Distributed and modular service platform for
dynamic multi-service home networks

Tiago Sousa¹, Antonio Varela¹², and Teresa Vazao¹²

¹ Instituto Superior Tecnico, Lisbon, Portugal
² Inesc-Id, Portugal
tiago.sousa@tagus.ist.utl.pt avarela@tagus.inesc-id.pt
tvazao@tagus.inesc-id.pt

Abstract. Foreseeing the future of technology, Mark Weiser once said that it should integrate into people’s lives in such a way that it would become unnoticed. Service oriented architectures appear as an approach to address the challenges of creating such pervasive systems. These challenges concern the devices’ heterogeneity and mobility, privacy and security issues and systems’ scalability and adaptability to new scenarios. We propose a service oriented architecture for residential environments, which includes mechanisms for software discovery and composition, hardware control and services for high-level interaction with users. The service platform, based on standards such as OSGi and SCA, attempts to achieve the goals of modularity and reuse of components, scalability, adaptability, robustness and being agnostic to the communication protocols and implementation technologies used by components. A proof-of-concept application which tracks object’s location in a physical environment using RFID technology, was developed over this platform. Additional, the system is interacted by two different kind of clients: a legacy spoken dialogue system in Portuguese, integrated into the platform, and a newly developed rich-client graphical user interface.

1 Introduction

Two decades after Mark Weiser predicted ubiquitous systems [1] as the future of technology, people are now starting to interact and depend, during their daily lives, on an ever-growing number of services and devices. These systems are increasingly affecting more dimensions of human life: from their impact on economical markets to automation and control of physical environments, computer systems are everywhere and starting to integrate greatly in people’s lives, inclusively shaping their social relationships. To face these changes, systems have to “disappear” in peoples lives by focusing on users and their tasks [2]. However, the existence of so many technologies and devices, supported by several communication protocols and networks, creates a great number of challenges that motivates the creation of new paradigms and architectures which provide mechanisms for service collaboration in these environments [3]. Service oriented architectures (SOA) emerged as an effort to address some of the technological challenges existing in environments where several services interact with the user, by taking
advantage of context information from the environment. Decoupling the services in their building blocks and promoting the separation of functionalities requires modular architectures, which allow for more effective solutions in dynamic environments where an entire set of services might not be available. Also, system architectures should provide service discovery and composition mechanisms that are agnostic to underlying technology supporting services (communication protocols and implementation languages) so that interoperability between different devices and services becomes possible. Service discovery protocols include, for instance, the Service Location Protocol (SLP) [4], Jini [5] or Universal Plug-n-Play (UPnP) [6]. To finalize, the user must be the central piece of service oriented architectures so that its information can be shared and used by different services when delivering contents and functionality back to the user. [7]. These aspects are particularly relevant within the scope of home networking, where a wide variety of devices, technologies and applications coexists. In this paper, we propose a dynamic, distributed and modular service platform based on OSGi and SCA, aimed to support a wide variety of services, through different network technologies and access devices that are integrated into a single IP backbone. The platform provides automatic distributed discovery of services in dynamic execution environments. The platform also provides data integration compatibility for different clients, which improves interoperability independently from business logic and agile data persistence mechanisms. A case-study application, providing object tracking in a physical home scenario using RFID technology and developed on top of the framework will also be presented. This application, decoupled in its key building components, consists on:

- Multiple distributed RFID hardware controllers, automatically exposing low-level services and event notifying through the platform.
- Singleton RFID high-level manager, that automatically discovers controllers within the distributed environment in order to manage them and, consequently, process its low-level object tracking events. Through the use of a graph-oriented data model, semantic information from users, objects and locations, gathered from low-level events, is then stored to provide high-level services and functionality to users.
- A voice interface, implemented using an existing spoken dialogue system, to interact with the services. This legacy system is integrated through the interoperability mechanisms provided by the platform using web-services.
- A graphical interface, developed using a framework that shares component-oriented concepts (Spring-RCP), which can be used to demonstrate how a performance gain can be obtained with minimum service-definition impact. In this case, by using a communication protocol which is more efficient than the web-services used by the voice interface (the Java Message Service - JMS).

