Home SOA – Facing Protocol Heterogeneity in pervasive Applications

André Bottaro[1, 2], Anne Gérodolle[1]

[1] *France Telecom – Orange Labs R&D*
28, chemin du vieux chêne,
38240 Meylan. FRANCE
{firstname.lastname}@orange-ftgroup.com

[2] *Grenoble University*
Laboratoire LIG-Adele
38041 Grenoble, Cedex 9, France
{firstname.lastname}@imag.fr

Abstract

Numerous innovative applications are envisioned in the home area network today, mixing current or nearly marketable applications: multimedia content sharing, home comfort, home surveillance, people care. The home network becomes a ground for pervasive computing design. Its openness to dynamic distributed and heterogeneous devices emphasizes the pervasive challenges in home application design.

We present an open architecture for home service development and compare it to related work. Distribution and protocol heterogeneity are managed by service-oriented drivers leveraging the "service platform" concept while network dynamicity is locally reified on the platform. Through some implemented use cases, we show how technological choices and service-oriented patterns facilitate the dynamic composition of distributed and heterogeneous home device services.

Keywords: SOA, pervasive computing, home network, OSGi, plug-n-play protocols, field buses.

1. Introduction

Electronic devices are increasingly spreading in all the application domains of the home area, e.g., entertainment, information, communication, office, comfort, security, cooking, cleaning. The actual new trends make all these devices progressively become networked together. The home network will soon be ready to realize the vision that Mark Weiser described years ago [1]: our environment is more and more filled with networked devices that are slowly fading into the background. Ambient lights are reacting to the user activity while some devices like the set-top-box, the fridge are connected to the Internet in order to facilitate and enrich our daily activities.

However, while the "home network" becomes a reality, only few and isolated device islands show some real networked collaboration. The best example is probably given by home multimedia devices whose success is accompanied by UPnP/DLNA standardization (see www.upnp.org, www.dlna.org): Devices playing the role of Media Servers, e.g., network-attached storage (NAS), mobile, PC, are put in relation with devices playing the role of Media Renderers, e.g., television, media center, set-top-box (STB), hi-fi system, by devices playing the role of controllers, e.g., media center, mobile, PDA, PC, home gateway. Other examples are given in the home automation area where a centralized system is connected to specific sensors and is controlling lights, blinds and doors through a specific field bus.

Moreover, the examples show not only that application domains still remain separated from each other but also that numerous protocol sets are competing even in each application domain. For instance, the multimedia domain is targeted by UPnP/DLNA, Apple (e.g., Bonjour, iTunes) and IGRS (www.igrs.org) standards. Home automation is also targeted by open organizations or strong brands, e.g., European KNX (the convergence of EHS, BatiBUS, EIB), Asian Echonet, ZigBee Alliance, Lon, X10.

Thus, before envisioned applications mixing various protocol sets and various domain applications arise, challenges remain. Designing an application composing the adapted device features found into an open environment according to the environment context still remains a complex task. In research labs worldwide, we think of a media player that is put in relation with media sources obeying various protocol sets and that lowers the sound volume when the phone is picked up in the lounge, of a light that is dimmed when a movie is played on the main TV screen, etc.

In this paper, we expose the home context and propose an architectural answer to its challenges. This answer relies on the *service platform* concept at the heart of our Home SOA solution. Then we describe how we applied our model opportunistically in three

examples mixing multimedia, home automation and device management domains.

The proposed environment allows designers to model networked applications as a local composition of uniform service interfaces lately bound at runtime. It deals with three of the main challenges of Pervasive Computing: dynamicity, distribution and heterogeneity. The seamless integration of local and distributed services using various protocols is one of the issues addressed by the Amigo[1] European project. This framework leverages the assets of the OSGi platform standard [2]. Part of this work is already published through the OSGi standardisation process [3] [4] [5].

The paper is organized as follows. The next section introduces home use cases and the technical requirements related to the design of home applications. Then we expose our Home SOA principles and implementation above the OSGi technology in part 3. We categorize the flexibility advantages of our solution through architectural considerations in part 4. Part 5 presents opportunistic design and implementation of introduced use cases. Performance tests on our service-oriented layer are described and analyzed in part 6. We confront this work against the state of the art in part 7. Finally we conclude by pointing out major contributions.

2. Home application design requirements

2.1. Home control protocols

2.1.1. IP Plug-n-play protocols

The IP layer is relevant and affordable when applications imply heavy use of physical bandwidth, for instance playing digital resources. The multimedia market is divided between 3 main standards: UPnP, Apple Bonjour and IGRS protocols. In the printing domain (e.g., printers, scanners), we find the previous ones as well as the new DPWS protocols pushed by Microsoft (Devices Profile for Web Services). In the IP connectivity (e.g., router, Wifi access point), UPnP took the lead. And before these standards have won market shares, some other technological attempts existed, e.g., Jini, SLP, Salutation.

