RUNES: a Real Ubiquitous Network Elaboration Solution?

Paul Antonelli
M2 IFI IAM
14 av Charpentier
06100, Nice, France
+ 33 6 21 06 80 46
antonell@polytech.unice.fr

Olivier Barafani
SI5 IAM
15 av. Jean Cuméro
06130, Grasse, France
+33 6 13 58 19 92
barafani@polytech.unice.fr

Nicolas Galea
M2 IFI IAM
10 petite av. des Orangers
06100, Nice, France
+33 6 06 53 66 08
ngalea@polytech.unice.fr

Alexandre Jannotta
SI5 IAM
31 av. F. Mistral
06130 Grasse, France
+33 6 15 33 32 65
jannotta@polytech.unice.fr

ABSTRACT
A quick analysis of RUNES real capabilities to handle heterogeneous and ubiquitous network embedded systems and environment.

Categories and Subject Descriptors
D.3.2 [**JAVA, C**]: Ubiquitous computing Middleware – *middleware, sensors, network communication.*

C.2.1 [**Network Architecture and Design**]: Automatic reconfiguration.

C.2.2 [**Network Protocols**]: Protocols for heterogeneous networks – *IPv4, IPv6, MAC, gateway*.

General Terms
Design, Reliability, Documentation, Verification.

Keywords
Ubiquitous computing, middleware, heterogeneous network, components, services.

1. INTRODUCTION
There are more and more middleware for ubiquitous application development. This article is about one of them: RUNES, "Reconfigurable Ubiquitous Network Embedded Systems", a European project. Because it is one of the most advanced, we are going to analyze if its architecture is a real solution for heterogeneous and ubiquitous network embedded applications development.

2. APPLICATIONS DOMAINS
Middleware are generally designed to respond to a concrete need. RUNES is not an exception. Because ubiquitous computing is more and more widespread but on various systems and heterogeneous network, middleware are important to simplify development for concrete applications domains. RUNES is about manage some of them.

2.1 Healthcare
Cost of healthcare increases each year. Because of this, we need to find solutions more efficient that the basic system of seeing doctors only if patients have problems. The basis is so to provide healthcare services and monitoring solutions closer to patients that must run well everywhere, regardless of the network environment, to help diagnostic just in time.

Cardiac monitoring is a good example of such applications. Cardiac problem is one of the most important causes of mortality. Symptoms are many: fatigue, decreased exercise tolerance, unexplained cough, decreased food intake, delirium, abdominal symptoms… This kind of disease implies that the patient must be mobile and active to reduce the apparition of more problems. Because of this, we need a real-time monitoring system. In-home package of sensors can be easily installed: RFID tags to locate the patient in the house and evaluate mobility, weight sensors in the bathroom floor. The patient can also handle a portable monitoring system that has sound sensors to monitor patient's breath, blood pressure and pulse sensors. All of these sensors values have to be collected, here with the RUNES middleware. Critical information has to be sent to the local clinic with secured connection. Cardiac specialist can give better treatment regime that was adjusted and specify more appropriate activities to the patient. The patient can receive all these information and can communicate with specialist with an in-home interface, also handle by the RUNES middleware.

Today, there are no unified solutions to develop concrete global healthcare application like this example. RUNES directly compete with middleware like Context Toolkit but has the advantage of offering a hardware abstract layer for sensors.

However, it appears that RUNES maybe is a too much complex firmware. In the case of a simple healthcare device that monitor blood pressure and pulsation, it is maybe more easy to develop a simple application for it.

2.2 Automotive safety and security
Modern vehicles have already a lot of embedded electronic sensors and actuator that have to be driven by more and more software. Because of increasing difficulties to factorize code for this, even for car models of the same car maker, the need for a reliable middleware is important.

Assisting drive is a key feature for enhance safety and security on cars. Many sensors like ABS sensor, wheel pressure, and engine various sensors can help, with a smart monitor software controller, to have safer driving conditions.

Such applications are real stakes as great car makers, like BMW with "ConnectedDrive", or the "CAR 2 CAR" consortium (Audi, BMW, DaimlerChrysler, Fiat, Renault, Volkswagen…) are working on them. The need for unifying theses applications is important for such consortium.

However, RUNES, does not handle concrete network issues like connections and disconnections on the fly between cars. This have to be solved by car makers.