In the next section, we describe the background and related work, giving particular emphasis to the methodologies, technologies and tools that allow the creation of service oriented architectures. Our service platform is presented in section 3, where we describe the service model, the design and programming
goals and the architectural components. The platform was implemented in a prototype test-bed that is described in section 5. Several tests have been realized and enable us to claim that our platform is scalable and provide a fair access to the users. The main results are described in section 6 and the conclusions and future work presented in section 7.

2 Background and related work

2.1 Technologies

The SOA paradigm emerged as a new approach in software development that, unlike vertical and closed software applications, promotes the modularization of software in loosely-coupled components that collaborate with each other, locally or remotely, through service managing middleware. The main concept behind SOA is that of managing and using a distributed group of computational capabilities that may be under different domains. This can be seen in the same way as people creating capacities to address or support problems. In this way, one’s needs can be fulfilled by the capabilities offered by another. Visibility, interaction and effect are three fundamental aspects of service oriented architectures. Visibility refers to the means with which an agent with needs, and another who may fulfill those needs through capacities, can see each other (service discovery). Interaction relates to the communication processes that enable data exchange between components (service invocation). Finally, the effect corresponds to the result of interacting with services, through internal state change or the return of processed information. Two main technologies were used to accomplish the most important mechanisms that support service oriented architectures: service description, service discovery and service execution. Those are OSGi and SCA, which are driving the infra-structural needs for modular and dynamic software cooperating through distributed environments, and will be introduced below. The OSGi initiative aims to deliver an API, running on top of the Java Virtual Machine, which enables the development of agile and dynamic applications based on modular pieces of software (bundles). Started in 2000, and with its scope spreading to several software industries, the initiative delivers an unified service platform with the following goals:

- to establish a standardized development framework adaptable to different business needs.
- to provide a powerful model for co-operable components on a single Java VM, minimizing used resources, improving its performance and enhancing inter-application communication.
- to develop a flexible API to control application life-cycle, supporting dynamic installation, removal or updates without stopping operational systems.
- to deliver a secure runtime environment for applications, that allows its concurrent execution without compromising each others’ integrity.
- to build a cooperation model allowing applications to dynamically discover and use services offered throughout the platform.
– to provide remote platform control and management.
– to assure interoperability between systems through compliance mechanisms, allowing increased software reuse and better overall quality.

Eclipse’s core runtime, based on OSGi, is the most used implementation of the specifications: Equinox [8]. Apache Felix [9] and Knoplerfish [10] are other implementations compliant with the latest release of OSGi, version 4. The SCA standard, started in 2007, is the result of an industry driven effort to define a neutral programming model for service oriented applications which should be agnostic from programming languages and communication protocols. Through SCA it is possible to describe, not only the building parts of applications (SCA components), but also the way they orchestrate themselves through a composition (SCA composition) and the means used to provide communication between them (SCA bindings). This represents a description that is agnostic to implementation types, setting the service description free from infra-structural issues or communication protocols.

2.2 Related Work

Home networking has been a very important research topic for the last few years. Since the moment when access networks became capable of providing sufficient bandwidth for multimedia services, a lot of interest has been focused on home networks and new ways of creating and providing new services on top of that infra-structure. In this way, two kind of research topics are converging: on one hand, semantic description of services and discovery mechanisms inside of the environment and, on the other, increasing home gateway processing capabilities that can enable it to control and manage this increasing number of services and different devices. Projects like the Virtual Networked Appliance (VNA) [11], the SUIT [12], Semantic HiFi [13] or NASUF [14] propose solutions for semantic description of services and consequent discovery in distributed environments based on specific technologies like web-services or UPnP. From the home gateway research topic, projects such as i2Home [15] and MediaNet [16] propose solutions based on OSGi for concurrent execution of services provided by different operators, while maintaining security across the platform and unique means of interaction with services, without considering service providing in distributed environments. Finally, the MidGate [17], in pair with the VNA project, attempt to address the heterogeneity of communication protocols used by client devices.

