KALIMUCHO
Software Architecture for Limited Mobile Devices

Cyril Cassagnes
LaBri/Univ. Bordeaux I
350, Cours de la Libération
33405 Talence - FRANCE
cyril.cassagnes@gmail.com

Philippe Roose
LIUPPA / IUT Bayonne
2, Allée du Parc Montaury
64600 Anglet - FRANCE
Philippe.Roose@.univ-pau.fr

Marc Dalmau
LIUPPA / IUT Bayonne
2, Allée du Parc Montaury
64600 Anglet - FRANCE
Marc.Dalmau@.univ-pau.fr

ABSTRACT
This paper presents a software platform (called *Kalimucho*) for mobile and embedded hardware. The architecture of such application is based on two component models: *Osagaia* for software containers and *Korrontea* for connector containers. *Kalimucho* uses service factory permitting the creation of instances of software components. This factory allows making the most suitable configuration according to services decision managing reconfiguration. The reconfiguration service decisions depend on data sent by control interfaces and the users defined rules (via the platform and control interfaces).

Categories and Subject Descriptors
D.2 [**Software Engineering**]: Requirements/Specifications, Design, Software Architectures, Interoperability, Adaptable Architecture, Reusable Software.

General Terms
Design, Experimentation.

Keywords
Heterogeneity, embedded devices, CDC/CLDC, platform.

1. INTRODUCTION
Because of the massive growing of embedded mobile and limited devices uses in everyday life [19] (public transport, workplace, cars, domestic appliance, etc.), the study of distributed applications for such devices becomes a major thematic. Such tools do exist but are still too strongly relative to underlying technologies. Proposing design methods and software architectures models for the design, implementation and running of applications on mobile limited hardware is crucial. The objective of this paper is to present our platform able to manage dynamic reconfigurations of distributed applications deployed on various and heterogeneous embedded devices. The goal of the platform is also to ensure the dynamic management of software components in order to guarantee an adequate quality of service to the end user. The reminder of this paper is the following. We firstly present component and service-oriented component characteristics, and then we present related work our component model. In the third part, we introduce the Kalimucho platform: objectives, architecture, and functionalities. Finally, we present some perspectives before the conclusion.

2. DISTRIBITED APPLICATIONS
An application for heterogeneous mobile embedded and limited (low bandwidth, power consumption, etc.) device has to firstly prevent hardware and mobility limitation. We decided to use dynamic reconfiguration to bypass this problem. Intervention on the architecture of the application needs to have a specific software component model. Such model permits to act on the life-cycle of the components instances and to provide information on their execution state. Because we implemented the platform upon the OSGi service based platform, let's briefly make an introduction to OSGi.

2.1 Open Service Gateway Initiative (OSGI)
The OSGi (Open Service Gateway Interface) Framework implements a complete and dynamic service model [4]. Projects such as Apache Felix, Oscar or Concierge implement OSGi specifications. We use the power of OSGi with a component model over it.

2.2 Heterogeneity
The hardware heterogeneity leads an important limit for a full OSGi development. The platform is not available on all devices, and particularly on sensors or smart sensors. So, we cannot deploy the OSGI framework to the Sun SPOTs because the Squawk VM hosted on Sun Spot Sensors implements CLDC 1.1 specifications whereas OSGI requires CDC due to its need of the java.lang.ClassLoader mechanism. On such devices we shall thus use the isolate (JSR 121) mechanism intead. The component framework that we have designed and implemented take into account such limitations to bypass the heterogeneity limit.

2.3 Service oriented software components
A classic component model provides a separation of concerns (functional/non-functional preoccupations) but do not provide a tool for dynamic management. Service oriented approaches proposed new models called service oriented

components. Several service oriented models do exist [5] and particularly over the OSGi one on which we will base the Osagaia model.