While these protocol sets share common points, they also show important differences that make them non-interoperable. First, they all are service-oriented and show the same generic layers: IP addressing, discovery, description, control, eventing. However, each protocol set has got differentiated assets adapted to the targeted environment [6].

2.1.2. Field buses

When applications require little bandwidth and when stress is emphasized on price and power consumption level, field buses become relevant. They are mainly used in Home Automation domain. Like the IP plug-n-play protocols, they show similarities and also strong differences between each other. In the case of field buses, even the physical OSI layer is a differentiation criterion. ZigBee is optimized for very low power consumption rates and offers a discovery layer, KNX (EHS, EIB, Batibus) is known for its simplicity in Building Automation, X10 targeted the *Do It Yourself* market in the 70's. Many other protocol sets exist, e.g., LON, CEBus, Echonet, Zwave.

2.2. Use cases

The following list of use cases is enumerated in order to show distinct situations in which distribution and heterogeneity technical challenges are emphasized. In the use cases, Maxandre is a young man who installed the last networked technologies in his home. An implementation of these use cases with Home SOA principles is shown in part 5.

2.2.1. A reactive home theatre system

Maxandre installed a UPnP TV set that enables him to search multimedia files on his PC from the TV set in the living-room. He has got also a device named "Home SOA box" that listens to UPnP events in order to act on comfort devices like his X10 lights and his ZigBee blinds in an adapted manner. When a movie is started on his TV, the Home SOA box dims the lights and shuts the blinds in the living-room.

Many other use cases that are studied in the Home market involve distinct standard and proprietary protocols and mix distinct application domains (see part 1), here the home automation with the multimedia activity. This use case shows the requirement of a smart device in the home to support heterogeneous protocols and compose them in integrated services to the user.

2.2.2. Service sharing between smart platforms

The Home SOA box of Maxandre is a device that has got wired Ethernet and PLC (Power Line Communication) links. However, the box is not able to listen to wireless protocols like the ZigBee protocol set. So the company selling this device sells also other

[1] This work is partially supported by the European Commission under IST Amigo Project.

radio devices that are complementary. Maxandre then buys a technological bridge between ZigBee devices and the protocols understood by the box.

Some add-ons are necessary for the Home SOA box to handle the new services provided by the complementary device. Therefore, when plugging the device, Maxandre is asked to accept that some application update be done on the Home SOA box.

The use case illustrates the typical home automation situations where services are accessible through a specific gateway. The gateway is connected with a specific physical layer to devices constituting a capillary network. It enables other home services to use home automation services.

Physical layer heterogeneity and physical resource sharing are not the only issues that lead to the need for a protocol bridge between service platforms. We can mention the case of two platforms with distinct programming technologies that are meant to share services. In the Amigo project for instance, some services were developed on a .NET platform while others were developed on an OSGi platform.

2.2.3. Home device diagnostic and testing

Maxandre notices that one of the lights in the living-room is not dimmed when a movie starts. He opens the web pages provided by the Home SOA box to test and configure his home devices. He tests the lighting devices in the living-room one by one and configures the location of the ZigBee light. The location of the device was not known by the system. Now the home theatre system dims the right lights.

Thanks to device discovery protocols, device description, configuration and functions are accessible by smart devices like the Home SOA box. The latter gives the means to diagnose, test and configure the devices through simple web pages to the user.

This self-care scenario shows the need for a protocol-independent diagnostic tool for home users. The need for control protocols in network diagnosis and testing is given in several papers, e.g., [7].

2.3. Home application requirements

Use cases envisioned by telecommunication operators, service providers, manufacturers and software vendors in the smart home environment introduce the hard challenge due to the openness of the home environment. They demand the collaboration of heterogeneous devices and protocols in a mix of various application domains (see Figure 1).

Flexibility is the main requirement in order to face the openness of the Home environment. The targeted flexibility must enable the architecture to rapidly and smoothly accept new protocols on-demand and keep a uniform high-level API to compose the home pervasive entities. Precise technical requirements deriving from this vision have to be met:

- **Separation of concerns.** The application logic has to be separated from distribution and protocol details. Handling distribution and protocol heterogeneity must be transparent to application developers.
- **Uniform interfaces.** The protocol heterogeneity must be masked to the developer in order for them to use the right device features without a deep knowledge of the device network interfaces of every technology.
- **Code-On-Demand.** Handling a new protocol or a new device type must ideally not stop running applications.
- **Reactive protocol discovery.** The entrance and the departure of a device must be notified on the application side. A device bringing required features that are accessible through a new protocol triggers the process to handle this new protocol.

