Towards an Automatic Integration of Heterogeneous Services and Devices

Jonathan Bardin, Philippe Lalanda
University of Grenoble
Laboratoire Informatique de Grenoble, Adele team
Grenoble, France
{firstname.name}@imag.fr

Clement Escoffier
akquinet AG
Bülowstraße 66, 10783 Berlin, Germany
clement.escoffier@akquinet.de

***Abstract*—The recent evolution of mobile smart devices and their convergence with pervasive computing and software as a service raises new challenges. Developers of applications targeted to these environments have to face at least three major challenges: dynamicity, heterogeneity, and distribution. In this paper, we propose a service oriented component framework which addresses these challenges by automatically reifying available services in a distributed pervasive environment. Therefore, the only thing developers have to focus on is writing the business code. We have implemented and validated our framework by using several real applications developed within collaborative projects (including ITEA ANSO project).**

***Keywords*-component, soa, pervasive, integration, framework.**

I. Introduction

As envisioned years ago [1] [2], our environment, at home or in the work place, is more and more pervaded by smart, communicating devices. These devices are not always perceivable by human beings; they have the ability for self-configuration and self-repair, and to perform context-based cognitive and physical actions. The vision of coordinated devices teaming up transparently to provide human beings with services of all sorts is already a reality.

Thanks to previous research efforts from pioneering labs and companies, an increasing numbers of such devices are being commercialized. Nevertheless, designing applications able to take advantage of this new ecosystem remains deeply complex. The designers of these innovative applications face at least three major challenges: dynamicity, heterogeneity and distribution. These challenges are amplified by the convergence of pervasive computing with cloud computing and software as a service [3].

In our previous work, we have already defined a platform-centric architecture for pervasive applications, especially related to the home environment [4]. This architecture leverages the asymmetry between light standard devices and rich service platforms where integrated composition takes place. An important aspect of our work is the development of a residential gateway providing a suitable runtime for home applications. This runtime is based on a service oriented component platform, a set of common services, and a mechanism that reifies all the available devices as services in the gateway.

In this paper, we focus on the problem of transparent interactions with available services in a distributed pervasive environment. Those services can be provided by attainable devices or third parties software running on other gateways or outside the home. The complexity of dealing with volatile and heterogeneous devices is particularly high. We propose the creation of an entity, *the remote service manager*, which automatically handles available services in the home environment. The remote service manager is in charge of providing service proxies which make the distribution transparent to the developers of pervasive applications. Underlying this, our *distribution framework* deals with the heterogeneity of devices and gateways communicating through different protocols. It also renders remote services available dynamically, faithfully reflecting the service state.

A prototype of the framework has been implemented which uses the service oriented component platform iPOJO. Several applications created through the European ANSO projects have been used to validate it.

The rest of the paper is organized as follows. First, we present the background of this work including the home computing environment; the service oriented computing paradigm and the already existing residential gateway. This is followed by a presentation of our approach and the contributions of this paper. Then a presentation of the distribution framework implementation, some examples and an evaluation are discussed. Finally, we conclude by pointing out the major propositions of this paper and ongoing work.

II. Background

In previous work, we have developed an open infrastructure for building and executing residential applications. This platform-centric infrastructure, named H-Omega, has been validated in the European ANSO project.

Figure 1 shows the proposed architecture. The H-Omega house is filled with volatile devices and a residential gateway. The purpose of such a gateway is to coordinate the devices and to ensure natural, sometimes invisible, interactions with the users. It has the ability to deal with dynamic devices and to interact with them whenever possible and desirable. Such an infrastructure permits the development of home oriented applications. Those applications, typically, provides functions related to comfort, energy saving, people assistance, and so on. For instance, devices controlled by a home gateway include shutters, heaters, air conditioning equipment, or multimedia systems.