Certainly the most used by professionals is the Spring Framework (Spring Dynamic Modules) [17]. Nevertheless, it has two unacceptable restrictions: it does not permit substitution of service providers (into a composition) and doesnt support reconfiguration [16]. Moreover, Spring is more oriented for application server development. It exists other models, more opened and extendable as iPOJO [15]. This model, proposed by the Apache foundation is based on OSGi. It mainly proposed customization by handlers. A handler is an object who can be plugged on an instance container. It is possible to extend the component model by creating a specific handler. A handler manages one non-functional requirement of the instances [16][17][18]. Nevertheless, it depends on the following OSGi platforms: Apache Felix, Eclipse Equinox, Knoplerfish and cannot implemented without OSGi platform (or on other OSGi platforms), this is a problem for the heterogeneity, because it can not manage hosts as Sun Spots. Nevertheless, the handler mechanism is very interesting and the Osagaia model widely draws with: InputUnit, OutputUnit and ControlUnit, able to manage input/output data flows as well as control flows and commands. We also (as others models) will use component description [13][2] in order to describe the architecture of the application. So, the non-functional part of the Osagaia model is the same for all components composing the application.

2.4 Related Works

In [16] authors explain how interesting it is to provide dynamic reconfiguration with sensor networks. In [20] we can find a software component model for sensors with connectors dedicated to measures. Our approach is close to this philosophy. We propose to goes on and uniform the deployment of components for dynamic reconfiguration of application of heterogeneous hosts.

Building such applications has been the topic of several works [4][5][17][19] [13]. Nevertheless, none of such design method integrates strongly limited hosts. Some of them use additional software layers highly consumer of resources, unacceptable for very limited hosts.

The goal of out software component model is to facilitate the work of the software developer. He focuses on the applicative/functional part and dont care about mobility, dynamic nor heterogeneity. So, we propose a Java uniform framework. Software architectures are often coupled to specific hardware [13] and highly power consumer. We propose a lightened architecture able to be deployed on (very) limited hosts. We do not use middleware but a set of services integrated to the Java platform with specific component and connectors containers, called Osagaia (means component in Basque language) and Korrontea (means connector in Basque language). The following part details each of these containers and how they interact with the platform.

3. COMPONENT MODEL

Functional properties implemented into the components are the core of the application. Non-functional ones deal with mechanisms implementing the dynamic of the architecture and the supervision of the application by the platform which manages its QoS (user and hardware oriented).

3.1 Osagaia Component Model

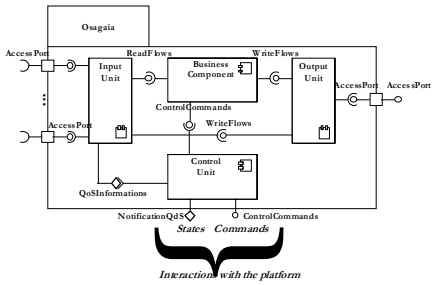

Figure 1. Osagaia Component

The Osagaia is a first class component container (Fig. 1) divided into three non-functional entities: InputUnit (IU), OutputUnit (OU) and ControlUnit (CU). The functional part of the component is called the Business Component (BC). The CU exports the interface for managing the life cycle (creation, configuration, validity, destruction), initializes configurable values of component, receives commands and sends states. The configuration deals with the management of QoS events. These events are then captured and evaluated by the platform which triggers dynamic reconfiguration [3]. The IU and OU allow the container to be connected to the corresponding Korrontea connectors according to the connection commands send by the platform.

3.2 KORRONTEA CONNECTOF MODEL

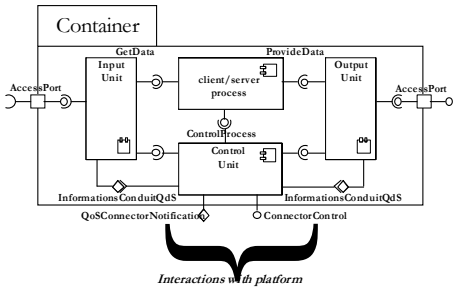

Figure 2. Korrontea Component

Because the assembling of Osagaia components may need various communications paradigms, we decided to design a connector model. The Korrontea container is based on the same architecture as Osagia (first class component). The main difference is that the Business Component implements communications modes. Each one is made of a triplet: Communication (local, distant), Mode (Synchronous/ Asynchronous/Real-Time/With-Without loss), Interface (socket, RPC, RMI)}. Thanks to the CU, the platform also supervises connectors in order to detect transmission problems (bandwidth, loss of connection, errors, etc.) and to change communication and interface characteristics. A distributed application is deployed with Osagaia components instantiations linked by Korrontea connectors. Our component model allows creating a dynamic architecture as well as giving QoS information through the CU.

