Pervasive Computing

Juan A. Holgado-Terriza, Sandra Rodríguez-Valenzuela
Software Engineering Department
University of Granada, Granada 18071, Spain
Email: jholgado@ugr.es, sandra@ugr.es

***Abstract*—Collaborative mechanisms between services are a crucial aspect in the recent development of pervasive computing systems based on the paradigm of service-oriented architecture. Currently, trends in development of services computing are taking into account new high-level interaction models founded on services composition. These services make up their functionalities with the objective of creating smart spaces in which services with different purposes can collaborate to offer new and more complex functionalities to the user transparently. This leads to the creation of collaborative spaces with value-added services derived from the composition of existing ones. However, there are many aspects to consider during the development of this type of systems in pervasive spaces, in which the extensive use of embedded devices with limited characteristics of mobility, computing resources and memory, is a large handicap. This paper describes a model of services composition based on a directed acyclic graph used in a services middleware for home-automation, in which we work with loosely coupled services-oriented systems over the peer-to-peer technology JXTA. The presented composition model guarantee the acyclicity of the composition map between services as well as favours the building of collaborative light services using peers as proactive entities, which could be executed on embedded devices. These ones are capable of establishing dynamic intercommunications, synchronizing with others and form coalitions to cooperate between theirs for a common purpose.**

I. Introduction

THE PROLIFERATION of smart communication devices and the extensive use of the Internet in any device (e.g. mobile device) have brought the need of integrating business process models into any kind of system [1]. Business processes provide complex and sophisticated services and products as consequence of the massive penetration of data from internet-enabled devices, the data management capabilities of mobile devices and any other wearable and embedded devices, the context information depending on its location, its physical and computing environment, and even its human users. Services such as videoconferencing, VoIP, ambient assisted living appliance, distance education or learning, smart security home are now possible. Moreover the integration of more resource-full computing devices into the home environment may assist us by means of autonomous decision making based on the context or available data which is part of the Smart Home concept [2]. The convergence between the operation given by in-house devices and the business process, require computing models where the interactions between the device operation and the business process should be more natural. The technological advances necessary to build a pervasive computing environment fall into four broad areas: devices, networking, middleware and applications [3].

Many years ago, Weiser defined the main lines of pervasive computing research that are focused initially on hardware issues such as the reduction in size and power consumption, the processing power, the wireless communication protocols and the invisibility, transforming the devices in invisible objects [4]. Other key features of a ubiquitous environment are the dynamic reconfigurability, modularity, extensibility and portability. Thus, the middleware platform for pervasive systems must solve problems such as interoperability, heterogeneity and transparency with respect to devices, as well as dynamic discovery, selection, composition and adaptation of components [5]. Today, the principles of the SOA (Service Oriented Architecture) paradigm are the most widespread and used for the development of pervasive computing systems [6].

However, the development of services-based software architecture may become very complex in ubiquitous computing when the interaction space includes embedded devices of small size. The resources of these devices are often too limited to run certain processes and store information. Besides, they possess limited capabilities in terms of processing power, memory, time battery life and bandwidth [7]. These characteristics mean that the application development and business processes in ubiquitous environments requires new specific computer models, and therefore, new software infrastructures that support these applications, taking into account that they must be integrated into embedded execution environments, i.e., devices with limited resources.

By means of services composition a service can make up its functionality in base of the collaboration with the rest of services. Hence, a collection of collaborative services can provide a way of creating smart spaces that offer new and more complex functionality transparently to the user [8]. The most common service collaboration models are based on the orchestration and choreography of services. In the service orchestration the execution flow control is always responsibility of one of the parties involved in the collaboration, while in the service choreography any of the entities involved in the collaboration can take part in the interaction [9].

There are several service composition approaches which study how to make a composition from existing services in order to achieve a more complex functionality that typically is not provided by a single available service [10]. This paper

presents a model of services composition based in a directed acyclic graph and used in a services middleware for home-automation. The middleware is based on the SOA paradigm and it has been built over the peer-to-peer framework JXTA, which has been selected by the good scalability and the decentralized nature achieved by the P2P systems [11]. The service composition model is based on the orchestration principles in which the interaction control is responsibility of each service, as well as the execution order of operations and flow of messages and transactions required in the collaboration. The composition model in the service middleware often differentiates between simple and composite operations. The execution of a composite operation involves the execution of a set of requested operations on several collaborative services. The presented model of services composition favours building of collaborative decentralized lightweight services, which could be executed on embedded devices, using peers as distributed and dynamic entities.