2.3 Factory automation

Today industries search to produce more and more at lower and lower costs. Mass production needs also flexibility to reach customization level that users want. Waste and time of operations have to be reduced. For this purpose, manufacturing facilities have to be monitored to control process and improve manufacture performance.

Wine production, which represents a global market of 150 billions €, is a good example for such applications that can handle RUNES. Quality of wine is important and depends on production and distribution chains. Currently, tracking and monitoring systems exist but with no seamless link between these 2 chains and information are lost. "Eurojenet Wine" for example can monitor the production but not the distribution. Unified monitoring system for such heterogeneous environment is the key to provide better quality of wine. For example, light, moisture, heat measurement for agriculture, grape acquisition, barrel management, laboratory controls can be monitored in the production chain while temperature, humidity, pH, CO2 measurement for bottles or tanks can be reviewed in the distribution chain. Vehicles can also be managed to provide cooperative distribution chain with multiples transportation ways.

However, replace such existing monitoring application by RUNES for wine production chain may be complex and expensive for adding distribution chain monitoring. It appears that RUNES does not offer any solutions yet to integrate itself with other best known middleware like OSGi, ContextToolKit, Fractal…

2.4 In-home safety and security

Enhancing home safety and security against thieves for regular users is a great stake that represents an important market. Sensors must be added at home and applications have to be easy to develop. RUNES is designed to handle multiples sensors or devices on a heterogeneous network system.

As we live older, old people safety and security is very important. Multiple sensors can be installed to send their information to a monitoring system handle by RUNES that can help healthcare specialists to detect abnormal behavior of old people. Currently, CSTB "GERHome", a well advanced project, uses OSGi and has lead to concrete sensors development.

3. OPERATING PRINCIPLES – THE TUNNEL FIRE SCENARIO

These applications are some examples that can be handled by RUNES. However, the main scenario studied for this middleware is a tunnel fire. In 2012, on a busy day, in a old tunnel, several vehicles, including a tanker loaded with vegetable oil, have a collision. The oil leak over the road and begin to get in fire. The scenario shows all the issues that RUNES have to solve as it combines an urgency state, emergency coordination, healthcare monitoring (2.1), car communications (2.2) and heterogeneous systems (2.3).

3.1 Network Adaptation

In this section we will look at the network where RUNES is supposed to operate. In fact we are going to see that RUNES is effective but only on simple networks.

3.1.1 Network Architecture

RUNES uses a network divided in 3 layers. The first one is a primitive sensor node. It is used to collect data on the field and it is relied to a gateway. The second layer is a sensor routing node. The previous node uses it as a gateway. The last one is a pure routing node. It has at least 2 network interfaces. One is to connect sensor routing node each other, and another is to rely rooting node.

An advantage of this architecture is to have a simple network organization. And it can simplify applications deployment. But in other side this can be too simple to models the heterogeneity of existing networks.

3.1.2 Network Movement Reaction

Even if RUNES uses simple network architecture, there is an issue that it does not completely solve. This is the network movement reaction.

This issue can be encountered in two main situations. The first one is when we have a broken connection on the network. The second is when a sub-system is moving (like a car along the tunnel).

The authors of RUNES study some solutions but do not arrive to a final one. Currently they don't explain how the network must react, they don't say neither how application was notify. In fact they try to find a solution without consider existing systems (like mobile phone) where this issue is still unsolved.

In the Tunnel fire scenario, they just give the example of a broken link. This link is just replaced by radio-frequency communication. But we don't have the process way when this radio-frequency communication doesn't exist.

3.1.3 Organization Sample

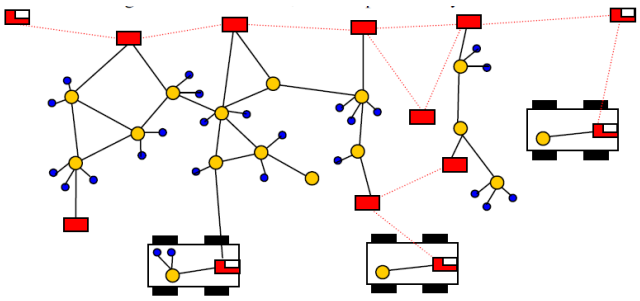

Figure 1. Network Field Sample

In the Figure 1, we have an example of the network that could be deploy in the tunnel. We find the 3 different types of nodes (Red for the routing node, Yellow for the Sensor routing node, and Blue for the Sensor node).