The platform receives such QoS data; compute them in order to decide if a reconfiguration is needed.

4. KALIMUCHO platform

The objective of the distributed platform, called Kalimucho (means "lot of quality") is to supervise the Osagaia software components and Korrontea connectors as well as to reconfigure the application acting on life cycle of components/connectors or moving them on other hosts. A reconfiguration is done in order to ensure an adequate QoS [3].

Table. 1. JVM with corresponding JSR

Abstract Interface		Java Enterprise Edition (J2EE)	Java Standard Edition (J2SE)	Java Micro Edition (J2ME)					Existing OSGi Framework implementation
				Connected Device configuration (CDC) JSR 218					
							Connected Limited Device configuration (CLDC) JSR 139		
				Foundation Profile JSR 219	Personal Profile JSR 216	Personal Basis JSR 217	Mobile Information Device Profile (MIDP) JSR 118	Information Profile Module Profile (IMP) JSR 195	
Java Virtual Machine (JVM)	JAVA	X	X						Felix, Oscar
	Compact		X	X	X	X	X	X	Felix, Oscar
	Kilobyte						X	X	inconnu
	SQUAWK							X	Non possible
	Mysaifu	Libraries GNU Classpath							concierge

As previously mentioned, the hardware heterogeneity is hidden with the use of the Java Virtual Machines (JVM). This language is quit natural because of the wide variety of devices/hardware supported, and with the wide possibility to get light JVM for (strongly) limited (mobile) devices. The Java environment is also suitable for devices without OS as Sun Spot Sensors. It is made of several platforms (J2EE, J2SE, J2ME), Java API, and a JVM specific to each device.

4.1 Services

The Kalimucho platform is based on services. As the platform will be deployed on each device, including limited ones, the question is how to distribute those services (it is a utopia to deploy the whole platform on too limited devices) according to limitations of the host. So, all services cannot be deployed verbatim on all devices. Consequently, the platform has various architectures, which is totaly compatible with the Kalimucho service based platform. Kalimucho proposes the following functionalities:

1. Creation of Osagaia containers encapsulating the business component (*service factory*).
2. Création of Korrontea connectors establishing communications between two components (*service factory*).
3. Capture of QoS events from components/connectors and determine when they imply to reconfigure the application.
4. Provide new configurations (ie. new assembling) with addition/suppression/moving/substitution of components and links between them.
5. Ensure coherency between services distributed over devices (discovering, unicity, addition, suppression, etc.).
6. Manage the mobility of the business code, ie, BC.

Container and Connectors factories are a solution for the first and second point. They create entities specifically adapted to the host onto which they are deployed (it exists a container for each kind of host). The Supervision Service and Configuration Generator correspond to the third point. When an event occurs, the supervision needs the evaluation of the current QoS. Then the Configuration Generator provides another configuration, corresponding to the fourth point. The fifth point is solved with the Component Register having the knowledge and maintaining the list of available components for mobile and/or limited devices which are not able to run the OSGi platform. So the Kalimucho platform has a global vision of the deployed application.

Each service needs more or less resources. For example, the configuration generator needs numerous calculations. So, the selection of services to be deployed on limited hosts is done according to resources available on devices. That is why all services are not deployed on each one. The objective is to maximize the distribution of the platform in order to maximize the autonomy of each device according to its potential (memory, energy, power, etc.).

We present here the deployment of the platform according to resources available of the host. On mobile devices (see. Fig. 3), development is adapted to the CDC API. For example, the Configuration service is not implemented, so the Supervision service does not exactly run as on a fixed device with no limitations (desktop PC) because it can't directly access to this service. Consequently, the configuration evaluation will be done by the nearest non limited host. Factories still have the same role but they are adapted to the CLDC implementation interface.