Distribution and heterogeneity are not the only challenges in pervasive environments like the Home Network. Dynamicity [8], security [6] and embedded constraints [9] imply also critical requirements in pervasive applications. Dynamicity is brought by device availability, device context, user location and activity. Security issues are raised by the openness of such environments to any kind of attacks and the privacy level demanded by the user. Embedded constraints are due to device low cost drawn by the consumer electronics market in Home applications.

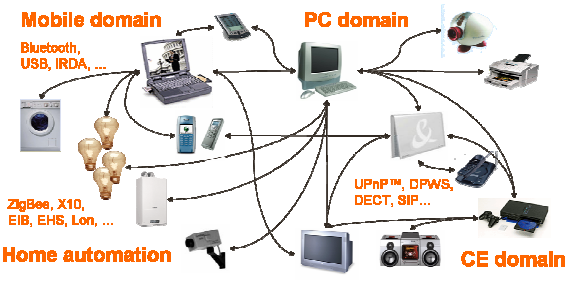

Figure 1 Protocol and application domain variety

Our contribution is the design of our service-oriented device control framework above the OSGi technology (see next part). Coupling service-oriented

design patterns with a platform technology, which brings software modularity and code-on-demand features, achieves to satisfy the flexibility needs. The flexibility or our solution is discussed against architectural considerations and is demonstrated through the description of the implementation of the introduced use cases in following parts.

3. Home SOA principles and implementation

3.1. The Service Platform concept

Service orientation models the networked entities into logical units that provide and require operation sets, which are called services. Figure 3 depicts the well-known service orientation paradigm. Service providers and service requesters only share the knowledge of a service abstraction. Service requesters actively request or passively listen to the registration of service providers in a service registry, i.e., entities that implement a known interface name and optionally satisfy specific properties. When a service reference meets service requirements, the service requester may invoke the provided service.

Service abstraction and loose coupling are needed in home application design so as to compose the various entities into user applications. Home networked devices are the pervasive services to be composed in targeted use cases. Our Home SOA solution dynamically reifies these heterogeneous entities into programming language proxies in a local service registry. The registry is coupled to a notification system that mirrors the network events into programming language events on the local platform (see Figure 2).

Thus, the service composition in our Home SOA solution is a software collaboration model between objects developed on a local programming language runtime. Here are the main concepts of this service collaboration model:

- A service interface is a programming language interface under the name of which an object is registered.
- A service provider is an object registered in the service registry. Registered references are available to any object on the platform. However, scopes and access rights may optionally restrain service visibility and accessibility for structural and security purposes.
- A service client is an object that gets a reference on a service provider thanks to an active request or a received event and that invokes methods of the referenced service provider.
- An adapted service event is fired by the service registry whenever a service reference is registered, unregistered or modified.

The service platform concept is characterized by this service-oriented design. The registry of services is the masterpiece of the collaboration model. This service-oriented design relies on the design pattern named Service Locator [10]: It simplifies the injection at runtime of implementations unknown at development time into an application composing abstract entities. The latter abstraction uses the *interface* pattern well-known in object-oriented programming.

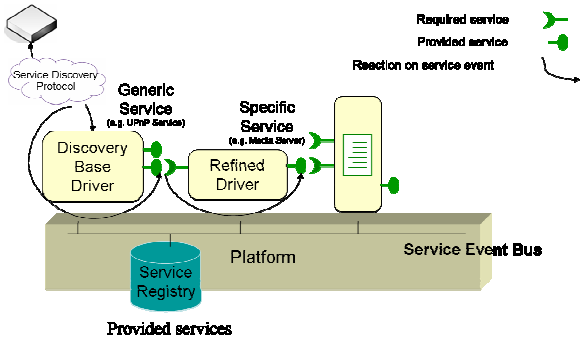

Figure 2 Dynamic mediation on a service platform

3.2. Service-oriented drivers

A first range of drivers, named *Base Drivers* – a naming convention taken from the OSGi Device Access specification [2] –, is responsible for reifying devices on the service platform (see Figure 2). These drivers hide distributed aspects by registering proxy objects in the service registry. Specific details of the protocol discovery, control and eventing layer are invisible to the application developer. The first flexibility requirement is met thanks to this service-oriented driver layer – separation of concerns requirement of section 2.3. Base drivers relay local invocation calls on device proxies into remote operation calls on devices.

Each base driver is specific to a protocol set. The device description structure is represented in object-oriented programming interfaces adapted to the protocol set. Therefore, a service interface corresponds to each protocol set. The only common entity shared by all the drivers on the platform is the service registry. It enables the support of an extensible list of protocol sets without dependency between each other.

The device reification on the platform is dynamic. Base drivers are responsible for registering, unregistering and modifying registered proxies whenever the represented device joins the network,

quits the network or is updated. Device availability is mirrored by service availability on the platform.