In the first look we can easily understand that this architecture can be efficient.

In the second look we see that it is a RUNES network designed for RUNES applications. But today we know there are many different networks and many existing application. And in fact we can understand that RUNES just search to replace existing system

to make it better: RUNES is like a reinvention of the wheel. But if we look at the past, experience says that is not always efficient.

3.1.4 Network Issues

After these sections we must add classic network problems.

The first and main problem is security. In RUNES specification there is no given process to protect data.

The second problem is localization. It is explained that RUNES must be auto configurable, but when we had a sensor to the network, it is impossible to know where it is without manual configuration. So if we have no static localization, we can't have localization for a mobile system.

The last problem is network saturation. With many sensors, we have many communications. And if we use RF communication we can easily reach the maximum bandwidth. That can provide a heavy load of information treatment. So to not have saturation it must implement data filter and communication prediction.

3.2 Component architecture

You can find the component model in three RUNES implementations in C/Linux, C/Contiki and in Java. As the C/Contiki version has few limitations because of the OS, and the C/Linux does not represent the exact architecture because of the procedural aspect of C, we will explore opportunities available in java.

3.2.1 Presentation of the model

The connection between components is done using interfaces and receptacles. A receptacle represents the component that can provide a service while an interface is a client of it.

Figure 2. A pictorial representation of a component

A component has the possibility to implement several interfaces and containers. Complex architectures may be realized from this; moreover it is possible to produce components that are composite, i.e. composed by other components. It can, within an application, define precise component features, for example in the case of the tunnel fire scenario, we can represent the firefighter entity like a component composite that is build with other component like his helmet or his leather.

3.2.2 Connection between components

The connection between the two entities is done through a "Connector" which is a particular component. This one provides properties of introspection to architecture, possibilities of adding interceptors on receptacles and interfaces. We can also specify pre and post-conditions at the invocation of components for adding checks or additional security to architecture.

Figure 3. Representation of the connector component

3.2.3 Capacity of reflexivity

In RUNES architecture, there is a component called "Capsule". In this, we can deploy other components with possibilities of reflectivity are offered. We can follow the life cycle of the component, but also instantiate other components. The "Capsule" has the feature to keep interfaces and receptacles of each component instantiated, so it is easy to disconnect a component and replace it with another or search component attributes.

Possibilities of reflexivity and introspection are important for the model including the target. Considering the case of a server that accepts a certain amount of connection, the component connections manager may for example be "unplugged" if the maximum number of users is reached.

In fact, RUNES default – and only! – implementation can only load components based on the exact string pattern of their class names. In the case of the fire tunnel scenario, this can be an issue if we want to search and load in the architecture any component that provides temperature sensors.

3.2.4 Comparison with an existing model

3.2.4.1 Presentation of Fractal component

The characteristics of RUNES can be found in a component model such as "Fractal", the differences are in the implementation. Fractal is a modular, extensible and programming language agnostic component model that can be used to design, implement, deploy and reconfigure systems and applications, from operating systems to middleware platforms and to graphical user interfaces. A component can be represented with some client and server interfaces ("receptacles" in RUNES model), some controller, interceptor and a membrane.

3.2.4.2 Differences between the two models

In fractal, each component has a "membrane", which adds reflexivity properties. The life cycle is managed by this membrane.

In the case of the tunnel fire scenario, this difference is important. In fact, if a RUNES capsule system host is broken by a fuel tank explosion, all components managed by the capsule will disappear, even if components are not in this damaged system. With Fractal, each component is independent. Doing the same thing on RUNES implies that each component is implemented along with a capsule, making more complexity to the development as it is done automatically in Fractal.

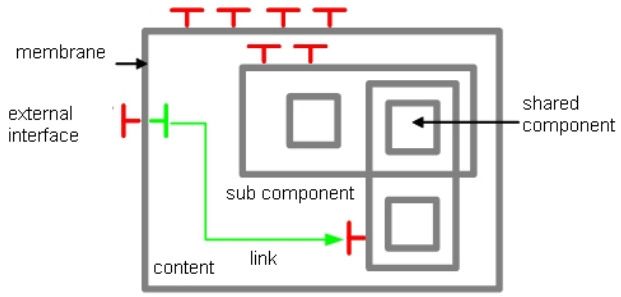

Figure 4. Representation of a Fractal component

However, in RUNES we have the opportunity to choose the components that will have a configuration at runtime with the "Capsule", while in Fractal all the components use reflexivity. This is significant because through the "Capsule" if we desire to obtain such attributes of a component, the components being stored in a Hashtable, it is relatively easy to find. In Fractal, for against, it is not uncommon to have to go up among all interfaces from the component "root" to find the desired one.