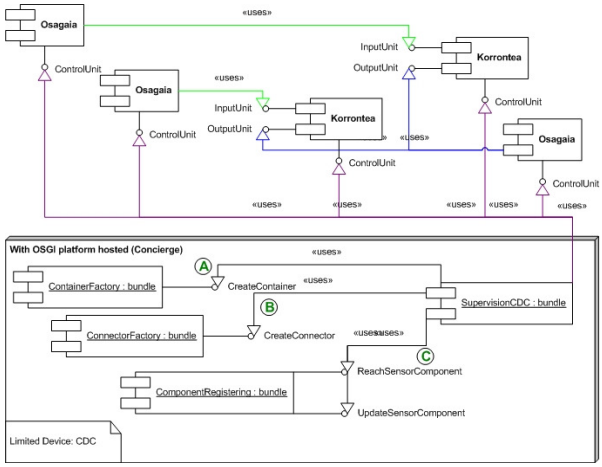

Figure 3. Deployment example of a limited device

4.2 Component Container factory service

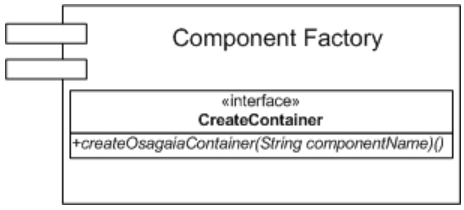

Figure 4. Component Container Factory

The container factory service allows building Osagaia containers. This service is available on each device family because instantiation of a software component is strongly coupled to the hosting device.

This service interacts with the supervision service in ordre to get the ID Card (a component description as used with COTS [1]) of the business component from a component repository available on the network. When using Squawk JVM on very limited devices (CLDC), the repository is embedded onto the device. This choice is done because such devices do not provide class loader mechanisms (the number of BC has to be limited due to storage limits on such devices). ID Cards contain static properties (name, version, etc.) as well as dynamic information (input/output connectors, target host, etc.).

4.3 Connector Container factory service

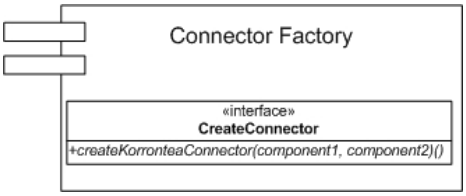

Figure 5. Connector Container Factory

Connector Container Factory as well as Component Container Factory receives commands from the Supervision Service. Such factory creates links between component instances. The connector encapsulates a business component implementing the communication mode/politic. If the connector is distributed on two devices, each supervision service builds one part of the connector. The supervision service provides information necessary to build the connection hosts (IP address, port, etc.). For a given communication politic, the designer has to provide as much connectors as device types. In order to modify the QoS, we must have several components able to provide the same service with various qualities, or we must provide various components or components combinations ensuring this service. The choice of the adequate combination is done by the platform.

4.4 Component Register Service

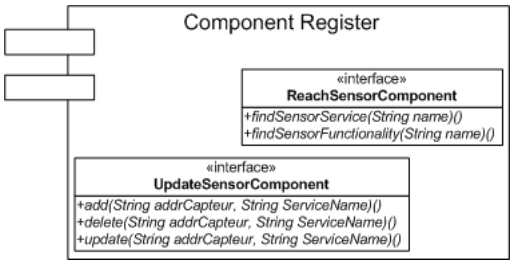

Figure 6. Component Register Service

The main task of the Component Register Service is to provide the two following interfaces: ReachSensorComponent and UpdateSensorComponent. Its role is to replace the lack of the OSGi platform, and especially the Service Register on CLDC devices. With this service, the platform is aware of components deployments of each CLDC compliant sensor. When reconfiguring, the Component Register Service sends the new deployment schema to the Supervision of the nearest non limited host. Then, the register is updated if needed and components hosted on sensors are reconfigured with the new local deployment schema.

4.5 The Supervision Service

The Supervision Service uses the set of previously presented interfaces. This service supervises all the components of the application using their Control Unit (CU) interface. Communication between the application and the platform are only done between CU and the Supervision Service hosted on the device onto which components are hosted. The Supervision Service controls the execution of the application using QoS events raised by each container and monitors the hardware resources (battery, memory, CPU, number of current connections, etc.).