The rest of the paper is organized as follows. Section 2 introduces the most representative aspects of the SOA-based middleware which has been developed taking into account the constraints imposed by the use of devices with limited resources. Section 3 specifies the details of the service composition model designed over the platform. Section 4 introduces the communication patterns involved in the service composition model. Section 5 explains the behaviour of a service when a composite operation is executed. In section 6 we will show an example which involves several services with composite operations. Finally, section 7 reviews the related work and shows the future research, before concluding in section 8.

II. SOA-Based Middleware for Pervasive Computing

DOHA (Dynamic Open Home-Automation) is a services platform for the access, control and management of home-automation systems that facilitates the construction of dynamic, scalable and pervasive applications, based on a set of lightweight and independent services. The DOHA services platform is based on the SOA paradigm and uses the peer-to-peer middleware JXTA as the support platform.

DOHA abstracts the physical distribution of devices and its management by a set of high-level collaborative services, as shown in Fig. 1. The platform promotes the collaboration between services which involve communication between peers at a lower level, and the interconnections between devices across different networks placed on diverse subnets at the lowest level.

DOHA is supported by the peer-to-peer framework JXTA, which allows any device connected to the network to collaborate and communicate as a peer, providing positive features such as interoperability, platform independence and ubiquity. This enables integration in the same network of nodes that can represent services, physical devices, applications requiring the use of services, etc., leading to a system easily and naturally scalable, where new features and new devices and services

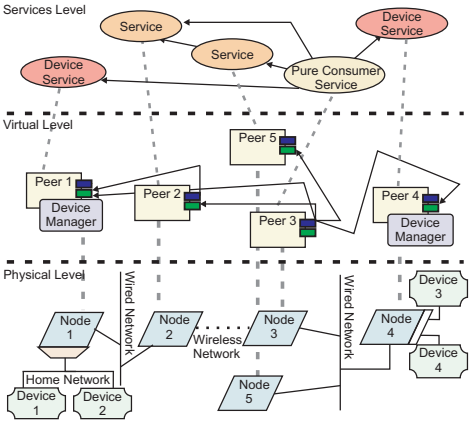

Fig. 1. Levels of abstraction of DOHA platform.

can be added in a more flexible manner. There are several approaches which implement a distributed collaborative model based on P2P technologies. For example, the JXTA-Overlay is a JXTA-based P2P platform designed with the aim of leveraging the capabilities of Java, JXTA and P2P technologies to support distributed and collaborative systems [12]. As in the Barolli et al. model, the DOHA platform has been successfully applied to large-scale systems with powerful embedded devices [13]. However, when the memory resources of embedded devices are scarce, there is no space for the full JXTA middleware. In this case a variation of JXTA for J2ME (CLDC-MIDP2) is deployed on embedded devices [14].

The DOHA platform takes into account another important aspect in pervasive systems, the large number of heterogeneous hardware devices that may be part of a network, and how the hardware interaction is managed from the service level (e.g. HVAC, temperature sensor, alarm clock). The JavaES (Java Embedded System) framework is used to enable the operation with different types of physical devices (e.g. appliances, sensors, actuators) in the environment. The access to the hardware is carried out in a standardized fashion, since JavaES abstracts the specifics hardware capabilities of each embedded device [15].

A service in the context of DOHA is an autonomous self-contained component capable of performing specific activities or functions independently, that accepts one or more requests and returns one or more responses through a well-defined, standard interface. There are two special types of services in DOHA: the *Device Service* and the *Pure Consumer Service*. The *Device Service* can interact with the physical devices of the environment and it provides physical device control. The *Pure Consumer Service* does not provide access to other services; it often provides access to end users applications, usually with a graphical user interface, and it does not offer functionality to the rest of services.

A DOHA service has an internal structure organized in a set of software layers. The multilayer structure facilitates the decoupling of tasks performed by a service in cohesive components; e.g., separating the task of requesting services

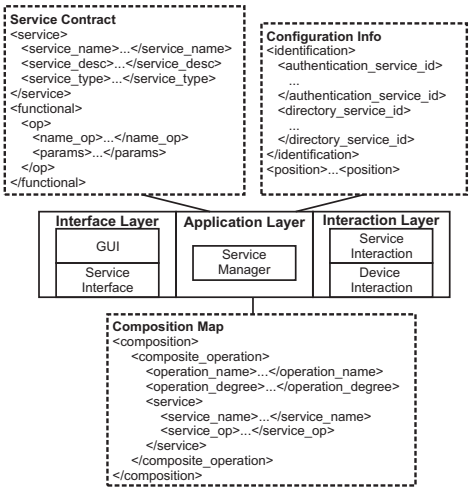

Fig. 2. Anatomy of a DOHA service.

from the task of providing services. Fig. 2 shows the anatomy of a DOHA service which provides a model to design and implement services and also allows managing the behaviour flow of the service in each step of its execution.