3 System Architecture

Following the goals and orientations of the OSGi and SCA specifications, a service platform to apply in a multi-service residential environment is proposed, and its design principles and architectural options presented. The platform, based on distributed components, aims to achieve the characteristics of scalability, modularity, fault tolerance and service personalization that can be the foundation to ease the development of agile and flexible applications for dynamic environments
3.1 Service model

The service model presented below establishes a conceptual organization for each service running on the platform, in accordance to its role. Although there are several abstraction layers in services, they all run, at an atomic level, as OSGi bundles in local containers. These can be deployed in any of the nodes building the distributed runtime environment and executed in any of the others. Therefore, the model presented represents a global picture for service functionality, independently of its location.

Each of the categories will be described below:

- **Service Orchestration** Group of services which deliver the core functionality for SCA service execution. These include the OSGi runtime, the service discovery mechanisms and the automatic composition of service framework. Additional support services, such as services to organize and save data in a persistent way or data interoperability enhancers, are also included in this category.
Low-level Services: This category includes those services which provide technological specific functionality, acting as wrappers of hardware control software, and exposing them to other services or clients, which may then discover and interact with them. These include TV services, telephone, video on demand, domotics, RFID readers, sensor networks, etc.

Service Clients: Service consumers are included in this category and can make use of service discovery and interaction through the platform. These include any kind of interfaces, from desktop applications to HTML/Javascript or Flash web-interfaces, pocketPC applications, spoken dialogue systems, etc.

Additional Services: These services take advantage of contents available on the platform (other services, web contents) in a way that, without having to control specific hardware, they can create new added value functionality for both clients and other services.

Virtual Services: The virtual service concept relates to the possibility to create new services from the composition of already existing ones in a semi-automatic way by the user, for instance, using a graphical interface.

3.2 Design goals and programming model

This section presents the non-functional goals to be obtained through the use of the proposed service oriented architecture. Those goals are: Standards based by making use of OSGi and SCA, as the industry driving standards for SOA, will empower the platform to make use of their functionality and facilities in service development. Combining the dynamic runtime environment of OSGi with the distributed service composition mechanisms provided by SCA, it is possible to create distributed applications that can easily be enhanced, adapted or changed, as the communication protocols or implementation technologies are completely separated from the business logic; Modularity and component reuse by developing modular software, in which an application is seen as the orchestration of different components and modules, strongly promotes functionality separation and software reuse. Additionally, makes it easier to add functionality without breaking already working systems; Scalability and Adaptability by having a distributed platform based on components and services strongly adds resilience and performance capabilities to service providing. Organizing which processing nodes run each service can be manually or automatically defined, and execution-time re-organization is not only available in case of failure as it can be tuned to be based on QoS or performance metrics; Infrastructure Agnosticity by creating an abstraction layer for execution runtime, communication protocols and implementation technologies, strongly enhances business logic separation which adds flexibility and agility to the software development process; Ease of Development since service oriented applications give more emphasis to the description of components and tools used to achieve some functionality than on the code necessary to provide that functionality. Business logic can be made available through simple POJOs (Plain Old Java Objects) which not only contributes to faster development times but also makes it much simpler to maintain.
and understand systems through operation time and Robustness by the principle of loosely-coupled components distributed throughout the environment not only adds fault tolerance as it promotes the development of software that expects failure to be natural, making it much simpler to recover when it occurs: systems which accommodate failures by expecting them and having mechanisms to rapidly recover from them will generally have a lower downtime than ones which attempt to predict and prevent all faults, but have no way to recover from a failure caused by an unpredicted one.

