Towards a Flexible and Secure Runtime for Embedded Devices

Albert Royo Manjón, Eric Simon and Sébastien Jean
Laboratoire de Conception et d'Intégration des Systèmes (CTSYS Team), Université de Grenoble, 50 rue Barthélémy de Laffemas, 26000, Valence, France

Keywords: Service Oriented Architecture, Embedded Systems, Security.

Abstract: Advancing towards the Internet of Things, a need for bigger connectivity between every time smaller embedded devices is foreseen. In the near future, heterogeneous resource-restricted devices will probably have a set of services with a strong need for connection. Two needs are envisioned as mandatory: flexibility and security. There is firstly a need for some degree of isolation between services but there is also a need for services to be able to have their runtime altered without having to stop the whole platform. This generates a clash of interests and needs, since achieving both flexibility and security balanced is apparently incompatible. The purpose of this article is to explain the needs and requirements that such systems will most surely have, as well as inspiring technologies and related works, in order to advance towards a platform with flexible and secure services that will add bigger capabilities to the devices.

1 INTRODUCTION

The Internet of Things (Atzori et al. 2010), where daily real-world objects will be added interacting with users and between themselves, supposes a big opportunity for embedded devices. The definition of these devices doesn't specify their size and regarding those that are resource-restricted, the Internet of Things means several issues that will have to be taken care.

Nowadays, small heterogeneous devices tend to interconnect to work together and share data. Home gateways and sensor networks with different applications, using different technologies try collaborating. Each one of them works in a different way and under different architectures, but can profit greatly from such collaboration. Evolution is likely going to affect size, since sensor networks are too small for these needs and home gateways are not embedded enough. This way, either the first become bigger or the latter reduce their size, both turning into a single tiny device.

It is likely that these devices will offer and use services that will give them new capabilities, like data exchange or remote monitoring. Resource-restricted devices are already evolving towards connectivity, while keeping themselves small. Regarding the services, we can easily foresee two of the needs that will have to be fulfilled. Coming from several different providers, services cannot trust each other completely. The system has a need for a given degree of security that is necessary in the form of a certain extent of isolation between services, allowing services to have their own private space inaccessible by the others. There is also need for flexibility in runtime, so that operations altering a service's execution don't alter the rest nor restart the platform.

An important point as for services is the portability of their code. Services should be written once and then used and reused many times in many different devices. Because of this, the choice is to search for technologies developed in Java, that offers a higher portability (Hugunin, 1997). We have identified two technologies that satisfy one of the necessities: Java Card for security and OSGi for flexibility. None of them fulfils both of the needs, which shows an apparent incompatibility between them.

On the rest of this paper we will try to discuss which approach is better for trying to fulfil both of the needs on a new platform. Chapter 2 will explain the current context and identify the necessary properties for the platform. Chapter 3 gives an insight of the existing Java technologies and strong points for each one of them. Finally, chapter 4 will deal with the conclusions and the future work.

Royo Manjon A., Simon E. and Jean S..
Towards a Flexible and Secure Runtime for Embedded Devices.
DOI: 10.5220/0004756301680173
In *Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2014), pages 168-173
ISBN: 978-989-758-000-0

2 CONTEXT

Nowadays, most (if not all) of the target devices live exclusively in the physical world, receiving a number of entrances through their sensors and using a number of activators. This paradigm has been valid for the past decades, but with the introduction of the Internet into embedded devices, connection has become a major topic to discuss. Despite lacking power and resources compared to bigger systems, resource-constrained embedded devices can profit greatly from becoming connected.

However, their lack of resources has been a big barrier avoiding advances in the subject. In the close future, anyway, it is likely that these problems will be overcome. Devices, having gone past the physical world will become every time closer to a services gateway. Services will offer their capabilities and be used when necessary.