The *Interface Layer* is the public access point of the service which provides the functionality of the service in terms of operation (simple or composite) that can be invoked from any other consumer service. The layer is responsible of receiving the requests, executing the service operations by forwarding them to the *Application Layer* and finally returning a response. Services may have a graphical interface to allow the user's access. In this case, the *Interface Layer* should include the *GUI component*.

The *Application Layer* is the real core of the service and it is in charge of processing the operations from the received requests given by the *Interface Layer*, supervised by the Service Manager component. The Service Manager handles the execution of an operation in the context of the service, including the necessary invocation of operations of any collaborative service, and provides an adequate response to the service requesters. The decoupling of the *Application Layer* with respect to the *Interface Layer* allows us a way to control the "stateless" feature of the service, since the use of state information may adversely affect its availability and scalability.

All DOHA services are stateless from an external point of view, since the *Interface Layer* always provides a response to any request carried out by any service at any given time. But in many occasions the service could be required to maintain state information; for example, when the service virtualizes the state of a physical device (e.g. temperature sensor, illumination sensor) with a logical state enclosed in the service. The state-dependent part is bounded and limited by the *Application Layer* and it is addressed by the Service Manager. Therefore, it is imperceptible to the outside.

Finally the *Interaction Layer* contains the logic necessary to make possible the communication, the invocation of operations to other services or directly to devices (e.g. sensors or actuators), and the recovery of responses which are delivered later to the *Application Layer* in order to complete the execution of the running service operation. The *Service Interaction component* is responsible of managing the collaboration with other services, acting as a client of consumer of these services. On the other hand the *Device Interaction component*, which only appears in the *Device Services*, interacts with JavaES which gives a hardware abstraction to access the physical devices in the environment.

The deployment, start-up and execution of the service require knowing additional information that is managed by the service during its life-cycle at runtime. This information is enclosed into three descriptive documents which form the base of the service specification. These ones are the *Service Contract*, the *Service Composition Map* and the *Service Configuration*. Each of them abstracts a fundamental aspect of the service within the platform and contextualizes his collaborative behaviour with other services. The *Service Contract* is a public resource exchangeable between services, containing a description of the requirements, restrictions and functionality of a specific service. This information will be exploited by the rest of services in order to be aware of the functionality offered by the service and later make use of it. The *Service Composition Map* establishes the relations among services and the operations they are to perform, in order to carry out composite operations. This information is private and only accessible by the service itself, which is the only one who knows with which other services it is supposed to interact, in order to carry out a composite operation. Finally, the *Service Configuration* is needed to initiate the execution of the service, and it contains configuration parameters related to the software infrastructure that provide support to the service such as JXTA and JavaES. Related to JXTA, the *Service Configuration* encloses the identifier of the *Authentication Service* (peer group id) and the identifier of the *Directory Service* (rendezvous peer).

III. Service Composition Model

The service composition model of the DOHA platform is based on the activities that a service should perform when it needs to collaborate with other services to complete a requested operation. Dynamic modelling of services, such as the execution flow of a service, can be shown from the point of view of the type of the operation to be carried out and what activities the service must perform for it. The types of operations that services can perform are simple or composite. A simple operation is a single transaction that the service can perform by itself, i.e., the service has all the resources necessary to carry it out and it does not require interaction with other services. In contrast, a composite operation involves the invocation of one or more operations in one or more services. The service that owns the composite operation is responsible of its execution and it must interact with the services involved in the operation, the *requested services*, to get the necessary functionality in order to complete the whole

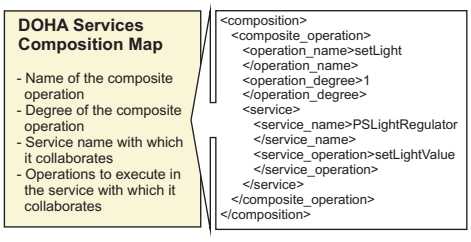

Fig. 3. Structure of the *Service Composition Map* and an example represented in XML.

operation. A service that only has simple operations is called a basic service; whereas a service that implements composite operations invoking other services is called a composite service. A *Device Service* is an example of a basic service that is composed only of simple operations. The composite operations are the base of the collaboration model of the platform and are listed by the *Service Composition Map*.

In the XML code of Fig. 3 we can observe the structure of a DOHA *Service Composition Map* that lists the requested services (using meta-data enclosed into <*service_name*> tags) and the corresponding invoked operations required for the execution of a composite operation available on the service. In this case, the *Service Composition Map* belongs to the *PSLight* service, which make use of the *PSLightRegulator* service to manage the composite operation *setLight()*.