In addition to previous non-functional goals to achieve within the service platform, a programming model for service-oriented applications that is intimately connected to the SCA and OSGi standards will be presented. This model is based in a design pattern called the whiteboard pattern. Assuming the execution environments are extremely dynamic, this model motivates a more effective way of handling events generated from different sources than that of the listener pattern. The main concept is that each service doesn’t need to know which components are listening to its events, consequently it doesn’t need to maintain references to them. All this is automatically managed by platform code, through the use of a shared service registry.

3.3 Architectural components, tools and technologies

The proposed software architecture, describing key platform execution components, will now be presented. The most important components of the the platform are those that build up its core, providing the dynamic runtime environment, service discovery mechanism, automatic service wiring and composition and the support services for data interoperability and data persistence. Each of those components will be described below, explaining which tool was used to provide its functionality and briefly refer to some of their insights.

**OSGi execution environment** The OSGi runtime provides the fundamental functionality for modular and dynamic software support through bundle management, namely: managing bundle’s life-cycle; controlling bundle’s parameterization through properties; managing bundle’s exporting interfaces; allowing service ranking attribution; managing bundle’s imported references and supporting dynamic bundle’s install, removal and update. Due to its greater user base, and shown performance in such a widely used tool as Eclipse, the OSGi container selected was Equinox [8].

**Spring service wiring** The Spring tool [18] is an application development framework using POJOs with the following goals: provide a non-invasive programming model, where application code is independent from framework code through configuration; establish an infrastructure for software development in Java that makes it possible for programmers to focus on business logic, abstracting from generic application problems and simplifying and fastening software development processes. The main concept behind these goals is the Inversion
of Control (IoC) which implies moving the control from the application code to the framework code. It is based on the Hollywood principle [19]: “Don’t call us. We’ll call you!” Through Dependency Injection it is possible to accomplish this only through configuration, guaranteeing a complete separation between application and framework code. With this principle in mind, an initiative to build the Spring framework on top of OSGi made it possible to have the best of both worlds through the Spring Dynamic Modules. These are OSGi bundles providing Spring functionality to every other software bundle’s running on the OSGi dynamic environment. The main advantages of building Spring on top of OSGi are: providing better business logic separation through modules; supporting different versions of software modules in the same environment; making it possible to discover and automatically wire up modules during runtime; allowing for dynamic install, removal and update of modules during runtime; providing intra and inter-wiring of modules through Spring and simplifying the OSGi programming model for developers through a familiar and more accessible model.

**Newton distributed composition** In the same way the Spring Dynamic Modules were built on top of OSGi, the Newton tool provides a distributed service components architecture based on OSGi, in which groups of local OSGi environments are wired together through Jini technology in a distributed execution environment. In a difficult exercise, one can summarize the Newton framework as a distributed container for software models that are component-oriented. As the previous tools, Newton tries to create an abstraction layer that makes it possible for programmers to develop distributed service oriented applications without worrying about the challenges and issues of a distributed environment. While focusing on business logic, Newton also provides compatibility to Spring implementation types, making it extremely suitable to hold Spring models through its SCA compositions describing business services.

**Mule enterprise service bus** Being the services environment rich in different services providing different functionality consumed by a great variety of devices, the existence of mechanisms to auxiliate integration through data formats adaptation is provided by an enterprise service bus [20]. The advantages of this approach consist in freeing the programmer from concerns about data interoperability by providing another abstract layer for decoupling business logic from communication protocols adaptation code. This functionality is provided by the Mule Enterprise Service Bus [21], which can be described as a lightweight framework to exchange messages and manage communication details between several services and clients in a distributed environment. The only entity the programmer needs to know about is the Universal Messaging Object, which is responsible by receiving requests, routing them to adequate processing through business logic and replying the results.