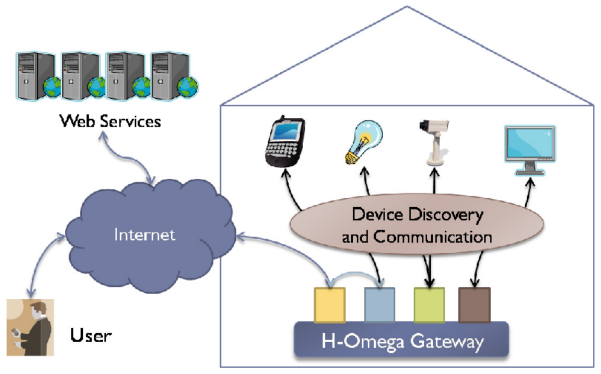

Figure 1. The ANSO service infrastructure.

This infrastructure has been designed to meet the stringent requirements of home computing, including:

- Dynamicity: due to device availability, device context, user location and activity.
- Distribution: due to the natural location of devices in the home.
- Heterogeneity: due to hardware, software and protocol variety laid by the market evolution.
- Embedded constraints: due to the restricted capability of a residential gateway.

In order to face these challenges, application flexibility and development simplicity are demanded. Service-Oriented Computing (SOC) represents today a solution of choice to deal with these issues. SOC promotes the use of well-defined composition units – services – to support the rapid development of applications. The central objective of this approach is to reduce dependencies among composition units, where a unit is typically some remote functionality accessed by clients. By reducing such dependencies, each element can evolve separately, so the application is more flexible than monolithic applications [5] [6]. Services are assembled in a service-oriented architecture (SOA) that provides mechanisms and rules to specify, publish, discover and compose available services. Depending on the target application domain, different protocols and mechanisms are used to implement SOA. For instance, UPnP (www.upnp.org) or DPWS (Devices Profile for Web Services) are preferred in small area networks. OSGi [7] is often used in centralized, embedded equipments. Web Services (www.w3c.org) are widely used to integrate IT applications.

Figure 2 shows the internal design of the H-Omega residential server, based on iPOJO. iPOJO [6] is a service oriented component runtime simplifying the development of applications on top of OSGi. The iPOJO framework merges the advantages of component and service oriented paradigms. Each component is fully encapsulated, self-sufficient and provides server and client interfaces exposing its functionalities and dependencies, respectively. iPOJO components consist of a component implementation that is managed by a reusable container. iPOJO containers provide common middleware services to the component implementations they manage (e.g. state persistence, life-cycle management). In order for a component's services to become valid, all the component's dependencies must be resolved. For this purpose, available services corresponding to a component's required (or client) interfaces must be found and connected.

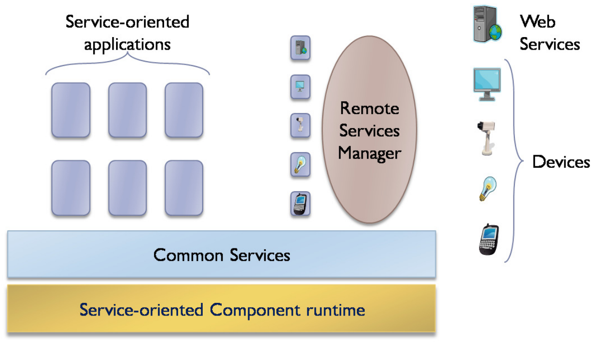
Service-oriented applications
Remote Services Manager
Web Services
Devices
Common Services
Service-oriented Component runtime

Figure 2. The home gateway architecture.

All available devices are reified as services in the framework. These particular services handle the communication between applications and real devices. Furthermore, their life cycle mimics the availability of the devices they represent. Application developers can then rely on these services to build their own application. The entity responsible of managing these particular services is called *the remote service manager* and represents the core of our proposition.

III. Requirements

As previously said, the development of context-aware pervasive applications composed of dynamic devices and services remain a quite complex task. The purpose of this work is to simplify the developer's task on the distribution aspect. More precisely, we seek to provide a distribution framework on top of the H-Omega gateway offering developers high level facilities to handle remote (and local) devices and services.

The usual solution is to rely on proxies in order interact with remote devices or services. Proxies provide the functionalities of the remote services allowing developers to only deal with local services. Somehow, they are not aware of most of distribution-related concerns. From their point of view, pervasive applications are built composing local services. This solution, appealing for its simplicity, is however hard to design and implement. It imposes in fact stringent requirements related to heterogeneity, dynamism, the proxy's providers, and the proxy's deployment.