3.3. Service-oriented mediators

A second set of drivers, called *refined drivers*, is responsible for masking protocol heterogeneity, i.e., the second flexibility requirement (see section 2.3). Each refined driver is mediating services of a specific device type into services following an application domain semantic (see Figure 2).

Refined drivers dynamically react to the device service availability in the service registry. Whenever watched device services are registered, unregistered or modified, refined drivers respectively register, unregister and modify matching application domain specific services.

Finally, thanks to service-oriented drivers, distributed aspects and protocol heterogeneity vanish while network dynamicity is mirrored on the platform. The technical mediation performed by both types of drivers dynamically reifies home pervasive services into local uniform protocol-agnostic services. Applications are notified of the arrival, removal and modification of required services by the platform service notification.

3.4. An OSGi implementation

The service platform concept is implemented by the OSGi standard [2][11]. The proposed Home SOA architecture is thus implemented in a straightforward manner above this technology. The OSGi specification also defines the way to deal with devices in the Device Access Specification. However, the technical mediation chain, the design of our drivers [9] and moreover the opportunistic implementation of mentioned use cases (see part 5) is original in our work.

Apart from delivering service-oriented features, the OSGi core platform defines an original deployment unit model that specifies the way to share and isolate resources between modules called bundles. The standardized bundle lifecycle coupled with the Java dynamic class loading enables module update at runtime, which answers one of the third flexibility requirements – the Code-On-Demand feature mentioned in section 2.3.

The service-oriented pattern is well implemented when service interfaces (Java API), the service requester (Java client) and the service provider (Java implementation) are contained into separate bundles and when requester and provider cooperate through the service registry (Figure 3). This recommendation is followed by our design. Base and refined drivers are bundles that provide respectively device services and mediated services, which service interfaces are defined in separate bundles (see the use case implementation in part 5). This separation, the dynamic OSGi bundle deployment features and the technical mediation possibilities make our solution meet the last flexibility requirement, i.e., the reactive protocol discovery requirement defined in section 2.3.

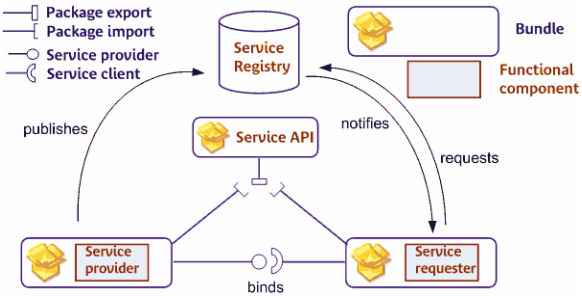

Figure 3 The service orientation in OSGi design

4. Home SOA flexibility

Middleware defines network or software layers that enable several programs to work together. We explore the trade-offs of three architectural aspects in this section. Each of them represents a precise issue between two distinct middleware design approaches.

Every design model is relevant in some situations and is wrong in others. The flexibility needed by the home platform developers addresses all the situations. The described Home SOA solution is an attempt to deal with all of them.

4.1. Network or language interfaces

Middleware can either mean protocol or programming middleware. A protocol middleware provides network protocol sets, e.g., the Web Services. The other type provides a set of interfaces at the programming language level, e.g., the OSGi standard.

The second use case shows the need for service sharing between smart home platforms. In that case, a unique network protocol has to be chosen for the communication between those platforms.

While the distribution of the intelligence in the environment is bringing application flexibility in some cases, a drawback is lower computation efficiency. A language interface pivot is better for computation efficiency between software entities.

The Home SOA solution proposes an answer to both situations. However, the advocated software modularity and the support of heterogeneous protocols rather bets

on the long-term extensibility of a general device than on the network interoperability of specific devices developed during short cycles.

4.2. Interface granularity

Loose coupling of software entities is obtained by minimizing the knowledge shared by software entities. In the Home, interfaces representing devices may be coarse-grained of fine-grained. The coarse-grained approach is natural in device control domain: The service grain matches the device consistent set of control operations.

While the coarse-grained approach is advocated for composition consistency, application adaptability is increased if the composition is made of smaller grains. The operation with its signature is the basic element of semantic composition approaches [12]. However, a huge work on domain-specific ontology is demanded by these approaches in order to satisfy home use cases. The need is unfortunately not fully answered by the described work. The consistency of syntactic interfaces is kept as a general basis for our project today.

4.3. Interface generality

The generality level of a middleware layer is mainly a trade-off between adaptability and efficiency. The Web Services are for instance a technology that is adaptable in every client-server application but that requires a deep knowledge and important work to be applied in specific domains. This is a lesson learned by many authors, e.g., [13], who apply Web Services in device control applications. Interface generality is also related to memory overheads (see Figure 8).

Our search for network and language pivots that separate the application logic from technology details is opportunistic. A defined scope of use cases leads to a specific set of requirements and the right interface design choice. Our Home SOA framework uses mediation techniques to refine device interfaces into opportunistic protocol-agnostic representations in an object-oriented programming language.

