Management of Wireless Software Components

Fabien Romeo
LIUPPA
Université de Pau et des Pays de l'Adour
F-64000 Pau, France
+33 5 59 40 76 52
fabien.romeo@univ-pau.fr

Franck Barbier
LIUPPA
Université de Pau et des Pays de l'Adour
F-64000 Pau, France
+33 5 59 40 77 43
franck.barbier@franckbarbier.com

ABSTRACT

Component-Based Software Engineering (CBSE) is nowadays so widely recognized that its principles naturally apply to ubiquitous computing. In this scope, we discuss software components that are deployed in mobile and wireless devices.

Event though mobile and wireless devices execute programs like ordinary computers do, their limited capabilities preclude for fully controlling their embedded software components. This calls for a global management infrastructure in which wireless components are just pieces in a puzzle. This puzzle is a highly distributed application in which assemblies are made of wireless and non-wireless components.

Each component is implemented by means a UML 2-compliant state machine. A proposed Java library supports the executability of this kind of UML diagram. Management occurs through the supervision and the control of remote components through replicated components. The paper describes how the management architecture and activities may formally rely on state machines and event processing.

Keywords

Software components, management, mobile and wireless systems, UML.

1. INTRODUCTION

A major trend in mobile systems is the design of their software as an assembly of components (Java components in J2ME environments, C# components in Windows CE environments…). Components are interconnected through their interfaces while hiding their implementations in order to increase their reuse and to allow to be deployed by third parties. Deployment occurs on various devices such as mobile phones, PDA, set-top boxes, smart cards and so on. Owing to the fact that deployment environments are different from development environments, abnormal behaviors and/or misuses occur and, consequently, call for remote administration and supervision [2], [11].

The purpose of this paper is to present an administration system for wireless software components. Section 2 describes the global view of our system while we detail in Section 3 how we may design components behaviors so that models (*i.e.*, state machines) are directly accessible at runtime and serve as supports for management actions, for instance, roll backs in case of failures.

2. GLOBAL SYSTEM

Figure 1 is a global view of the system we propose for managing wireless software components. Administration focuses on the configuration of components and their possible dynamic reconfiguration in order to ensure a real control of wireless devices: reaching given states through "reset" actions for instance.

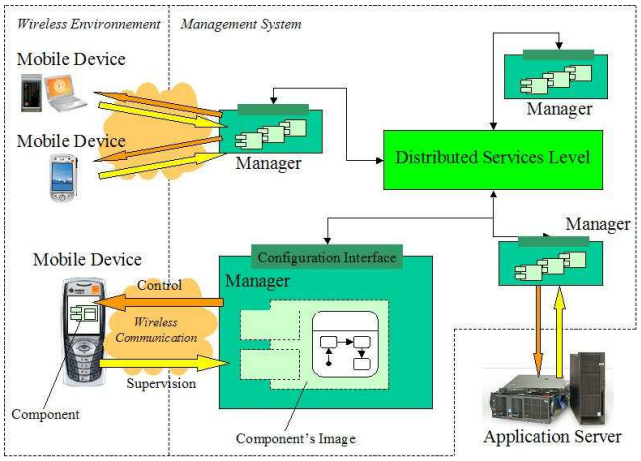

Figure 1. Management System of Wireless Software Components

Supervision and control are dual activities involved in management. While control implies that the manager changes the state of the managed component, supervision consists for the manager in acquiring information on the actual state of the managed component. Considering that these activities have to be realized by means of wireless communication and in relation with highly constrained devices, our intention is to minimize overheads generated and sustained by our management system on the mobile side. The quality of service must not be damaged by the execution of administration/supervision functions. We on purpose create images of the managed components in the management system. The manager directly accesses these image which acts as a proxy for the wireless components.

On the management side, due to the existence of large and complex behaviors, we represent component behaviors as UML statecharts [8] and implement them with an associated Java library that in essence supports executability for UML statemachine diagrams. By plugging into components such executable statemachines, we forward wireless components' events to their images or replications on the management side. By this means, the view of the application behavior remains up to date.