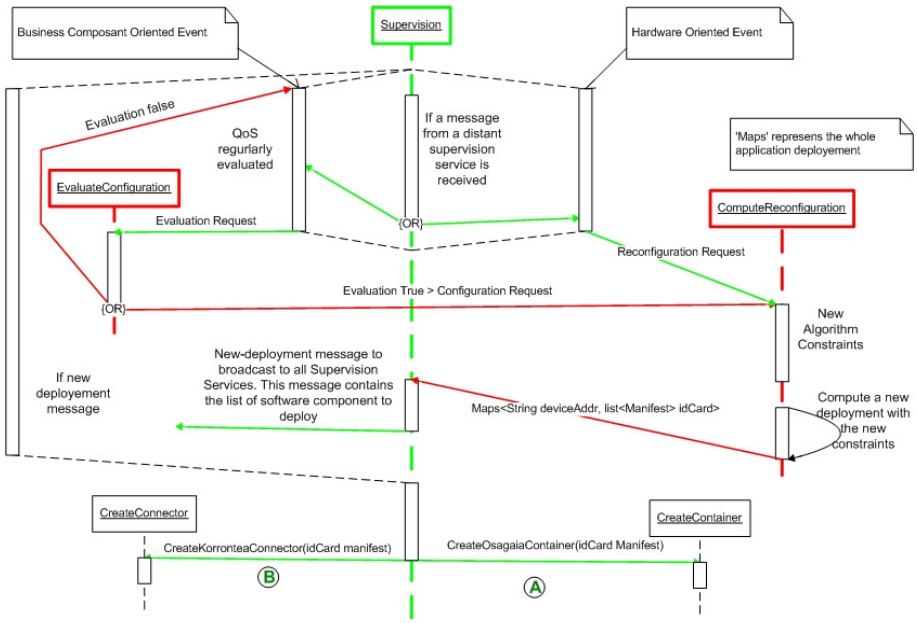

Figure 7. Supervision on a non limited device.

Because all services are not hosted on all devices, when QoS information is captured, according to the device type, it has to be relayed towards a Supervision service able to access to all services. Such organization implicitly expresses a hierarchy between Supervision Services (Supervision, SupervisionCDC, SupervisionCLDC) as well as it exists between devices (more or less limited). The organization is detailed in the figure 3:

When a local Supervision Service express the lack of a hardware resource (memory, energy, etc.), it raises a QoS event. The platform decides to act according to its configuration and thresholds values defined by the designer. If the device is mobile, the reconfiguration has to follow the Supervision hierarchy. When re-deploying, the Supervision service send query to the Configuration Generator with the QoS states. According to the values, the evaluation can be positive or negative. If so, according to the device, the Supervision Service gets the list of components causing the failure. This list comes from a description file sent by a distant device or by the Component Register and is transmitted to the Configuration Generator. It makes evaluation and starts the reconfiguration process. The new generated configuration is provided to the Supervision Service which broadcast the new deployment to corresponding hosts and starts its deployment.

Concerning the functional parts, the CU of the container detects that the QoS level is no more ensured. It then raises an event to the local Supervision Service ant the process goes on.

4.6 Configuration Generator Service

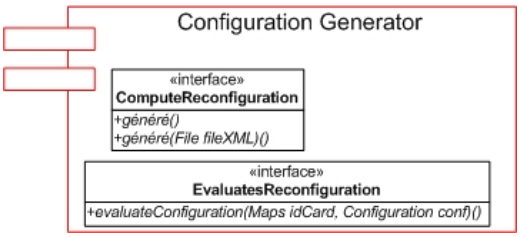

Figure 8. Configuration Generator Service

This service provides EvaluateConfiguration and ComputeReconfiguration interfaces. It uses an enhancement criterion and a configurable user defined criterion [2]. The final user can configure the platform by modifying critical range values (energy level, CPU charge, memory, execution time, etc.).

A reconfiguration can have consequences on the architectures. A local or distributed reconfiguration can trigger other reconfigurations of the architecture. The evaluation is firstly based on parameters given by the user, secondly, the Configuration Generator obtained from the description file of the application, the list of related components. Consequently, the configuration generator provides the list of component to suppress, to migrate and to deploy on each corresponding device.

5. Conclusion and Future work

The design of ubiquitous applications needs to integrate the management of the hardware hosts heterogeneity as well as functional aspects. This implies to provide solutions in order to reconfigure applications (software deployment) in such mobile and versatile environment.

The service based Kalimucho platform has been implemented in order to adapt its own structure to application hosts possibilities as well as functional adaptations. These works are supported by Sun Microsystems (for the use of SunSpot sensors), “Conseil Général des Pyrénées Atlantiques” with a technological transfer with Dev 1.0. Software components, connectors and the Kalimucho platform are available on PC (OSGi/Felix platform), PDA (iPaq/OSGi/concierge) and SunSpor smart sensors (Squawk Java Virtual Machine).