Once a service is running, it can interact with other requested services of the platform to perform composite operations. The service is only aware of these requested services, allowing the collaboration with them to be carried out without user intervention, creating autonomously collaborative applications at service level. However, the service is not aware of the rest of the services available in the platform.

Service composition can be modelled using graph theory. The composition map of a service is formed by composite operations. Each composite operation op of a service S can be defined by a directed graph $G_{op_S} = (o_{op_S}, V(G), L(G), E(G))$ where:

- o_{op_S} is the main vertex of the graph, and corresponds to the origin service S of the composite operation op_S.
- $V(G)$ is the set of vertices of the graph where each vertex represents a service invoked from the operation op_S, i.e. a required service.
- $L(G)$ is the set of labels where each label embodies an invoked operation in a required service.
- $E(G)$ is the set of edges related with two vertices where the origin vertex is o_{op_S}, the destination vertex is an element of $V(G)$ and the label of the line is an element of $L(G)$. Accordingly, each element of this set is defined by the function $edge_i(o_{op_S}, op_i, v_i)$, where op_i must be an invoked operation of a required service v_i, verifying that $E(G) \subseteq o \times L(G) \times V(G)$.

The composite operation graph is directed because the arcs or operations between vertices always have a sender and a receiver. The former is the service that starts or invokes a composite operation and which performs the request, and the latter is the requested service which is the owner of the invoked operation. Thus, we can define a syntax based on graph theory to model the composition between services. By means of these premises, the full composition map of the service $map(S) = G_{op_{1_S}} \cup G_{op_{2_S}} \cup G_{op_{3_S}} \cup ... \cup G_{op_{n_S}} = \cup_i G_{op_{i_S}}$ is a directed super-graph formed by the union of all the composite operations of the service S.

The composition graph of a given service may contain calls to several services operations in the same composite operation, but these operations are performed in a sequential fashion, not nested. The depth of the composition graph of a service is always of one level, however, we must take into account that when a service starts a composite operation, it does not know if one of the operations in the requested services is composite too. In this case, we could have a chain of composite operations. Because of this, we define the function $degree(G_{op_S})$ as the complexity degree of an operation which establishes the depth of its graph. A simple operation has complexity degree 0, i.e. $degree(op) = 0$. The complexity degree of a composite operation G_{op_S} is defined by $degree(G_{op_S}) = max(degree(op_i)) + 1, \forall op_i \in L(G)$. We can say that the degree of a service is the maximum degree of all the composite operations of its composition map, i.e. $degree(map(S)) = max(degree(G_{op_{i_S}})), \forall G_{op_{i_S}} \subseteq map(S)$.

The function $degree()$ is used to ensure the acyclicity of the service composition graph. We have established the following restrictions on the construction of composite operations:

- All composite operation must finish with a simple operation.
- A composite operation can only invoke another composite operation with lower complexity degree. A composite operation with degree 1 can invoke only simple operations, i.e. operations with degree 0. A composite operation with degree n can invoke another composite operations with maximum degree $n - 1$.

We can see an example of composite operations degree in Fig. 4. The degree of the simple operation, *setTemperatureValue()* in the service *Temperature Regulator Service*, is equal to 0. This operation is invoked by the composite operation *setTemperature()* of the service *Temperature Service*, therefore this operation has degree 1. Also, the operation *setTemperature()* is invoked by the composite operation *setProfile()* of the service *Comfort Service*. The operation *setProfile()* also invoke the simple operation *tvOn()* of the service *TV Control Service*. Hence, we can say that the degree of the composite operation *setProfile()* of the service *Comfort Service* is 2, because that is the maximum degree of its invoked operations plus 1.

Based on the restriction imposed on the invocations between composite operations for the function $degree(op)$ we have defined the axiom $edge(o_{op_{v1}}, op_{v2}, v2) \rightarrow degree(op_{v1}) > degree(op_{v2})$. This axiom imposes that the degree of a composite operation op_{v2} of $v2$ must be strictly less than the degree of the composite operation in o which invokes it, op_{v1}. Using

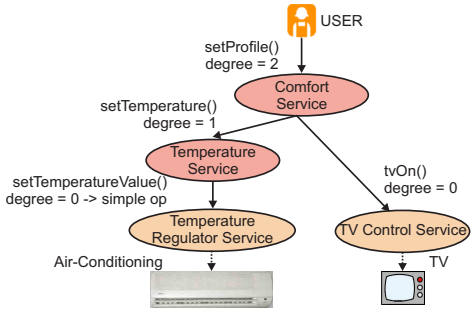

Fig. 4. Degree in composite operations.

this axiom we can prove the property of acyclicity of the *Services Composition Map*. By construction, the model verifies the acyclicity of the service composition graph with degree 1, a composite operation invokes only simple operations. We could consider a more complex composition graph with several services with composite operations interacting, $v1$, $v2$ and $v3$. This example must verify that $edge(o_{op_{v1}}, op_{v2}, v2)$ and $edge(o_{op_{v2}}, op_{v3}, v3)$, which according to the axiom defined above involve that $degree(op_{v1}) > degree(op_{v2})$ and $degree(op_{v2}) > degree(op_{v3})$. By transitivity it could also be verified that $degree(op_{v1}) > degree(op_{v3})$. Therefore we can affirm that no matter how complex is the composition map of a service built on DOHA, the graph associated with it is a directed acyclic graph.