A. Heterogeneity

During the last years some standardization effort has been made in the area of home computing. For instance, UPnP [8] defines a standard for home network communications and is now widely used in many devices such as televisions and Hi-Fi systems. X10 is used for simple electric devices (light, heater), Web-services for interaction with the Internet, Bluetooth for communication with mobile phones. Integrating these different protocols is complex and requires very advanced skills. In addition to that, it is clear that the set of protocols is not bounded and will probably grow during the pervasive system life-cycle.

As a conclusion, we believe that a distribution framework has to include several interaction protocols and a way to easily add new protocols in order to follow the evolution of the technologies.

B. Dynamism

The dynamic availability of remote services can be triggered by several events. First of all, mobile devices are by nature very volatile. For example, a smart-phone, or a PDA can freely move from a wifi coverage to another one, thus modifying its availability from the network point of view. Some network latency or break down can also cause devices and services to appear and disappear. The framework has thus to deal with the dynamic availability of these remote services and to constantly reflect this availability within internal facilities.

C. Multiple providers

The services available in a pervasive environment generally come from a wide set of different vendors. The set of services that can be accessed through a distribution framework is potentially infinite, and there is no possibility to know all of them in advance.

Vendors of such services certainly want to control what type of request can be sent to their services or devices. Thus vendors will probably provide themselves a proxy that can be deployed on a gateway. The distribution framework has to cope with that and provide solutions for the deployment of already developed proxies. Some vendors might not want to provide such proxies for their services. In this case, the framework has to provide the possibility to generate proxies on the fly based on an introspection of the functionality of the service.

To cope with both alternatives, the framework has to detect whether or not a proxy exists for a discovered service. Here again, the decision belongs to the remote service provider, who is in charge of providing all the necessary information in case of proxy deployment (where the proxy can be found, authentication information, deployment protocols and so on.)

D. Deployment

Proxy deployment is challenging since proxies will probably be held in the provider's computing infrastructure and will require authentication. The deployment of a proxy can also be very complex, as it could involve the deployment of several entities in order to work properly. For example, some proxies will be able to work only if a particular library is already deployed on the gateway. In addition each provider can use his own deployment protocol.

The distribution framework has thus to support a set of different deployment protocols. In this paper we present two of them. The first allows a simple deployment of a single deployment unit from a remote location. The second proposes to deploy one deployment unit and all its dependencies in order to behave properly. This second option introspects the gateway in order to know the dependencies that should be deployed and those that are already present. Both of them can support authentication.

E. Publication

The publication of information about available services provides the necessary means for discovery, selection, binding, and composition of services [5]. These basic operations are the taproot of SOC and are thus central in *zero configuration networking technologies* [9] such as DNS-SD (www.dns-sd.org) and UPnP.

Nowadays, these technologies are widely used and a majority of devices (printer, PDA, camera etc.) are compliant with them. Consequently, these technologies provide an effective solution for the discovery and the publication of services over the network. Therefore we believe that the framework has to include zero configuration technologies in order to easily access existing devices but also to provide a solution for the publication of information about available services.

IV. Proposition

The framework presented in this paper, conceived to run on a residential gateway, has been designed in order to provide two main functionalities to application developers. The first is to provide access to devices and remote gateways transparently. Conversely, it also aims at providing access to local services to remote gateways in a transparent way.

This framework has been built on top of the H-Omega service-oriented framework for seamless integration but also to benefit from the native dynamicity of the service paradigm. It was however necessary to extend it in order to deal with the dynamicity of remote services (not in the scope of the gateway SOC framework). Another major extension is the possibility to design pervasive applications independently of the distribution concerns (i.e. communication and discovery protocols).

More precisely, the distribution framework allows the discovery of remote services, the automatic deployment of a specific proxy with different strategies, the management of the life-cycle of the proxies and the ability to automatically generate a proxy for well-identified services. Thanks to the proxies, remote services are present on the same registry as the OSGi services. Consequently, these remote services become easily accessible in an OSGi way.