6. REFERENCES

[1] Philippe Roose - *IS-COTS: a help to COTS Products Integration* - Third International Off-The-Shelf-Based Development Methods Workshop (IOTSDM '08) - *Position Paper* - Held in conjunction (ICCBSS '08), 25/02/2008, Madrid, Spain, 2008.

[2] C. Louberry, M. Dalmau, P. Roose – Architectures Logicielles pour des Applications Hétérogènes Distribuées et Reconfigurables – NOTERE’08 - 23-27/06/2008, Lyon/France.

[3] S. Laplace – « Conception d’Architectures Logicielles pour intégrer la qualité de service dans les applications multimédias réparties » – PhD Thesis – University of Pau. 2006, France.

[4] M. Cremene, M. Riveill, C. Martel, C. loghin, C. Miron - *Adaptation dynamiques de services* – DECOR’04 Déploiement et (Re) Configurations de Logiciels, 28-29/10/2004 – Grenoble, France

[5] A. Ketfi, H. Cervantes, R. Hall, D. Donsez – *Composants adaptable au dessus d’OSGi* – Journée Systèmes à composants adaptables et extensibles - 17 et 18 octobre 2002

[6] P. Grace, G. Coulson, G. S. Blair, B. Porter – *Dynamic Reconfiguration in Sensor Middleware* – MidSens’06, Proceedings of the international workshop on Middleware for sensor networks - November 1, 2006 Melbourne, Australia.

[7] K. Geihs, M. U. Khan, R. Reichle, A. Solberg, S. Hallsteinsen - *Modeling of Component-Based Self-Adapting Context-Aware Applications for Mobile Devices* – IFIP Working Conference on Software Engineering Techniques, October 18-20, 2006, Poland.

[8] K. Balasubramanian, J. Balasubranian, J. Parsons, A. Gokhale, D. C. Schmidt – *A Plateform-Independent Component Modeling Language for Distributed Real-time and Embedded Systems* – In Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Sym., San Francisco, CA, Mar. 2005.

[9] R-OSGi and Distributed OSGi: differences and similarities - *http://mauriziostorani.wordpress.com/2008/09/04/r-osgi-and-distributed-osgi-differences-and-similarities/* - 4 september 2008.

[10] J. S. Rellermeyer M. Duller, G. Alonso – *Using Non-java OSGi Services for Mobile Applications* – MiNEMA’08, March 31-April 1, 2008, Glasgow, Scotland.

[11] J. S. Rellermeyer, G. Alonso – *Services Everywhere: OSGi in distributed Environments* – EclipseCon, 2007 March 5-8, Santa Clara.

[12] The SquawkVM Project - https://squawk.dev.java.net/

[13] The MUSIC Project - http://www.ist-music.eu/

[14] Y. Royon, S. Frénot - Un environnement multi-utilisateurs orienté service – In CFSE'2006, Octobre 2006, Perpignan, France, 2006.

[15] Apache iPOJO - http://felix.apache.org/site/apache-felix-ipojo-supportedosgi.html

[16] C. Escoffier, R. S. Hall - Dynamically Adaptable Applications with iPOJO Service Components - 6th International Symposium on Software Composition (SC 2007), Vienne, Autriche.

[17] C. Escoffier – « iPOJO : Un modèle à composant à service flexible pour les systèmes dynamiques » - PhD Thesis University of Grenoble , 2008

[18] A. Diaconescu, J. Bourcier et C. Escoffier, " Autonomic iPOJO: Towards Self-Managing Middleware for Ubiquitous Systems ", 1st International Workshop on Social Aspects of Ubiquitous Computing Environments (SAUCE 2008), Joint with : 4th WiMOB, October 2008, Avignon, France.

[19] C. Lee. S. Ko, S. Lee, W. Lee and A. Helal, "Context-Aware Service Composition for Mobile Network Environments," Proceedings of the 4th International Conference on Ubiquitous Intelligence and Computing (UIC), Hong Kong, July 11-13, 2007.

[20] M. Desertot, C. Marin, D. Donsez - SensorBean : Un modèle à composants pour les services basés capteurs - *Proceeding in Journées Composants JC'05,* Le Croisic, France, 2005-04-01