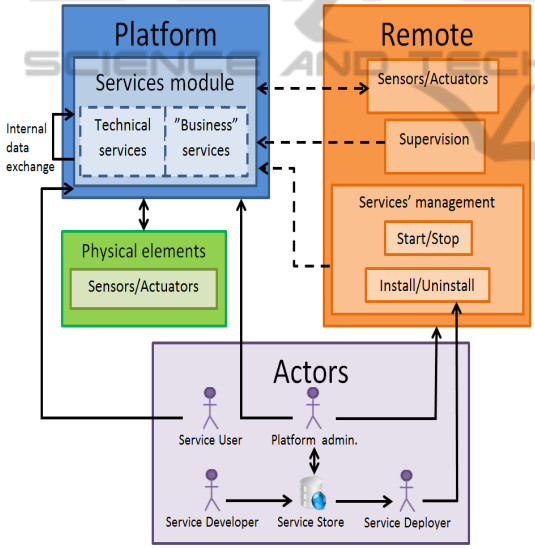

Figure 1: Platform actions and actors.

Figure 1 shows the way we envision our flexible and secure platform, showing both the actions and the actors. We distinguish between two types of services: technical and "business" services. Technical services are those related to the core, generic capabilities of the whole platform, such as external communication. They are key elements for the whole system, and so, they are generally managed only by its administrator. "Business" services, on the other hand, are related to the specific use that the platform is given. They are generally installed by service deployers and so, should be less trustworthy than the first ones.

We distinguish between several different actors related to the service, since we consider that for a service, its developer, its deployer and its user may not be the same individual. Developers will place their services into a service "store" from where they will be taken when needed. The service deployer will use the remote management capabilities to install the service into the platform. The platform administrator will be the only one able to remotely deploy technical services as well as having access to all of the remote capabilities. He is the actor who will be able to use the platform's full capabilities. Service users will make use of any type of services.

It is because of this amount of actors, as well as for the two different types of services, that there is a need for security. At the same time, applications will be changing frequently: new applications can be installed at any moment as well as existing ones can be removed, stopped or started. This turns stopping the whole platform unviable. Thus, there is a strong need for flexibility too.

2.1 Modularity and Dynamism

Services need to be reusable modules that once programmed for one device, can be reused on any other using the same platform. Services are not always needed and different operations could alter their execution. The four operations are installing, uninstalling, starting and stopping a service. Having standard, static modules is not enough, since our modules are dynamic and can have their runtime altered without altering the platform's.

2.2 Security

Services are external to the device's applications and independent, coming from different providers. None of the actors can trust the others if the platform does not give them the means. A certain level of isolation is needed to grant that services will not suffer from intrusions. At the same time, the roles of each actor have to be clearly defined, since each one will have access to different capabilities and some, like technical services' management cannot be granted to anyone because of its crucial nature for the whole platform (generally just to the platform administrator, though it could be sporadically delegated).

There is a major drawback about this and dynamism: they tend to suppose a clash of interests difficult to fulfil at the same time. It is necessary to find a solution offering both of them.

2.3 Resource-friendliness

Target devices are generally small in terms of resources for the flexible and secure platform we envision. We aim mostly for PLC-like systems, so we should be dealing with environments of around 32-bit CPUs, 128 kBytes of RAM and a few MBytes of persistent storage.

2.4 Synthesis

To sum up, future devices will need to enforce four properties: modularity of services, dynamism, flexibility of the platform and security, while being wary of a constraint: the embedded environment. Connectivity has also been added as an important point to analyse.

Already existing Java-based technologies have been compared in order to see if any progress can be done with already developed material, based on the following properties and values:

Table 1: Properties.

Property	Values
Modularity	No|Static|Dynamic
Dynamism	No|Yes
Security	No|Optional|High
Resource-friendliness	Extreme|Restricted|Standard
Connectivity	Optional|Native

3 TECHNOLOGIES

We have checked which Java-based technologies can fulfil one of our needs and analyse if it is possible to attain the second one.

The first need is security. Among all the Java distributions, Java Card (Sun Microsystems, 2008) has two main design focuses: portability (it is conceived for smart cards) and security. On the other hand, that of flexibility, a dynamic platform where services' runtimes are changed without altering the others' is envisioned. That is exactly the way that OSGi works (The OSGi Alliance, n.d.).