5. Opportunistic answers to the use cases

5.1. The reactive home theater system

Base drivers and refined drivers are used here to map multimedia devices into media source and sink representations (Figure 4). UPnP and Bonjour base driver dynamically populate the OSGi service registry with instances with respective *UPnPDevice* and *BonjourDevice* interfaces hiding protocol and distribution details while keeping protocol device description specificity. *MediaSource* and *MediaSink* refined drivers dynamically react to the registration of instances implementing previous interfaces to register service interfaces with multimedia semantics. This refinement hides the remaining device description specificity to embrace multimedia domain semantics.

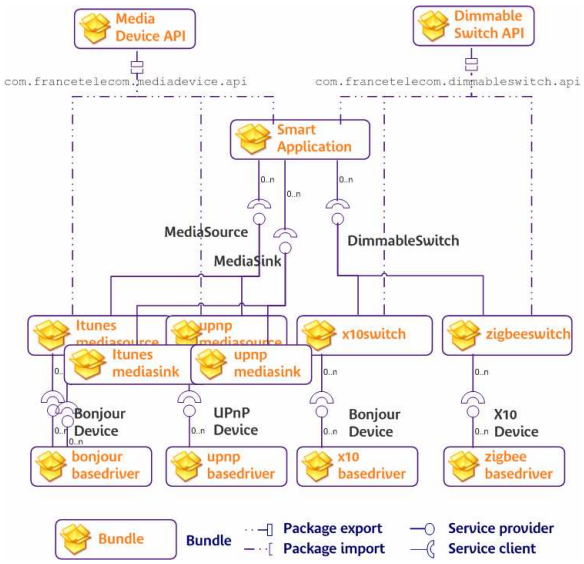

Figure 4 Media and Automation language pivots

Home automation devices are treated in a similar way. While X10 and ZibBee base drivers mirror the dynamic presence of associated lights and blinds, refined drivers reactively populate the service registry with associated service instances implementing the *DimmableSwitch* interface matching electric variation semantics, which are common to lights and blinds.

The media application composes the previous media services available in the service registry. It proposes to the user the media content that is stored on media sources, proposes to send them on media sinks like the TV in the living-room. The movie is started by the application on this platform or by the UPnP TV itself. When a movie is started, a standard event comes from the related *MediaSink* service. The application reacts by dimming *DimmableSwitch* services that are located in the living-room. Device location is a common service property that is expected on media services.

5.2. Service sharing between smart platforms

In the capillary network case, the chosen protocol has to remain as simple as the home automation control protocols are. UPnP with its ad hoc device description layer supporting only simple data types is well-designed to match the simplicity of control and event

layers of these protocols. Teaha European project made this choice for instance [14].

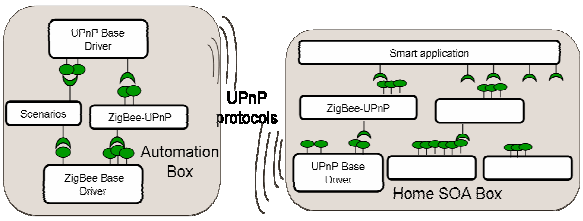

Figure 5 Smart devices sharing services

The Amigo project has chosen DPWS for service sharing between heterogeneous platform technologies: OSGi and .NET platforms. The choice of DPWS is based on the need for generality. Indeed, the protocol has to map the powerful expressivity of interfaces in programming languages such as Java and C#.

SLP discovery protocol coupled with binary communication like RMI has been found well-suited for service sharing between OSGi platforms in other projects. (See related work [15][16] in part 7).

In these applications, on the platform client side, base drivers populate the service registry with service proxies that represent the remote services advertised by remote platforms. Static or dynamic code generation is optionally provided by tools (capillary network and Amigo case) or runtimes [15] that generate proxies on the client side. Proxies are implementing service interfaces that are already deployed on the client side. The capillary network case is illustrated in Figure 5.

On the platform server side, services have to follow some rules in order to be advertised on the network by the base driver and be accessible from other peers. These rules are the implementation of a specific interface (capillary network and Amigo case) or only a configuration parameter that indicates that some functions are to be exported on the network. In the latter case, stub generation transparency is performed thanks to Java reflection and byte code engineering. Java reflection extracts the needed information from the service provider classes. Bytecode engineering exposes communication skeletons on the local platform [15]. Then the service registration, modification, unregistration in the OSGi service registry is mirrored by the base driver on the network through the chosen service discovery protocol.

5.3. Home device diagnostic and testing

Protocol independency of the self-care application relies on a pivot into which any device representation is transformed. The third use case is thus treated similarly to the first one. The DPWS semantics linked to the JWSDL standard [17] – Java representation of WSDL – are chosen as the opportunistic language pivot in order to map any device description protocol. This choice of the Web Services semantics was driven by the need for generality to support an extensible list of protocols. The existence of DPWS achieved to motivate the decision.