Fractal has a language of architecture definition, to "bind" components easily and provides a degree of visibility compared to RUNES, in where we have to specify all binding by hands in a capsule.

3.3 Component based middleware services

We will now describe the component framework abstraction used to build RUNES' component based middleware services. We will follow by a short summary and analysis of the main services that may be used in RUNES' reference scenario.

3.3.1 The component framework abstraction

3.3.1.1 Overview

A component framework is an encapsulated composition of components that addresses some focused area of functionality. It must allow the acceptation of additional components known as "run-time plug-ins" that may somehow modify or extend the component framework's (CF) behaviour. Notice that CFs are components themselves. Because of this, a CF (containing other CFs, in a recursive manner) built to provide a set of functionalities is finally called "service". In practical terms, the goal of CFs is to help developers in composing components together according to a set of constraints (e.g., defined in a specific language such as OCL).

For instance, a component framework can represent a network stack, and hence require (at the very minimum) the presence of a component implementing a "MAC" interface as well as a component implementing a "routing" interface. A constraint can be defined over this grouping such that the routing component can be stacked on top of the MAC component, but not vice-versa. Furthermore, a plug-in component implementing any kind of additional functionality on top of the routing component can be dynamically added to the CF if it meets the set of constraints present at the time it tries to enter the CF.

3.3.1.2 Benefits

The benefits of CFs are various. Firstly, they provide intermediate abstractions between components and whole systems, acting as a scoping mechanism. Therefore, they generally increase understand ability and maintainability of systems. Secondly, they simplify component development and assembly through design reuse and guidance to developers. Finally, they enable the use of lightweight components (plug-ins) that can be linked by assuming they share CF-specific state and services.

However, by adding more and more CF services, increasing the size of the "stack" used by this final component and thus its own size, this could lead to difficulties in the understanding or the analysis of what happens precisely at a given time in such component.

3.3.2 RUNES' main CF based services

Having described the component model (middleware kernel), and the CF abstraction, we will now try to see how this can lead to the support of various middleware services — i.e., services that can underpin application scenarios such as "the tunnel fire".

3.3.2.1 Network and Interaction services

The network services CF supports an extensible set of plug-in network communication services and provides a uniform set of APIs to these. It accommodates both ad-hoc networking and infrastructure-based networking. The interaction services CF supports an extensible set application-level "interaction paradigms" that may be layered on top of the Network Services. Examples of plug-ins accepted by this CF include: tuple spaces, reliable multicast, publish-subscribe and event notification, remote procedure call, etc. Many such plug-ins can coexist depending on application needs. The use of both of these services will allow the rescue team to get data from the whole remaining devices connected to a reachable node of the system.

3.3.2.2 Advertising and discovery services

One of the pivotal requirements of ubiquitous computing is the ability to discover devices and services that are offered in the environment. Using this service, the devices can potentially connect to different types of networks, either concurrently or at different times, with different hardware. Moreover, it can support many different protocols for advertising and discovery. In the emergency scenario of the disaster in the tunnel, as the rescue team is not aware of the configuration of the network and the measurements offered by devices installed in the tunnel (or worn by people), the advertising and discovery framework can be used to offer an up to date image of which devices are available and what services they offer.

3.3.2.3 What if some services lack?

For instance, despite its key role in such systems, it seems that there is no "Security services CF" actually available on RUNES Project, in order to protect communications of all the devices. It must probably be still in development and it may also be the same for other services or plug-ins. In the case of the lack of a CF-specific plug-in of a given CF service, the developer will have to build by his own the needed functionality that will finally be added to the existing CF. However, building from scratch a new CF able to fulfil a set of services and to communicate with

existing ones, seems to be much more complex and tend to be impossible for a non-member of the RUNES' project team.