IV. Communication Mechanisms In Services Composition

The DOHA communication mechanisms are based on the SOA scheme of publication, discovery and invocation of requests, which is implemented using JXTA primitives and protocols. A DOHA service takes the peer as the entity that provides communication capability. JXTA shares some of the SOA principles. The JXTA Peer Discovery Protocol is used for the publication and discovery of services using the concept of advertisement as exchanged information between peers. In contrast, for inter-peer communication JXTA provides three basic transport mechanisms, each one providing a different level of abstraction. The endpoint is the lowest level transport mechanism, followed by the pipe, and then finally, at the highest level, there are JXTA sockets [16]. DOHA exploits the pipe as the basic facility in order to provide a communication channel between peers in DOHA.

The DOHA services as peers publish their presence making use of advertisements to allow other services to discover it. Therefore, the services need discovery mechanisms that allow communication among services with different locations and functionalities. In JXTA special peers exist, the Relay Peer and the Rendezvous Peer, which provide remote discovery of advertisements between peers in different networks. DOHA treats these types of peers in the implementation of a Directory Service.

From a purely P2P point of view, JXTA does not require a specific service node to provide registry services. However, JXTA is versatile enough to accommodate a brokered mode of operation as well, whereby Rendezvous/Relay nodes can take over the role of the Registry and Lookup servers. The Rendezvous/Relay peer nodes can manage requests and responses to facilitate communication between pairs of peers [17].

When a service discovers new advertisements in the network, it stores them in the local cache of its peer. If this service is also connected to the Service Discovery (Rendezvous peer), it can also perform a remote search and discover more advertisements. A Rendezvous peer has the responsibility of coordinating all peers in the JXTA net and propagating the messages and advertisements remotely. If the peers are in separate subnets, we can use any one Rendezvous peer to manage the reception of remote messages and broadcast those within its local net. If the local net has a firewall or NAT (Network Address Translation), the peer can use a Relay peer to surpass it and allow remote discovery of its advertisements by peers of other external networks.

Each peer of any DOHA service has a pool of pipes with a name, an unambiguous identifier, and a pipe advertisement for each one. This information is known by the rest of the peers in the peer group due to its publication into a pipe advertisement, and in addition it could be delivered to remote networks through the Rendezvous Peer. When a service wants to establish communication with another, first it seeks one of those announcements in the cache of his peer or from the Directory Service, and then it uses this information for the pipe creation.

The peer associated with each service have a special pipe, named the interaction pipe or *PipeI*, to claim the use of collaborated services as an input pipe in a customer service. This pipe allows the peers to act as consumers in the network.

V. Behaviour Flow of the Service Composition Management

The behaviour of a service can be represented using an UML activity diagram that models how the services interact with each other. The execution flow in a service can be distinguished in function of the abstraction layer where the activity takes place. Therefore, we have partitioned the set of activities of a service according with the layer in which these operations take place: Interface, Application and Interaction. Each partition has a group of actions with respect to their responsibility as we show in Fig. 5.

The services execute the activity of *waiting for messages* in the partition associated with the *Interface Layer*. During this activity the services listen to their input pipes waiting to receive requests from other services. Once a message arrives, it is delivered to the *Application Layer*, which will be responsible for processing the message.

The bulk of the operations in a DOHA service is composed by the group of activities associated with the *Application Layer*. When a request is received on the interface, the flow passes to this layer in order to determine the type of the request. The types of request that a service can receive are related with the next activity to perform. The main activities