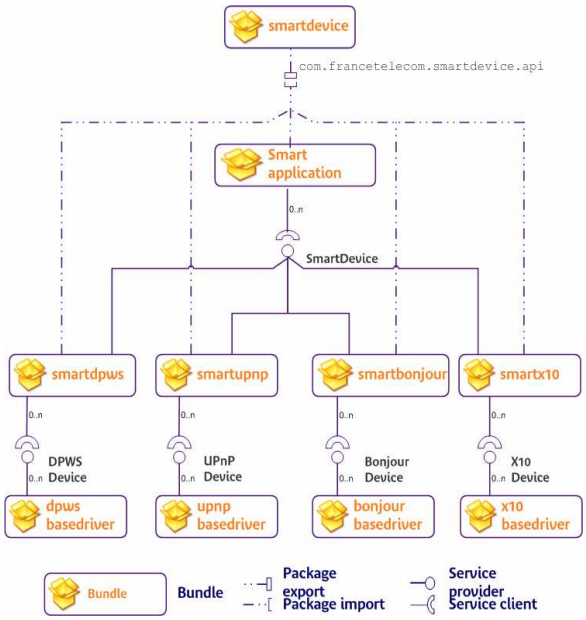

Figure 6 Smart device API – a language pivot

This language interface pivot for device control protocols is named Smart Device API in Figure 6. Refined drivers – the second range of drivers depicted on the figure – transform every registered device entity in the registry into a *SmartDevice* service. The Self-care application – named Smart Application in the figure – is then able to dynamically treat any available device independently of its original technology thanks to this uniform device interface.

6. Evaluation

6.1. Test description

Tests are performed on a device control operation above the architecture described in the last section. An action is called on a simulated UPnP light that is installed on the Home SOA platform or a remote OSGi platform. Numbers show the time that is spent in the network part, in the base driver and the *SmartDevice* refined driver during an average action call.

The series of tests involve the same simulated UPnP light once on a remote platform and once on the same platform as the Home SOA solution. The UPnP light

setTarget method is called 100 times during each test unit of a set of 10. Average durations are given in Figure 7. Given base driver computation duration is the one corresponding to the remote communication case. In the local case, since the invocation calls remain local, the computation is much quicker.

Entity	Refined driver	Base driver	Network	Total
Duration	3,41 10-4s	4,89 10-3s	0,276s	0,282s
%	0,12%	1,73%	98,15%	100%

Figure 7 Time spent in Home SOA entities

Finally a table compares the approximate orders of the size of used base drivers. Exact sizes are not relevant since it mostly depends on the externalization of features in software bundles in the JVM and on the OSGi platform (e.g., HTTP server, XML parsers). The Figure 8 shows that the more the protocol is general, the bigger the driver is.

DPWS	UPnP	Bonjour	ZigBee	X10
600/25 kB	250/22 kB	200/20 kB	150/20 kB	50/20 kB

Figure 8 Size of base/refined drivers

The software architecture use apache felix OSGi platform and UPnP Base Driver, France Telecom open source DPWS Base Driver [9] and proprietary Bonjour Base Driver, Prosyst commercial X10 Base Driver. Tests are carried out on Sun J2SE 5 Virtual Machine delivering the JVM Tool Interface that is able to measure durations in 10^{-9}s. We use this Java tool with Eclipse Test & Performance Tools Platform installed on a PC with a 1.8 GHz CPU and 1GB memory. The power of the device explains that most of the time is spent on the network (98,15%) rather than on message computation.

6.2. Results analysis

The functionality of the refined driver – named *smartupnp* in Figure 6 – leads to an overhead of less than 7% to the base driver computation at operation call when devices are available. This computation duration entirely depends on the treatment that is made in the refinement. Treatment may involve a simple method translation up to complex algorithms. Here, an XML transformation is made on the involved input and output arguments in the communication with available devices.

The previous work [9] completes the whole figure with results on the service orientation and modularity brought to simple Java drivers. The service orientation layer brings a reasonable computation overhead with regards to the development advantages.

Separation of concerns, uniform interfaces and modular reusability simplify the development work. Distribution and heterogeneity are handled by the Home SOA layers. Application developers reuse drivers in an adapted technical mediation chain and focus on the business logic.