3.1 Java Card

Java Card is the tiniest Java platform. Developed with a focus on portability and security, it consisted originally of applications called applets running on a small and secure environment. Due to the need for connectivity, Java Card has been split into two editions starting on Java Card 3. They are:

- **Connected Edition:** It adds connectivity capacities to the main platform.
- **Classic Edition:** the heir to previous Java Card edition.

The most suitable Java Card edition for our interests is the Connected Edition. From now on, we will analyse it to see if it satisfies our needs.

3.1.1 Connected Edition's Architecture

This edition adds for the first time connectivity capabilities to Java Card. There are three different types of applications that can run on the Connected Edition:

- **Classic applets:** Standard Java Card applications, using the Classic Edition's API.
- **Extended applets:** New Connected Edition applets. They use Connected Edition's extended API to get new features.
- **Servlets:** Web applications. They use their own API. This type of applications interacts with off-card Web clients via HTTP or HTTPS requests and responses.

Since extended applets and servlets are the only ones having connectivity capabilities, it can be deduced that they are the most similar to our services.

Regarding the main characteristic, security, Java Card inherits Java features (exclusive access to encapsulated data, type mismatches detection during compilation…) and others proper to Java Card itself. The most important one of these is that by using a mechanism called Firewall, each application is allowed a secure application environment.

3.1.2 Connected Edition's Conclusions

Java Card is the tiniest Java platform that exists. Proper to all Java Card editions is the focus on security, and it is difficult to think of any bigger security needs than those it satisfies.

Table 2: Java Card's properties.

Property	Java Card
Modularity	Static
Dynamism	No
Security	High
Resource-friendliness	Extreme
Connectivity	Native : improved Java

However, because of its severe restrictions, modularity is completely static. Furthermore, there is also a complete lack of flexibility, which clashes with our interests. Based on this, Java Card is not a

suitable solution.

3.2 OSGi

The second Java-based platform is OSGi, a module system and service platform that implements a complete and dynamic services model. With a design focus on flexibility, it was initially conceived for smaller systems. Despite that, the capabilities it offered were really appreciated by big, server applications and its design focus shifted into big resourceful systems.

As shown in Figure 2, applications can be installed, uninstalled, started and stopped without affecting the whole system. Through their entire life, applications go through a series of states depending on their situation. However, it leaves security as a non-compulsory layer.

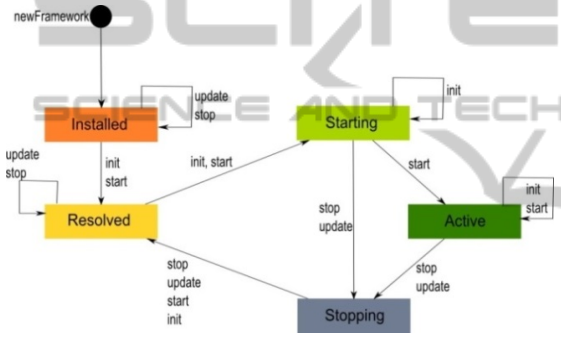

Figure 2: OSGi bundle lifecycle.

3.2.1 Architecture

The OSGi specification (The OSGi Alliance, 2011) defines a multilayer architecture. All of the layers are compulsory except for security and the architecture lies on top of a Java Virtual Machine.

The application layer defines OSGi's applications, called bundles. They are JAR files in which additional metadata is found on a manifest file. Extra resources can be added too. Bundles can offer services by using the services registry. Services, defined on their own layer, offer functionalities that can be searched by using the registry, where service providers register them. As for bundles' lifecycle, there is an API for its management, which defines all the states that they can attain in their lifetime. The lifecycle layer is composed by a total of five different states.

The module layer is the one that defines resources' encapsulation and the declaration of dependencies between bundles. Finally, the last layer deals with security. It is an optional layer that handles security aspects, by using the Java permissions system.

The main problem of this architecture is that most existing distributions are not resource-friendly. With our focus being on small devices, only small OSGi versions are suitable.

Table 3: Standard OSGi properties.

Property	OSGi
Modularity	Dynamic
Dynamism	High
Security	Optional
Resource-friendliness	No
Connectivity	Optional

3.2.2 Resource-friendly Distributions

The need for a resource-friendly OSGi distribution has already been discussed, and some solutions have been proposed. Since one of the first identified problems is that Java itself is a quite heavyweight technology, some have decided trying to move away from it. Using Java means a deep optimisation of the OSGi architecture.