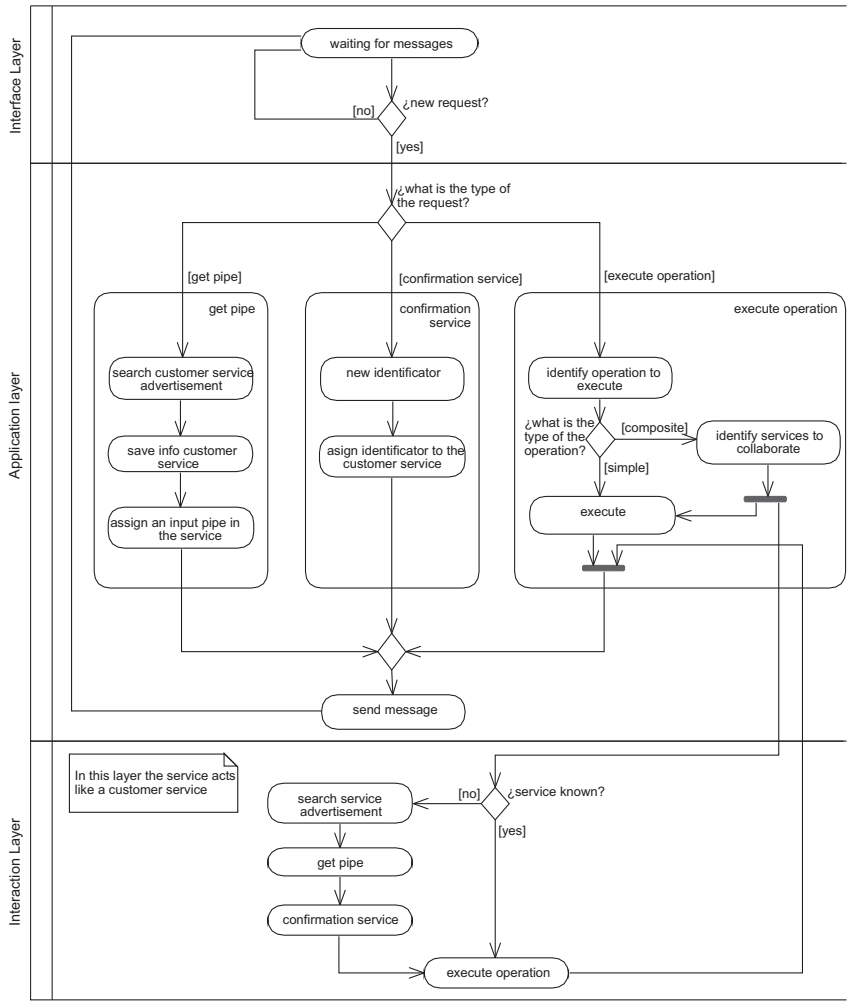

Fig. 5. Behaviour of the execution flow of a service based on their activities.

in a service are grouped as a subgroup of activities: *get pipe*, *confirmation service*, and *execute operation*. Depending on the type of the received request, the execution flow will continue for one or a subset of activities in this partition. All the groups of activities in the application partition finalize by sending a message to the service customer with information about the request. Fig. 6 shows the messages involved in the communication between a Customer Service and Provider Service by means of an UML sequence diagram.

The sub-activity *get pipe* has the function of associating an input pipe with the service customer in the service. The first activity of this group is *search a customer service*. This activity locates the customer advertisement in the network. Then, the activity of *save info customer* determines who is the service customer and what is his pipe of communication from the information contained in the customer advertisement. Finally, in the activity *assigns an input pipe* a pipe service is associated with this customer.

The *confirmation service* activity allows the customer service to know if the service is available before communicating

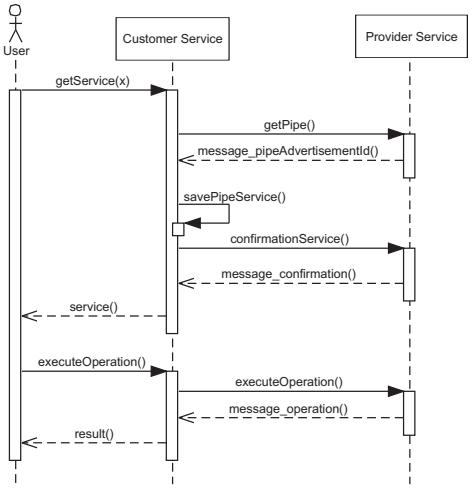

Fig. 6. Communication flow between services.

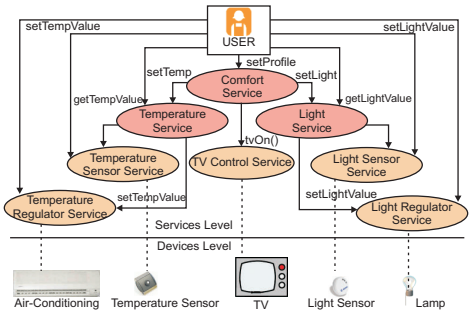

Fig. 7. Example of services composition.

with it. This activity assigns identifications to each customer and saves this information about what customer services are communicating with it.