**Neo graph-oriented database** Another support service provided throughout the platform is a graph-oriented database available through the Neo4J tool [22].
Besides its compatibility with OSGi and roadmap integration with Newton, the need for a robust, scalable and high-performance object oriented database was driven by the following reasons: **Object-relational impedance mismatch** Although relational databases have been one of the greatest advances in computer science, there are some limitations in their compatibility with object-oriented data models, which is named the “object-relational” impedance mismatch [23]. In order to address this mismatch, object-relational mapping tools were made available, which add an extra-layer of complexity to applications, sometimes preventing its flexible and agile evolution. **Semi-structured and constantly evolving data** Additionally, one of the requirements for pervasive computing environments is the constant evolution of data managed by services, which drives the need for agile development techniques. As such, object-oriented persistence engines are much more suitable to evolving data models and supporting semi-structured data, characterized by the existence of a few mandatory attributes and lots of optional data format.

**Other services and clients** Other services to be run on top of the platform include common residential services (categorized as hardware services) and web-related services (categorized as additional services). Although services like television, IP telephony, domotics and sensor networks integration were studied, they haven’t been ported to the platform yet. Additionally, making use of the platform itself or some of its tools (such as the enterprise service bus), several client types can be supported, including HTML/Javascript or Flash web-interfaces, voice interaction interfaces, mobile applications, desktop rich clients, etc.

The prototype implementing the proposed concepts and architectural goals will now be presented. This prototype was developed in two phases: the integration of tools was made through configuration and integration code to deploy the distributed runtime environment for SCA compositions, including the core and support services already presented; and a proof-of-concept application was developed. This application runs on top of the service platform, takes advantage of the functionality that it provides and adds new services to the pot. The vertical application was developed on top of the platform following the identified design principles and is based on SCA and OSGi. It consists in using RFID technology to track digitally tagged objects in a physical environment. The application was structured in different components which will now be presented:

- **RFID hardware controller** This component is responsible for controlling RFID hardware devices and providing a set of common functionality to higher-layer services, thus supporting several hardware vendors. Additionally, the design assumes the existence of several hardware controllers in a distributed environment that can generate low-level events. Its main responsibilities are: managing the hardware, process and store low-level events and notify those events to registered listeners.

- **High-level RFID service manager** The high-level manager is a singleton service per environment, that automatically discovers RFID controllers in
the network in order to configure them and listen to their generated low-level events. Its goal is to map data from hardware devices to physical world objects and locations, giving semantic value to the events detected. This is done using the graph-oriented database and creating high-level services based on the data collected, which is then exposed to clients through different communication technologies made available through the enterprise service bus.

- **Spoken dialogue interface in portuguese language** A spoken dialogue interface was used [24] to illustrate the platform capabilities in integrating legacy systems, in this case making use of web-services support through the enterprise service bus. Additionally, it represents a new way to interact with services that’s extremely suitable to residential environments, becoming closer to Weiser’s view on ubiquitous technology [1].

- **Component-oriented graphical interface using JMS** A proof-of-concept graphical interface was also developed for two main reasons: firstly, because it uses a rich-client platform (Spring-RCP) that shares most of the concepts used in the server-side (platform and services), unifying the programming model for programmers; secondly, because it uses an alternate way to communicate with the services that has almost zero-impact on the service implementation and definition (with only one line of code being needed to upgrade the service to support JMS), that greatly increases communication performance.

4 Tests

To evaluate the proposed solution, two different kind of tests were run which, on one hand, verify the correctness of services developed in top of the framework in terms of functionality, and, on the other, evaluate platform performance and other non-functional requirements regarding to its core components. Tests were run on a 2.5GHz Core2Duo processor with 3Gb of memory. Windows Vista SP1 was used, although the solution is independent from the operative system. The RFID hardware was provided by Phidgets readers and tags. On the software side, the following versions of tools were used: Java Virtual Machine - version 1.6.0-10-rc; Newton - version 1.2.3; Spring DM - version 1.1.0; Spring - version 2.5.1; Equinox - version 3.3.1; Mule - version 1.4.4; Neo4J - version 1.0-b6 and Spring-RCP - version 1.0.1. Functional tests are not going to be described here but results shown correct functionality of the developed services. Results of non-functional tests are going to be presented bellow and conclusions will be taken.