Concierge (Rellermeyer & Alonso, 2007) is an OSGi R3 open-source Java implementation designed for resource-restricted environments. The focus is to optimize OSGi's design so that it can perform better on small devices, while trying to get consistent behaviour across devices.

nOSGi (Kächele et al., 2011), is a C++ native implementation of OSGi R4 running on POSIX systems. Considering that there is a lack of a proper JVM for most devices, authors decided to use C++ which is also object-oriented and widely used by those working with embedded systems (Kriens, 2010). Results are compared to those of Concierge and Equinox (Eclipse Equinox, n.d.). They were significantly better than the others.

Motivated by the same main idea as nOSGi, Apache Celix (Broekhuis, 2010) creators chose C. It is an incubating project on the Apache Incubator and thus, there is no stable release yet.

3.2.3 Conclusions about OSGi for Small Devices

OSGi for embedded devices leaves some interesting advantages. First, the fact that it has been designed for achieving high flexibility gives us a platform that works the exact way we envisioned our services module. Another important advantage is that OSGi, as a platform definition, also has a really big community, which can be a big help for developers.

Table 4: OSGI distributions' comparison.

Distribution	OSGi version	Language	Speedup	Memory usage	Security API
nOSGi	R4	C++	1	1	No
Concierge	R3	Java	2	2	Yes
Celix	R4	C	-	-	No
Felix	R4	Java	3	3	Yes

Finally, classic Java OSGi distributions get another big benefit; they can take profit from Java's good connectivity. Furthermore, non-Java distributions are OSGi-compliant, and so, even though not inheriting these benefits from Java, they should implement them on their own different way.

On the other hand, the main drawbacks are related to some of our priorities. OSGi's design is not inherently resource-friendly, but the contrary, and security is left optional.

On Table 4, a comparison between the mentioned resource-friendly distributions and Felix (Apache Felix, n.d.), a standard size distribution, can be found. The speedup and memory usage columns show the distributions ordered in terms of best performance. Since these data is taken from their respective papers, no data is given about Celix.

In conclusion, OSGi would be an acceptable solution in terms of fulfilling the properties, as long as security would be made compulsory. However, the hardware constraints put by our platform's target devices make it incompatible.

3.3 Other Java Platforms

After checking technologies based on our main needs, a suitable embedded device allowing for Java application development was sought. This search led us to ST Microelectronics' STM3220G-Java board. An even smaller device, VirtualSense, can give an idea about extremely resource-restricted environments.

3.3.1 STM32Java

STM32Java (IS2T, n.d.) is a software development kit (SDK), an integrated solution based on the MicroEJ runtime and libraries that allows the development of both Java applications and Java Platforms (JPF), which are libraries that embed some native and some Java libraries.

The structure of any application created using STM32Java consists in a C part and a Java part, the one strictly used by the developer. Following a parallel C and Java separate compilation flow, the modularity obtained during programing is lost, since the process creates a single executable file for both the application and the platform. This executable is loaded into the board by flashing it. Thus, any change requires for the platform and the application to be compiled and flashed again.

STM32Java is a good experience on terms of usability. However, no dynamic deploying or uninstalling of services can be done, which turns it into a non-viable solution for a flexible platform.

Table 5: STM32Java's properties.

Property	STM32Java
Modularity	No
Dynamism	No
Security	No
Resource-friendliness	Standard
Connectivity	Optional

3.3.2 VirtualSense

VirtualSense (Lattanzi & Bogliolo, 2012) is a wireless node designed for being used in wireless sensor networks with severe power constraints. It disposes of an on-board Darjeeling (Brouwers et al. 2009) modified 16-bit JVM, allowing for Java-developed applications to be run on it over Contiki OS (Dunkels et al. 2004).

This platform is extremely resource-restricted, having a 25MHz microcontroller unit, 16KBytes RAM and 128KB Flash. It also gives the means for multitasking and wireless connectivity, through radio. The platform also provides a remote interface allowing loading, launching, stopping and unloading applications once installed on a node.