Finally, the *execute operation* activity performs the functionality of the service executing its operations. The first activity in this group determines what operation of the service is the one that the service customer wants to execute. If the operation is simple, the service can execute it itself. Otherwise, when the operation requires the collaboration with other services, the activity of identifying services to collaborate is performed inspecting the *Service Composition Map*. Remember that the service in the execution of a composite operation can only know the required operations from other services of lower complexity degree with a depth of one level according to the service composition graph. Subsequently, the control flow must pass to the interaction partition, because the service must act as a customer of other services.

The last partition of activities is the *Interaction Layer*. The activities of this partition are fundamental in the behaviour of the DOHA services and in the composition model. This partition contains the activities that enable the service to communicate with others services acting as a consumer and to manage a composite operation. In this partition the service performs the activities as if he was a customer, *get pipe*, *confirm the service* and *execute operation*. Finally, returns the execution flow to the partition of the *Application Layer*.

VI. An Example of Application to a Collaborative Services System

To illustrate the model of service composition in DOHA platform we are going to show an example scenario where different services implement their functionality using composite operations which involve the collaboration between services. The goal of the example application is to develop a large enough number of services to cover the user needs for home-automation. The implemented services are *Comfort Service*, *Light Service*, *Light Sensor Service*, *Light Regulator Service*, *TV Control Service*, *Temperature Service*, *Temperature Sensor Service* and *Temperature Regulator Service*, as we show in Fig. 7.

From the point of view of the owner of the composite operation, the service composition process starts when it receives a request for this operation from a service consumer. After receiving the composite operation request the service searches for the services involved in the operation. When the communication between services can be carried out, the service owner of the composite operation acts as customer of the rest of the services. The users that request for a composite operation do not know that it involves the collaboration between services and the composition of their operations. The users only make the request and receive a response from the service owner of the composite operation.

Each service has its own directed graph which models its composite operations by means of $map(S)$. The set of all composition graphs in an application form the total collaborative execution flow of the application based on composite operations. By construction, we can define the system composition graph of an application as the union of individual composite operation graphs of the services of the application app, i.e. $map(app) = \cup_i map(S_i) = \cup_{i,j} map(G_{op_{j_{S_i}}}) = \cup_{i,j}(o_{op_{j_{S_i}}}, V_{j_{S_i}}, L_{j_{S_i}}, E_{j_{S_i}})$. The system composition graph can be built dynamically in order to know the *Service Composition Map* of the running services at any time in contrast with other works that require a static definition of the system composition graph [18]. This information could be relevant when a study of performance, reliability, workload, end-to-end delay or any other QoS parameter should be performed.

VII. Related Work

According to Peltz, the language used to describe the flow of collaborative processes must satisfy the requirements of i) asynchronous invocation and ii) exception handling and integrity in transaction [9]. These requirements are met on the DOHA platform, because the invocation of the operations is asynchronous and is managed by the invoker service which is responsible for handling any exceptions thrown by the specific operation.

A research work with some similarities with the presented in this paper is shown in [19], which uses the P2P technology to support the design of a services middleware. It distinguishes between the actions of peers as service providers, consumers, or both at once. The main objective of the author's research is to present a services collaboration model based on the properties of the services which are crossed semantically to obtain services with more complex features. To achieve this, the authors introduce a formal model for services composition based on the semantic properties of the services, which is similar to the collaboration model used in DOHA, that also is represented using a formal model based on the graph theory.

The high complexity of pervasive systems makes the development of applications with the non-functional properties required for the SOA paradigm difficult. A well-know strategy to overcome this complexity is the use of a centralized architecture based on a gateway [18]. Nakamura proposes a service collaboration model based on behaviour profiles created from a collaboration graph and stored in a centralized configuration file (SMI definition - Service method invocation). The collaboration between services in DOHA is also based on a file which stores services and operations related

with the composite operations of each service. However, it is private and known only to the service which is associated, other services do not know how it is or how the service works to carry out their composite operations. The main difference between the composition model designed by Nakamura et al. and the presented in this research is the scope of the composition map. In the first case, the composition model is defined at system level and the composition model can be defined previous to the definition of the services. In our case, the composition model is defined at service level and requires the knowledge of the *Service Contract* of the services with which the collaboration will be performed to form the composition map.

Ontologies are a knowledge representation model being used in a wide range of research works to manage the services collaboration, as in the case of *SOAM* of Vazquez et al. [20]. *SOAM* is an experimental model for the creation of smart objects using ontologies on the web, i.e. Semantic Web technologies to enable communication between the semantic context and reasoning processes in order to provide an adaptation of environment to user preferences. It also uses behavioural profiles, based on which it provides a service collaborative model between different semantic objects in the environment. The main short-term objective in the development of DOHA is to become a platform for context-sensitive services, thus linking the development of pervasive computing applications with the development of ambient intelligence applications using ontologies as the method for the representation of the context information and its relation with the services.