In addition, the framework is reflexive in the sense that it can create a local service from a remote service as well as create a remotely accessible service from a local service (export a service) in an autonomic way.

Both functionalities follow a three-step process. In order to access a remote service the process is the following:

1) The first step is to discover the available remote services (published by a device or a remote gateway in various protocols).
2) Then for each discovered remote service a proxy is dynamically created providing a local representation of this service in the residential gateway.
3) Finally if a remote service is no longer available, its corresponding proxy is destroyed.

Reciprocally, in order to make local services remotely available, we use the following process:

1) The first step is to detect local services which should be remotely available.
2) Then, for each one of them, a proxy is dynamically created providing a remote access to this service. At this step, we can also transparently notify the network of the availability of a new remote service using various protocols.
3) Finally if the local service is no longer available or should not be remotely available anymore, the proxy is destroyed and the network is notified (if necessary) of its departure.

We should notice that, in both cases, various strategies can be defined for the second step, determining the behavior of our framework in an autonomic way (e.g. to favor the creation of a widely used proxy).

Let us now look at the framework architecture. As previously said, the framework is directly built on top of the gateway framework. It is then made of components in a service-oriented architecture. Each component is self-governed and designed to do a specific task in the framework.

As illustrated on figure 3, the main components of the architecture are the following:

A. Remote Service Manager

The Remote Service Manager is the main component, the conductor of the framework. Its role is to orchestrate the other components in order to access or provide remote services. Its tasks will be described in the next subsection through its relation with other components.

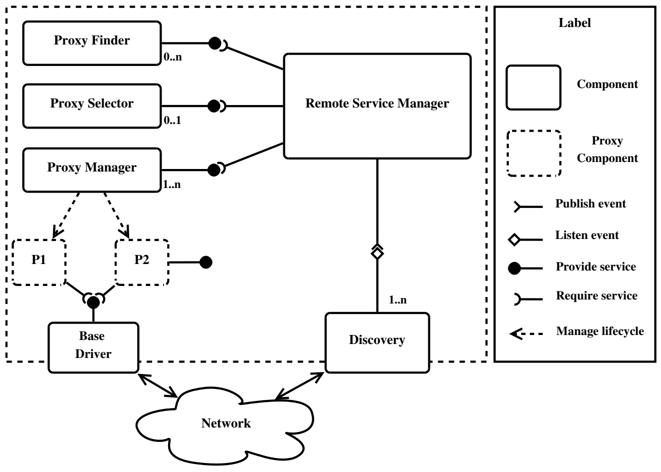

Figure 3. Distribution framework.

B. Discovery

Discovery components notify the Remote Service Manager of the arrival, modification and departure of remote services. In our approach, we recommend the use of a Discovery component for each discovery protocol.

C. Proxy Finder

After having been notified by a Discovery component, the Remote Service Manager uses Proxy Finder in order to find a proxy which matches with the discovered remote service. A Proxy Finder can find a single proxy according to the remote service properties. If several Proxy Finders are linked with our Remote Service Manager, each Proxy Finder is used in order to find a proxy. Afterwards, one of these proxies is chosen by the Remote Service Manager according to various strategies. In fact this selection is done by another component: the Proxy Selector.

D. Proxy Selector

This component is used by the Remote Service Manager in order to select one proxy among the proxies (matching with the remote service) found by Proxy finder. If there is no Proxy Selector available on the gateway, the first matching proxy is selected by the Remote Service Manager. The Proxy Selector defines the strategies of our framework. For example, if we want to save the space available on our gateway, we can use a Proxy Selector which favors the reuse of existing proxy by selecting a matching proxy which is already present in the gateway.

E. Proxy Manager

The Proxy Manager has a central role in our distribution framework. This component manages the life-cycle of the proxies. Once the Remote Service Manager has the information about which proxy must be deployed; it gives the final remote properties to a matching Proxy Manager. Then the Proxy Manager deploys the proxies and manages its life-cycle. If the remote service is no longer available the Remote Service Manager notifies the Proxy Manager to destroy the corresponding proxy. A Proxy Manager can deploy a proxy from an Internet repository and create an instance of a proxy from a proxy factory. For example, two remote services use the same proxy, one proxy factory is deployed and two instances of the proxy corresponding to the two remote services are created.