The behavior management mechanism based on events is detailed later. Figure 2 depicts its implementation on a wireless environment where a J2ME [18] wireless component called A (a simple gearbox for illustration purposes) communicates by means of message (B) with its J2SE replicated component (C). The messaging system that links the world of J2ME to the world of J2SE, is realized with the WMA technology [17].

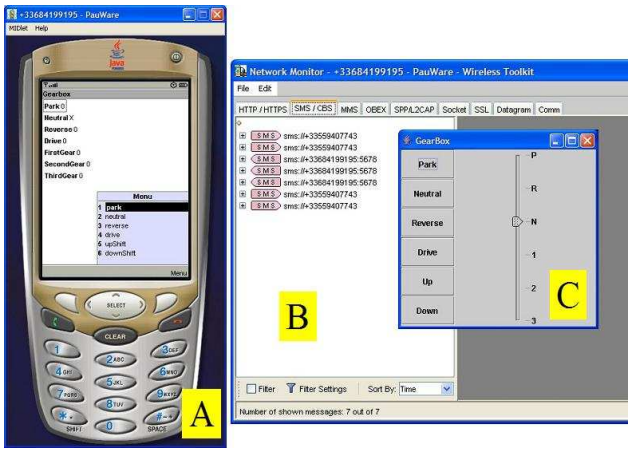

Figure 2. Java Implementation of a Management System for Wireless Software Components

The great advantage and the uniqueness of our approach is the fact that we take advantage of the expressiveness of statecharts to supervise components. Contrary to other supervision systems that represent the state of the system with the value of some variables or attributes, we have here access to **abstract**, **hierarchical** and **concurrent** states of statecharts defined when modeling component behaviors. This gives an accurate view of the global state of the application and allows to envisage new administration policies based on these logical states. Furthermore, the use of statecharts offers new capabilities in the control activity. By directly accessing statemachines of running components, we can constrain their behavior and completely manage the application.

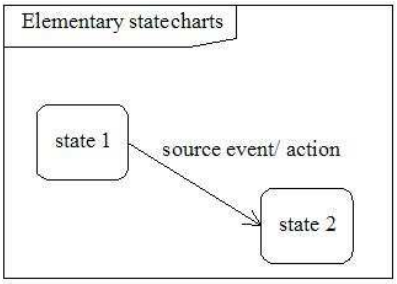

Figure 3. Elementary Statecharts

The global architecture of our administration system is based on a standard for administration called JMX (Java Management eXtensions) [13], extends the ideas of the Built-In Test (BIT) technology [3], [4] which did not initially provide any support for administration. This paper fills this gap by explaining how test code that remains in components is remotely monitored in our management infrastructure.

3. COMPONENT BEHAVIOR MANAGEMENT

In the spirit of model-driven engineering (MDE) [10], which is a recommendation of the OMG, we model the behavior of wireless components with UML executable statecharts.

Figure 3 depicts an elementary statechart diagram composed by two states, *state 1* and *state 2*, and a transition from *state 1* to *state 2* fired by an event of type *source event* which generates in reaction an action of type *action*. In the offered architecture, issues consist in keeping the statechart of the *Replicated component* "in line" with the statechart of the *Wireless component* when an event is processed within the wireless environment.

"In line" does not mean that both statecharts are always synchronized. One main advantage of Harel's Statecharts is broadcast which allows to send events without any knowledge about the receiver's status. In such a context, a basic principle is thus to notify the *Replicated component* when an event appears in the statechart of the *Wireless component*.

We however need for distinguishing between different types of events (next sections) in the sense that some types of events may be simulated on the management side instead of being systematically forwarded from the wireless environment.

Typically, a pressed button on the wireless device amounts to sending a typed event occurrence to the management side in order to capture the *Wireless component*'s behavior. In contrast, timer event services for instance, may be acquired independently by a *Wireless component* and by its image on the management side so that communication decreases. This case may thus lead to a lost of synchronization between both statecharts but without coherence lost since, as said before, any statechart is able to receive and to process any event type at any time.