7. Related Work

The article [18] advocates the use of the Web Services at home for home device interoperability. Although our framework can be used to build a gateway translating all the home device protocol messages into Web Services messages (DPWS) on the network, we do not agree that the Web Services is the ideal network pivot for all device control applications. Our architectural vision is more opportunistic. The pivot can be a protocol in some applications where several devices communicating with each other have to be developed (pre-facto design), but it is rather a programming language API in applications composing services provided by third-party boxes (post-facto design). And today, if we stick to a pre-facto design, the best pivot between multimedia devices and home automation devices for example is probably UPnP rather than Web Services. The reasons are that Web Services are not used today at home and UPnP simplicity is more adapted for the translation of all home automation field bus protocols that are semantically limited. In other domains, such as the Enterprise domain, other protocol service bus may be used to bridge all the device protocols [19]. In a post-facto design, the examples in part 5 show that programming language APIs have to be chosen in an opportunistic manner. This API separates the business logic from the driver logic. We adopt the concept of a service platform where every entity of the network is reified to let applications compose services with the openness and the efficiency of programming languages like Java.

This work improves our first work – called the Extended Service Binder – on a distributed OSGi platform [15]. The service composition automation, precisely following the Service Binder model [8], is now separated from distribution transparent management delegated to separate service-oriented drivers. Service binding automation and distribution transparency are independent and complementary aspects in the architecture. The Extended Service Binder relies on RMI for dynamic proxy generation at bind-time for cross-network invocation of services and on SLP for the discovery of services on remote OSGi platforms. R-OSGi [16] is perhaps the most achieved

attempt to transparently distribute services on several OSGi platforms. R-OSGi goes further first with a dynamic proxy generation refined with ASM bytecode generation and moreover with type injection to resolve distributed type system and with remote failures mapped into local module hot-plug events. It is possible to make the Home SOA middleware additionally support the R-OSGi mechanisms as another specific protocol set. A service-oriented driver would then be responsible for the SLP discovery with the R-OSGi remote method invocation and event handling. This new service-oriented driver would bring an opportunistic way of distributing applications on several OSGi platforms. However, this driver would be useless if controlled services are not OSGi ones, e.g., in a standard device control application.

The objective of this work is close to ReMMoC's [20]: allowing protocol adapters to be plugged in at execution time and hide protocol heterogeneity to home developers. While ReMMoC component approach was innovative at the time of the paper edition, the novelty of our design is now the use of a discovery API for the discovery of software services seamlessly representing local and distributed entities. The use of this API hides distributed and heterogeneous aspects to developers. Code sharing on the OSGi platform is moreover leveraged to enable the control of local entities to be conveyed by direct programming language calls, i.e. Java method calls.

Furthermore, ReMMoC advocates the only use of WSDL-based abstractions to achieve representation uniformity while our opportunistic approach enables designers to choose the best syntactic pivot between the application and protocol drivers.

Indiss [21] makes the assumption that service interoperability can be achieved without modifying either service providers or service requesters. The vision of Indiss and related works like MSD [22] is fundamentally different. All service clients and providers are developed with specific protocols. The MSD architecture acts as a bridge between topologic networks and between discovery technologies. It does not address software modularity to achieve distribution and heterogeneity transparency. The advantage of the approach is the externalization of heterogeneity management in the network architecture rather than in the software design. The main drawbacks are the bandwidth usage intensity and moreover the performance loss of a network bridge approach in comparison to the performance of a software driver.

Nakazawa et al. [23] expose an interesting categorization of interoperable middleware characteristics. They built a middleware that shares a mediated translation approach with device aggregated visibility on a common platform similar to the one we have on our service platform. However, the service orientation is first not visible on their platform. Second, while the authors advocate a fine-grained compatibility model based only on input and output service data types, we believe that semantically consistent entities have to be reified as is for service composition consistency. Since automatic ontology alignment does not seem viable today, we followed a coarse-grained compatibility model when designing refined drivers in the multimedia use case.

8. Conclusion

Many protocol standards compete in the home market today. These protocols are mainly IP plug-n-play protocols in multimedia and communication domains while various field buses exist in the home automation domain. Mixing several protocols and domain applications in innovative home applications remains a painful and error-prone development task.

In the paper, we share a platform-centric vision that can be applied in home application design to protocol-agnostic applications. Our work can be distinguished from the state of the art by proposing a modular architecture that can be opportunistically adapted on various topologies, protocol sets and platform technologies. The main contribution brought by this work is the integration of many home protocol drivers in an architecture where functions are registered in a uniform programming language API. The service composition consists of a simplified task: chaining syntactic functions together in reactive adapters.

We implemented this architecture and tested applications above the OSGi R4 service platform. Felix open source platform is our reference platform implementations. Parts of the work are delivered open source in the IST Amigo project and some related actions are carried out into the OSGi standardization process [3][4][5].

The main perspective of this work is the optimisation of the reification platform mechanisms. The immediate local representative instantiation of every entity distributed on the network will be subordinated to the application needs. This future work will improve the memory consumption of our reactive software architecture and make it even more acceptable for embedded home devices.