F. Base Driver

The Base Drivers are not part of our framework but play an important role in the distribution of services. A Base Driver provides the functionalities needed to communicate with a specific protocol. For example, the UPnP base driver allows for the development of Local Proxies which are local representations of an UPnP device. Indeed the base drivers are the API to communicate in a specific communication protocol. But Base Drivers can also include discovery protocols; in this case the Remote Service Manager can be notified of the availability of remote services via the Base Driver rather than a Discovery component.

G. Proxy

The proxies are the end-points of the remote services in our gateway. Indeed, a proxy provides a local view of a remote service in our middleware allowing the access to the remote services as an OSGi service and reciprocally a proxy can provide remotely a local service. We distinguish two kinds of proxies: customizable proxies and dynamic proxies.

A dynamic proxy is a proxy automatically generated from a remote service interface via reflection mechanisms. We use the dynamic proxy as a means to share services between various gateways. In this case there is no need for a syntactical alignment of the remote service because it was designed to run on a gateway.

A customizable proxy is a proxy created by a proxy factory thanks to the remote service properties. Typically, these proxies are used in order to deal with standardized devices like the UPnP devices. A proxy factory is implemented for each type of UPnP device. The framework customizes the proxy instance with device specific configurations.

In order to design home applications, developers have to know the specification (i.e. service contract, interface) of the devices which they want to control or at least listen to. This evidence makes the uses of dynamic proxies difficult. Indeed, these kinds of proxy are created thanks to reflection mechanisms and a semantic mapping. The semantic mapping takes place between two standards, the developer standard and the remote standard (e.g. WSDL,UPnP description). The problem is that most of the protocols do not define standards for devices. Consequently, the uses of dynamic proxies are quite limited.

This framework does not aim to provide an alternative for technologies such as UPnP, JINI or DNS-SD which are already preponderant in smart environments. Nevertheless, this architecture allows for the cohabitation and integration of these technologies thus providing a homogeneous and reliable framework for the design of pervasive applications.

V. Implementation

Our framework has been designed with the purpose of being used in the residential gateway H-Omega. The distribution framework has been implemented on top of OSGi and iPOJO. To date, we have implemented various distributed applications which are now part of H-Omega.

The H-Omega open-infrastructure provides various technical services necessary for the development of home applications like event management, administration-related features, persistence and scheduling management. We rely on these technical services to implement the distribution framework. Nevertheless, in order to simplify the development of distributed applications (and our framework) we had to develop new technical services related to distribution. We widely used these services as a means to implement various proxies. In this section we describe these technical services and their purpose in the framework.

A. Dealing with DNS-SD discovery

The DNS-SD is Apple's well known discovery protocol. It can be used for service discovery and registration in local area networks. We have integrated in H-Omega JmDNS, a Java implementation of DNS-SD fully compatible with Apple's Rendezvous. Thanks to JmDNS we have implemented a Discovery component which allows for the discovery of devices and services via DNS/SD, which means that no network configuration in a local area network is needed. Both JmDNS and UPnP provide to H-Omega, via the distribution framework, a Zero Configuration Networking solution.

B. Dealing with UPnP

UPnP is the second ZeroConf solution which has been integrated in our framework. In order to deal with UPnP devices, we have adopted the OSGi UPnP base driver that has been implemented by Apache Felix. We use the UPnP base driver as a means of being notified of the arrival and departure of UPnP devices in a local area network but also to handle the communication between the UPnP devices and their local proxies.

Actually, the *Remote Service Manager* listens to the services published by the UPnP base driver. We have also implemented a UPnP Proxy Finder. This Proxy Finder is similar to a registry which contains information (location, interface and other properties) about UPnP proxy factories available from the gateway. Currently, we have implemented three proxy factories, each one corresponding to a UPnP device type:

- The first allows for the creation of proxies providing an OSGi service for the UPnP Light devices.
- The second allows for the creation of proxies providing an OSGi service for the UPnP Media Render.
- Finally the third allows for the creation of proxies providing an OSGi service for the UPnP Media Server.