4. CONCLUSION

RUNES is a component-based middleware for ubiquitous applications over heterogeneous networks. Despite of the fact that it is a great European project with a lot of ambition, it appears that this middleware does not solve important issues like concrete network installation and reconfiguration, even if this network is a RUNES' dedicated one. The other issue is also that it cannot natively interact with existing middleware and solutions like OSGi or Context Toolkit.

In fact, we have to consider the size of the project and the number of participant as it is a European project with universities and industries. RUNES is promising big not finished project that can evolve and solve many of its issues in the future.

5. ACKNOWLEDGMENTS

Our thanks to Polytech' Nice – Sophia for allowing us to study middleware for ubiquitous computing.

6. REFERENCES

[1] University College London, "RUNES Reconfigurable Ubiquitous Networked Embedded Systems", 2004-2006:
http://www.ist-runes.org/
http://runesmw.sourceforge.net/contiki.html
http://www.ist-runes.org/potential_apps.html
http://www.ist-runes.org/scenario.html

[2] Domonkos Asztalos (ETH), Karen Lawson (Kodak), Stephen Hailes (UCL), Lesley Hanna (Sira), Ingolf Krüger (UCSD) , "Application Scenario Building/Definition" , 31/03/2005:
http://www.ist-runes.org/docs/deliverables/D2_01.pdf

[3] Eurojenet, 2006:
http://www.eurojenet.com/

[4] OSGi Alliance, OSGi middleware, 2009:
http://www.osgi.org/

[5] Anind K. Dey, Alan Newberger, University of Berkeley, Context Toolkit, 2003:
http://contexttoolkit.sourceforge.net/

[6] CSTB, projet GERHOME, 2008:
http://gerhome.cstb.fr/

[7] Adam Dunkels (SICS), Björn Grönvall (SICS), Mattias Johansson (EAB), Karl Mayer (IABG), Frank Oldewurtel (RWTH), Ossi Raivio (RWTH), and Janne Riihijärvi (Editor, RWTH), "Review Document: Existing architectures and components", 31/12/2004:
http://www.ist-runes.org/docs/deliverables/D1_01.pdf

[8] Mattias Johansson (Editor, EAB), Domonkos Asztalos (ETH), Preliminary Verification and Validation Report, 15/10/2006:
http://www.ist-runes.org/docs/deliverables/D1_05_02.pdf

[9] E. Bruneton (France Telecom R&D), T.Coupaye (France Telecom R&D), J.B. Stefani (INRIA), The Fractal Component Model, 05/02/2004:
http://fractal.objectweb.org/specification/index.html

[10] Paolo Costa, Geoff Coulson, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco and Stefanos Zachariadis In International Journal of Wireless Information Networks, "Reconfigurable Component-based Middleware for Networked Embedded Systems". Springer. June 2007:
www.cl.cam.ac.uk/~cm542/papers/jwin.pdf

[11] Cecilia Mascolo, Stephen Hailes, Leonidas Lymberopoulos (University College London), Gian Pietro Picco, Paolo Costa (Politecnico di Milano), Gordon Blair, Paul Okanda, Thirunavukkarasu Sivaharan (University of Lancaster), Wolfgang Fritsche, Mayer Karl (Indusrieanlagen-Betriebsgesellschaft mbH), Miklós Aurél Rónai, Kristóf Fodor (Ericsson Research, Traffic Lab), Athanassios Boulis (National ICT Australia), "Survey of Middleware for Networked Embedded Systems", 07/01/2005:
www.ist-runes.org/docs/deliverables/D5_01.pdf

[12] Cecilia Mascolo, Stefanos Zachariadis(University College London), Gian Pietro Picco, Paolo Costa (Politecnico di Milano), Gordon Blair, Nelly Bencomo, Geoff Coulson, Paul Okanda, Thirunavukkarasu Sivaharan (University of Lancaster), "Runes Middleware Architecture", 30/05/2005:
www.ist-runes.org/docs/deliverables/D5_02.pdf

[13] Cecilia Mascolo, Stefanos Zachariadis (University College London),Gian Pietro Picco, Paolo Costa, Luca Mottola (Politecnico di Milano),Gordon Blair, Nelly Becomo, Paul Okanda, Thirunavukkarasu Sivaharan (University of Lancaster), "Runes Middleware Architecture Version 2", 10/11/2005:
www.ist-runes.org/docs/deliverables/D5_02_01.pdf