9. Acknowledgements

Special thanks to various software developers who have contributed to the framework during a period within the two past years in France Telecom: Gregory Bonnardel, Teva Paoletti, Nicolas Peru, Xavier Goblet, Eric Simon, Stéphane Seyvoz, Marius Tankeu, Judex MBoulou.

10. References

1 M. Weiser, "The computer for the 21st century", Scientific American, 265(3):66-75, September 1991.

2 OSGi Alliance, "OSGi Service Platform Core Specification and Service Compendium Release 4", October 2005.

3 André Bottaro, "RFP 72 Extended Mapping for UPnP Discovery Transparency", OSGi Alliance, April 2006.

4 André Bottaro, Anne Gérodolle, Sylvain Marié, Stéphane Seyvoz, Eric Simon, "RFP 86 DPWS Discovery Base Driver, OSGi Alliance", May 2007.

5 André Bottaro, Anne Gérodolle, Sylvain Marié, "Combining OSGi technology and Web Services to realize the plug-n-play dream in the home network", OSGi Community Event, Munich, Germany, June 2007.

6 Feng Zhu, Matt W. Mutka, and Lionel M. Ni, "Service Discovery in Pervasive Computing Environments", IEEE Pervasive Computing, vol. 4, pp. 81-90, 2005.

7 Apostolos E. Nikolaidis, Serafeim Papastefanos, Gregory A. Doumenis, George I. Stassinopoulos, Marios Polichronis K. Drakos, "Local and Remote Management Integration for Flexible Service Provisioning to the Home", IEEE Communications Magazine, Oct. 2007

8 H. Cervantes, R. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model", 26th ACM International Conference on Software Engineering, Edinburgh, May 2004.

9 André Bottaro, Eric Simon, Stéphane Seyvoz, Anne Gérodolle, "Dynamic Web Services on a Home Service Platform", 22nd International IEEE Conference on Advanced Information Networking and Applications (AINA-08), Okinawa, Japan, March 2008.

10 Martin Fowler, "Inversion of Control Containers and the Dependency Injection pattern", http://martinfowler.com/articles/injection.html, 2004.

11 Pavlin Dobrev, David Famolari, Christian Kurzke, Brent A. Miller, "Device and Service Discovery in Home Networks with OSGi", IEEE Communications magazine, Volume 40, Issue 8, August 2002, pp. 86-92

12 W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, "The Case for Recombinant Computing", Xerox Palo Alto Research Center Technical Report CSL-01-1, April 2001

13 Elmar Zeeb, Andreas Bobek, Hendrik Bohn, Frank Golatowski, "Lessons learned from implementing the Devices Profile for Web Services", Digital EcoSystems and Technologies Conference (DEST'07), Inaugural IEEE-IES, Cairns, Australia, February 2007.

14 Hylke W. van Dijk, Hans J. Scholten, Alvaro Tobalina, Victor Garcia Munoz, Stephane Milanini, Antonio Kung, "Open Home Networks: the TEAHA approach", Sixth International Conference on Networking, 2007. (ICN'07), Sainte-Luce, Martinique, France, April 2007

15 André Bottaro, Anne Gérodolle, Philippe Lalanda, "Pervasive Spontaneous Composition", First IEEE International Workshop on Service Integration in Pervasive Environments (SIPE'06), Lyon, France, 2006.

16 Jan S. Rellermeyer, Gustavo Alonso, Timothy Roscoe, "R-OSGi: Distributed Applications through Software Modularization", 8th International Middleware Conference (Middleware 2007), Newport Beach, CA, November 2007

17 Java Community Process, "JSR 110: Java APIs for WSDL – Final Release 3", September 2006

18 Marco Aiello, "The Role of Web Services at Home", Advanced International Conference on Telecommunications (AICT/ICIW 2006), Guadeloupe, French Caribbean, February 2006.

19 Scott de Deugd, Randy Carroll, Kevin E. Kelly, Bill Millett, and Jeffrey Ricker, "SODA: Service-Oriented Device Architecture", IEEE Pervasive Computing Journal, vol. 5, no. 3, 2006, pp. 94-97

20 Paul Grace, Gordon S. Blair, Sam Samuel, "ReMMoC, "A Reflective Middleware to Support Mobile Client Interoperability", Proceedings of International Symposium on Distributed Objects and Applications, November 2003.

21 Yérom-David Bromberg, Valérie Issarny, "Indiss: Interoperable Discovery System for Networked Services", 6th International Middleware Conference, Grenoble, France, 2005.

22 Pierre-Guillaume Raverdy, Rafik Chibout, Agnès de La Chapelle, Valérie Issarny, "The MSDA Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments", Demo proposal, accepted for, Middleware 2005, Grenoble, France.

23 Jin Nakazawa, Hideyuki Tokuda, W. Keith Edwards, Umakishore Ramachandran, "A Bridging Frame-work for Universal Interoperability in Pervasive Systems", Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), 2006.