The component in charge of deploying these proxy factories and creating an instance for each matching device is the Proxy Manager. As these proxy factories are developed with iPOJO, we have implemented a Proxy Manager which manages all iPOJO proxy factories.

C. Dealing with Web-Services

Nowadays Web-Services are widely used as a foundation for machine-to-machine interaction over the network. Furthermore, Web-services enable systematic application-to-application interactions on the Web and a smooth integration of the existing Service-Oriented architecture [10]. For all these reasons we have implemented a Service-to-Web-service proxy generator. In our framework, this proxy generator is managed by a Proxy Manager which creates a Web-service for each service which should be remotely available. This is a central part of the H-Omega-to-H-Omega interactions.

Currently, the implementation is based on Xfire but we are working on a new version where Xfire will be replaced by its successor, the Apache CXF framework (cxf.apache.org).

D. Dealing with Distributed Events

At this point we have characterized various implementations about discovery and remote communication. Now we will describe how our framework deals with Distributed Events. In OSGi, events play a central role since they authorize asynchronous communications. The OSGi Event Admin service provides a generic publish/subscribe event mechanism. Furthermore, the Java Message Service API supports the publish/subscribe model as a means of sending messages between two or more clients. Therefore, we have naturally thought of implementing a bridge between the OSGi Event Admin and JMS.

Our bridge provides a simple mapping mechanism which allows to automatically publish an OSGi event as a JMS message and reciprocally.

VI. Examples

In the previous section, we described various distribution-related services that have already been implemented. Thanks to these services, we have designed and implemented some applications validating our proposition. As the purpose of our framework is to hide distribution aspects from the developers, we have fully implemented the following examples in iPOJO / OSGi without any specific distribution code.

A. The H-Omega UPnP Media Application

This application allows a user to interact with one or more UPnP MediaRenderers and UPnPMediaServers that are available on a local network. A MediaRenderer is a device capable of rendering Audio Video (AV) content from the home network. Examples of MediaRenderer devices include traditional devices such as TVs and Stereo systems. A MediaServer is a device capable of providing AV content to other devices on the network. Traditional devices such as VCRs, CD players, DVD players, TV tuners, etc. can be MediaServers. Thanks to our framework, any UPnP Media device present over the network is available on H-omega through a proxy. The media application is a component which interacts with the UPnP media proxies in order to provide a user interface. With this interface, a user can control media devices. A hypothetical scenario is the following:

“Robert is in his house. Yesterday, he took some shots with his brand new UPnP compliant mobile phone. He wants to see his photos on his DLNA TV (www.dlna.org). In order to do that, Robert logs into H-Omega via its web interface. Thanks to the media application he displays on his TV the pictures available on his mobile phone.”

B. The H-Omega Follow Media Application

This is a context-aware application which controls the Media player (e.g. TVs, Hi-fi) present in the house. Like in the previous example, this application interacts with the UPnP Media server and renderer proxies so as to provide a smart Media player. The goal of this ‘smart’ media player is to follow the user while he moves from one room to the other. Thereby, the user can continue to watch and/or listen to a media file even if he moves around the house. The following scenario illustrates the Follow Media application:

“Robert is watching a DVD in this kitchen while he is doing the dishes. All the dishes are clean, so Robert decides to go to the living-room and watch the end of the DVD. Fortunately, Robert has previously turned the H-Omega Follow Media application on. This means that the movie will stop the moment he goes out of the kitchen and will automatically restart where it had left off on the living room screen once he is there. So he will not have to take the DVD from the kitchen to the living room and select the appropriate scene.”