VIII. Conclusions

Development and deployment of service-based applications on embedded devices are highly complex tasks. These devices have very limited resources, little processing power and memory. Hence, the programs developed on them must optimize the use of scarce resources. For this project we choose to implement an engineering process to model a services architecture based on SOA principles and use the capabilities of the peer-to-peer JXTA platform, with a well-defined model of communication, behaviour and collaboration. Using SOA as the design philosophy can improve relations between technology and services development that support the needs of users, broadening the range of possibilities that the various applications built on the platform can offer. The ability of optimally achieving collaboration between services is a key factor for the competitiveness and growth of a services platform. The use of lightweight services composition maps based on directed acyclic graph and distributed in each service, can create new added-value services that release the potential of the applications and resources used by the platform, raising user satisfaction with them. Furthermore, the proposed composition model allows the services to define their individual composition maps, which are smaller and easy to manage and store in embedded systems than the models which define the composition flow of the whole application. A further advantage of the model presented with respect of others is its distributed nature, being a model based in individual composition maps, each service controls its own map of collaboration, without requiring a centralized node holding the information of system-wide collaboration.

Acknowledgment

This research is partially supported by the Spanish Ministry of Education and Science through a pre-doctoral FPU grant.

References

[1] P. Lalanda, L. Bellisard, and R. Balter, "Asynchronous mediation for integrating business and operational processes," *Internet Computing*, vol. 10, no. 1, pp. 56–64, 2006.

[2] A. Yachir, K. Tari, Y. Amirat, A. Chibani, and N. Badache, "Mdp and learning based approach for ubiquitous services composition," in *2010 IEEE Globecom Workshops, GC'10*, 2010, pp. 1668–1673.

[3] D. Saha and A. Mukherjee, "Pervasive computing: A paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31+4, 2003.

[4] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 256, pp. 94–104, 1991.

[5] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski, "Challenges: An application model for pervasive computing," in *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2000, pp. 266–274.

[6] Z. Stojanovic and A. Dahanayake, *Service-oriented software system engineering: challenges and practices*. Idea, 2005.

[7] S. L. Kiani, M. Riaz, Y. Zhung, S. Lee, and Y. . Lee, "A distributed middleware solution for context awareness in ubiquitous systems," in *Proceedings - 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005, pp. 451–454.

[8] J. Bronsted, K. M. Hansen, and M. Ingstrup, "A survey of service composition mechanisms in ubiquitous computing," in *Proceedings of UbiComp 2007 Workshop Innsbruck, Austria*, vol. 4717, no. 9, 2007, pp. 87–92.

[9] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[10] R. M. Pessoa, E. Silva, M. Van Sinderen, D. A. C. Quartel, and L. F. Pires, "Enterprise interoperability with soa: A survey of service composition approaches," in *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC*, 2008, pp. 238–251.

[11] J. F. Buford and H. Yu, "Peer-to-peer networking and applications: Synopsis and research directions," in *Handbook of Peer-to-Peer Networking*. Springer US, 2010, pp. 3–45.

[12] L. Barolli and F. Xhafa, "Jxta-overlay: A p2p platform for distributed, collaborative, and ubiquitous computing," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 6, pp. 2163–2172, 2011.

[13] S. Rodríguez and J. Holgado, "A home-automation platform towards ubiquitous spaces based on a decentralized p2p architecture," in *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*. Springer Berlin / Heidelberg, 2009, pp. 304–308.

[14] J. Holgado-Terriza and S. Rodríguez-Valenzuela, "Service oriented middleware for home-automation," 2011, submitted to Journal of Network and Computer Applications.

[15] J. A. Holgado-Terriza and J. Viúdez-Aivar, "A flexible java framework for embedded systems," in *ACM International Conference Proceeding Series*, 2009, pp. 21–30.

[16] L. Gong, "Jxta: A network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, 2001.

[17] R. L. McIntosh, "Open-source tools for distributed device control within a service-oriented architecture," *JALA - Journal of the Association for Laboratory Automation*, vol. 9, no. 6, pp. 404–410, 2004.

[18] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K. . Matsumoto, "Constructing home network systems and integrated services using legacy home appliances and web services," *International Journal of Web Services Research*, vol. 5, no. 1, pp. 82–98, 2008.

[19] J. Gerke, P. Reichl, and B. Stiller, "Strategies for service composition in p2p networks," in *E-business and Telecommunication Networks*. Springer Berlin Heidelberg, 2007, vol. 3, pp. 62–77.

[20] J. Vazquez, I. Sedano, and D. López de Ipiúa, "Soam: A web-powered architecture for designing and deploying pervasive semantic devices," *International Journal of Web Information Systems*, vol. 2, no. 3, pp. 212–224, 2007.