Even if it seems to enable application provisioning over the air, no service model is provided and cooperation between applications is very restricted.

4 CONCLUSIONS AND FUTURE WORK

We should keep in mind then that our priorities are: having enough isolation and security between services, a platform where services are installed, uninstalled, started and stopped dynamically;

Table 6: Technologies' comparison.

Technology	Security	Modularity	Dynamism	Connectivity	Resource-friendliness
Java Card	High	Static	No	Native	Extreme
OSGi	Optional	Dynamic	High	Optional	Standard
STM32	No	No	No	Optional	Standard
VirtualSense	No	Dynamic	Yes	Radio	Extreme

modularity, connectivity and resource-friendliness. Table 6 shows a comparison of all the technologies in terms of these needs.

Since none of the technologies fully satisfies the needs, something new is necessary. The arrival of OSGi ME (Bottaro and Rivard, 2009), an OSGi release for embedded devices with a focus on security, could suppose the appearance of a platform fulfilling most of our needs. However, it is not available for evaluation yet and it is not an open platform. Our objective is to get the best properties while keeping the new system resource-friendly. Several solutions are being studied:

- An ad-hoc structure of **"OSGi-like services" + JVM**. That would probably be the best solution, as long as this custom structure is sufficiently modified to get the properties that standard OSGi does not enforce enough. The usage of security will have to be mandatory for achieving the necessary isolation and certification and an eye should be kept on its resource-friendliness.
- A structure consisting of a **Virtual Machine** with a set of ad-hoc **services** and **isolation** added. This would be a suitable solution in getting the necessary features if resource-restriction would make the first option impossible.
- A structure using **native services** with **added security and isolation**. Right now there are plenty of services implemented in Java. Since this is not expected to change and our platform can benefit greatly from using them, transforming them to native services would be compulsory. There is a need for a new set of **code transformation tools** to be developed to turn Java into native.

Future work will be done towards obtaining a platform similar to OSGi in terms of dynamism and modularity, but with an approach to resource-friendliness similar to that of VirtualSense and added security for its services.

REFERENCES

Apache Felix, Apache Felix. Available at: http://felix.apache.org/.

Atzori, L., Iera, A. & Morabito, G., 2010. The Internet of Things: A survey. *Computer Networks*, 54(15), pp.2787–2805.

Bottaro, A. & Rivard, F., 2009. OSGi ME - An OSGi Profile for Embedded Devices. , pp.1–16.

Broekhuis, A., 2010. Apache Celix Proposal. Available at: http://wiki.apache.org/incubator/CelixProposal/.

Brouwers, N., Langendoen, K. & Corke, P., 2009. Darjeeling , A Feature-Rich VM for the Resource Poor. *Proc. ACM Sensys*.

Dunkels, A., Grönvall, B. & Voigt, T., 2004. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*.

Hugunin, J., 1997. Python and Java : The Best of Both Worlds. *Proceedings of the 6th International Python Conference*.

IS2T, STM32Java. Available at: http://www.stm32java.com.

Kächele, S. et al., 2011. nOSGi A POSIX-Compliant Native OSGi Framework. *COMSWARE '11. ACM.*

Kriens, P., 2010. Minimal OSGi Systems. Available at: http://blog.osgi.org/2010/10/minimal-osgi-systems.html.

Lattanzi, E. & Bogliolo, A., 2012. VirtualSense: A Java-Based Open Platform for Ultra-Low-Power Wireless Sensor Nodes. *International Journal of Distributed Sensor Networks*, 2012, pp.1–16.

Rellermeyer, J. S. & Alonso, G., 2007. Concierge : A Service Platform for Resource-Constrained Devices. *ACM SIGOPS Operating Systems Review*, 41(3).

Sun Microsystems, 2008. THE JAVA CARD ™ 3 PLATFORM. *White Paper*, (August).

The Eclipse Foundation, Eclipse Equinox. Available at: http://eclipse.org/equinox.

The OSGi Alliance, OSGi. Available at: http://www.osgi.org/.

The OSGi Alliance, 2011. OSGi Service Platform Core Specification. , Release 4,(April).