C. The H-Omega Devices Simulation

In order to test the H-Omega infrastructure we have implemented several simulated devices (window shutters, dimmer lights, heaters, refrigerators, etc.). We have implemented these devices as POJOs on the top of OSGi using iPOJO. Thanks to our distribution framework, these simulated devices are remotely available over the network. Each device is accessible via a WS and their events via JMS while the service discovery is done by DNS-SD. The following is a hypothetical simulation flow:

"First, we start the gateway S which contains the simulated devices. Then, we create the simulated-devices instances. If H-Omega is not running we start the gateway H which contains H-Omega. Once the simulated-devices are started, DNS-SD messages are broadcasted in order to notify the network of their availability. The distribution framework present in H-Omega handles the DNS-SD messages and instantiates a proxy for each device. Finally, the applications requiring these devices are now resolved and usable."

D. The H-Omega home automation apps

The H-Omega open infrastructure provides various home automation apps such as 'The Wake Up application' and 'The I leave/I come back application'. Both of these applications interact with the home devices in order to provide an end-user service. Thanks to our framework, these applications only have to deal with the OSGi services provided by the proxies of the devices. This means that these applications do not have to discover the devices over the network or use the right protocols in order to interact with them. So, the use of our framework and iPOJO has allowed us to develop these applications as POJOs.

VII. Evaluation

Our framework performances are intrinsically bound to the protocol used for the discovery and communication. Indeed, from an application point of view, the communication latency between a local and a remote service corresponds to the cost of distributing an application. Thus evaluating the performance of our framework from an application perspective will mainly reflect the cost of the standard protocols used in the evaluation.

In this section, we aim at evaluating the overhead of our framework independently from the protocol. That is why, rather than comparing the protocols performance we evaluated the cost introduced by our framework to transparently distribute an application. This specific overhead has been evaluated by comparing the instantiation of a component providing a local service and the same component providing a remote service. The figure 4 shows the result of a benchmark run on a laptop (2Go of RAM and an intel core 2 duo at 1.6GHz).

The evaluation principle consists in creating several instances of the same service with different properties. The

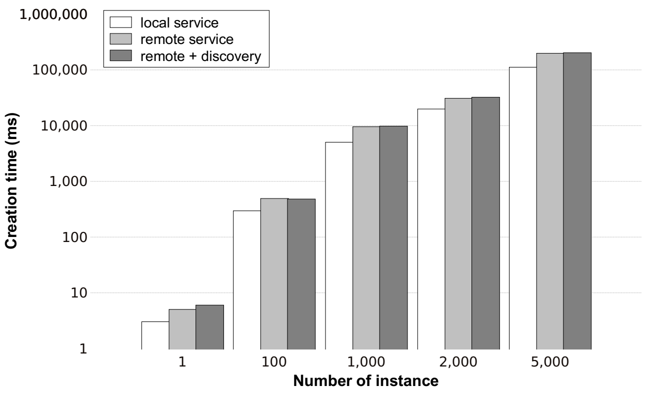

Figure 4. Framework overhead.

x-axis of this graph shows the number of instances created during the experiment. The y-axis represents the time in milliseconds needed in order to create that number of instances. The white, gray and black columns correspond to the instances providing a local service, remote service and remote service plus advertisement respectively.

This evaluation shows that providing remotely accessible services only takes a linear time in addition to providing it locally. Adding the proactive discovery mechanism to this creation does not introduce a significant overhead. This last observation is mainly due to the way we have implemented the framework. Indeed, the advertisement of a remote service is done trough a 'white board' pattern, which basically reduces the cost of this advertisement to the cost of providing a local service on the gateway.

We believe the overhead introduced by our framework is more than counter balanced by the benefits it offers to the developers of highly dynamic and distributed applications. Thus, we consider it to be a natural way to implement pervasive applications.

VIII. Related Work

Several protocols and standards have been proposed to support seamless interactions between communicating devices. The widely used UPnP technology provides a distributed, open networking architecture, which leverages TCP/IP and Web technologies to enable seamless proximity networking in addition to control and data transfer between networked devices [8]. The PSMP protocol extends UPnP in order to increase the reliability and robustness of smart home systems [11]. Unfortunately, these technologies don't interact with other standards and don't provide a framework for the development of pervasive applications. We believed nevertheless, that they lay down the foundation for a smart pervasive computing network.