Communication technologies comparison The following graph presents a comparison of invocation times using different clients and tries to verify the performance impact of the communication technology used. It presents the response time measured on both client and server for 10 invocations, with the server being run on debug mode, which was necessary in order to measure the time on
the server-side. And also the response time measured on the client-side during a stress test which consisted of 1000 invocations, with debug mode turned off on the server.

Fig. 2. Comparing web-services e JMS as messaging technologies in different clients

Observing the results, we can notice better performance of JMS technology when compared to web-services, as expected. We can also observe that the execution time on the server-side is almost the same independently of the clients used. Since the changes needed to add JMS support on the server-side were one line of configuration, one can conclude that, not only can this platform provide interoperability, but also allows performance gains with minimal changes in services (and zero-change in implementation).

**Scalability of the graph-oriented database** The scalability and performance test of the graph-oriented database was done comparing mean invocation times for queries about current item location, while using the RFID tracking service to increase the number of stored events. The blue graph shows the number of locations of an item throughout time, while the red one shows mean invocation times as the number of locations stored increased.

We can conclude that the number of stored information didn’t affect system performance when executing queries, proving that the proposed solution for data persistence matches the requirements.

**Fairness concurrency when accessing services** The last platform test held simulates concurrent access to services and tries to evaluate the fairness in service access for the multiple clients. It shows the mean invocation time for each client, as this number increases in each test run. Analyzing the graph it is possible to
Fig. 3. Invocation times and the increase of trackable events

conclude that, although mean invocation time per request increased, all clients saw their requests replied (none of the requests were dropped) at about the same mean time.

Fig. 4. Fairness in accessing services concurrently

5 Conclusions

A modular, dynamic and scalable service platform for residential networks is proposed in this article to enable flexible and agile development of component oriented applications based on OSGi and SCA. It attempts to provide mechanisms for automatic discovery and composition of services in dynamic runtime environments, while being agnostic to communication protocols, based on high-level
descriptions of services and compositions. From an SCA description of components and its compositions, it is possible to provide high-value services for users in a distributed runtime, which supports hotplugging of services during execution time. This is done from an orchestrated integration of tools like Equinox, Spring Dynamic Modules and Newton through Jini, with extra functionality provided by an enterprise service bus (Mule) and a graph-oriented database (Neo). With the developed proof-of-concept RFID application, it was possible to try out and evaluate most of the concepts proposed by the architecture, namely the automatic composition of components into services in a distributed fashion, the dynamic install and removal of software without stopping the system and the importance of an enterprise service bus to adapt and support different data types, providing a greater interoperability within services and clients through small changes in code. Also, testing showed how scalable the solution can be, including the graph-oriented database, allowing software to be developed in a more agile and flexible fashion, without compromising performance.

5.1 Future work

As this work reflects an initial approach to a bunch of new technologies and specifications, and tries to prove some of their potential, it still leaves some questions and paths to be explored and investigated. Among other upgrades and developments to be made, three different paths for future work are:

- In the platform functionality side, along with better integration of tools as services, an unique description based on SCA that could configure every other tool in runtime (Newton, Spring, OSGi container, Mule) would contribute to configuration re-use and minimize its redundancy throughout different tools.
- In the services side, although some requirements analysis was made for most of the common residential services, wrappers for these hardware services to run on top of the platform were not implemented.
- Finally, taking advantage of modular and dynamic software, a mechanism of plug-ins, that could be used by service clients to add functionality to their interface based on runtime learning of new services made available, would extremely enhance user-experience and improve the development time-to-market of creating new value-added services within the service platform.

Lastly, the same principle of the OSGi runtime, where deployed software can take advantage of the already existing functionality at the same time it adds new one to future services, can be generalized to high-level distributed services (through SCA descriptions) that can enrich and drive the constant evolution of services on top of the platform.

References