R-OSGi provides a distributed middleware platform that extends the centralized OSGi specification to support distributed module management [12]. Some effort has been made in order to use R-OSGi as a solution for smart home systems [13]. The R-OSGi approach is indubitably attractive since it allows for transparent distribution. However, R-OSGi supports only *R-OSGi-to-R-OSGi* communication which is unfortunately a strong limitation for its integration in a pervasive environment.

In [14], a reflective middleware, ReMMoC, enables a mobile client to be developed independently from both discovery and communication protocols. In [15], the MUSIC planning-based middleware combines SOA and component-based software development, and creates service plans using service descriptions and agreements. Finally, in [16], The MySim middleware uses events to transparently and spontaneously adapt service-based applications to the pervasive environment. Essentially, these approaches abstract discovery and interaction protocols in pervasive environments by computing strong syntactic or semantic equivalence between heterogeneous service descriptions. In contrast, our framework is based on a service-oriented component runtime which allows the developers of pervasive applications to easily add essential mechanisms such as logging and state persistence.

IX. Conclusion

Developing correct, maintainable pervasive services is a real challenge nowadays. It is clear that most of the techniques currently available are not mature, are hard to master, and consequently, are challenging for the major players of the market.

In this paper, we have presented recent developments in the area of service oriented home gateways with a focus on distribution management in home networks. We believe that two important aspect of pervasive service computing have to be improved: the development environments and the runtime environments.

Our solution, based on automatically updated proxies, allows developers to avoid most of the complexities of the distribution. The proposed framework handles the dynamic and heterogeneous devices and services available in the network. As future work, we plan to complement our framework with a specific development environment. In particular, techniques are needed to make developers aware of distribution constraints and to avoid coding technical aspects as explained in this paper.

References

[1] M. Weiser, "The computer for the twenty-first century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] M. Satyanarayanan and Others, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[3] V. Pendyala and S. Shim, "The web as the ubiquitous computer," *Computer*, vol. 42, no. 9, pp. 90–92, 2009.

[4] C. Escoffier, J. Bourcier, P. Lalanda, and J. Yu, "Towards a home application server," *2008 5th IEEE Consumer Communications and Networking Conference*, pp. 321–325, 2008.

[5] M. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," vol. 3, 2003.

[6] C. Escoffier, R. S. Hall, and P. Lalanda, "ipojo: an extensible service-oriented component framework," *Services Computing, IEEE International Conference on*, vol. 0, pp. 474–481, 2007.

[7] A. OSGi, "Osgi service platform core specification release 4," may 2007. [Online]. Available: http://www.osgi.org/Specifications/HomePage

[8] F. UPnP, "Upnp device architecture 1.1," october 2008. [Online]. Available: http://www.upnp.org/resources/specifications.asp

[9] E. Guttman and S. Microsyst, "Autoconfiguration for IP networking: Enabling local communication," *IEEE Internet Computing*, vol. 5, no. 3, pp. 81–86, 2001.

[10] F. Curbera, W. A. Nagy, and S. Weerawarana, "Web services: Why and how," 2001.

[11] C. Liao, Y. Jong, and L. Fu, "Psmp: A fast self-healing and self-organizing pervasive service management protocol for smart home environments," pp. 574–579, 2008.

[12] J. Rellermeyer, G. Alonso, and T. Roscoe, "R-osgi: Distributed applications through software modularization," *Lecture Notes in Computer Science*, vol. 4834, pp. 1–20, 2007.

[13] J. Wu, L. Huang, D. Wang, and F. Shen, "R-osgi-based architecture of distributed smart home system," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1166–1172, 2008.

[14] P. Grace, G. S. Blair, and S. Samuel, "A reflective framework for discovery and interaction in heterogeneous mobile environments," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 1, pp. 2–14, January 2005.

[15] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments," pp. 164–182, 2009.

[16] N. Ibrahim, F. Le Mouël, and S. Frénot, "Mysim: a spontaneous service integration middleware for pervasive environments," New York, NY, USA, pp. 1–10, 2009.