We simply solve such a problem by supplying a management operation called "re-synch" which may be launched on demand (*i.e.*, user-oriented decision) and only acts on the *Replicated component*'s statechart. We ground such an approach on a method which consists in forcing the statechart of the *Wireless component* and that of the *Replicated component* so that they fall into the same states. Once again, such a method is supported by the precise and rigorous components' inside description resulting from the power of Harel's Statecharts and our associated Java library called *PauWare.Velcro*.

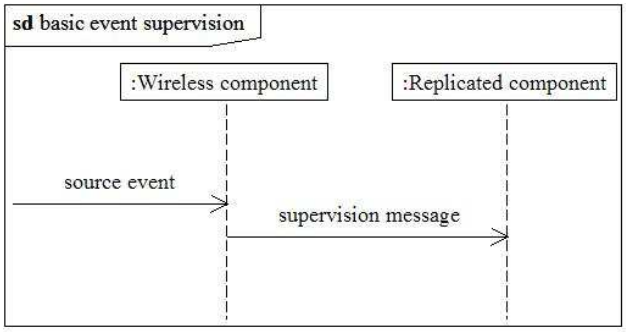

Figure 4. Basic Event Supervision

In the following sub-sections, we define the different types of events we have identified and we describe the way they are managed in our management system.

3.1 Basic Events

Definition

"A basic event is an event whose source is local to the wireless system and that trigger an internal action in the component, *i.e.* an

action that does not need any other resources than those available in the considered wireless system."

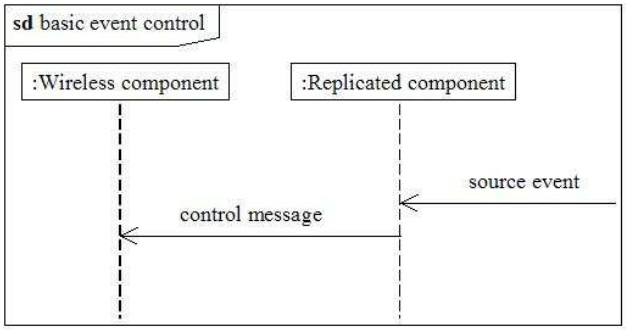

Figure 5. Basic Event Control

This is the kind of event we use in particular to tell an user interface to display new information [9] or to launch some processing in the component for instance.

The interaction diagram in Figure 4 depicts the communication between the *Wireless component* and the *Replicated component* when a *source event* is received: a message of type *supervision message* is dispatched to the *Replicated Component* which in fact synchronizes the statecharts of both components. This is how is realized the supervision of a basic event.

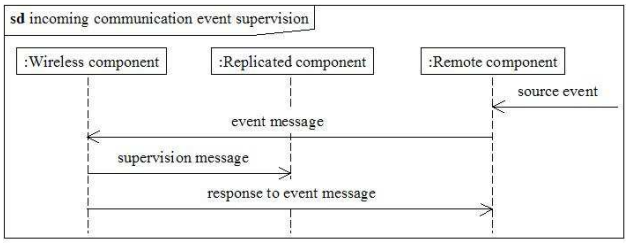

Figure 6. Incoming Communication Event Supervision

The control of a basic event is described in the diagram of Figure 5. Contrary to Figure 4, the *source event* is not received by the *Wireless Component* but the *Replicated component* does since it is the one that is in direct relation with the manager. When such an event is received, the statecharts of the *Replicated component* are modified and a *control message* is sent to the *Wireless component* which synchronizes then and performs the actions related to the triggered transition. The *control message* is interpreted differently from an event in order not to activate the supervision process which, by sending another *supervision message*, would trigger once again the event on the *Replicated component* and would consequently desynchronize the statecharts if the reaction to this event is not idempotent.

3.2 Communication Events

Definition

"A communication event is an event that involves an incoming or outgoing wireless communication between the considered wireless system and a remote component."

Software components deployed on wireless systems are highly prone to use wireless communication capabilities available in their host systems in order to realize distributed applications.

The diagrams of figures 6, 7, et 8 detail the supervision and the control of communication events. In those diagrams a third entity comes into play, the *Remote component* which is the component that establishes a wireless communication with the *Wireless component*. There is however no diagram to explain the control of an incoming communication event but there is no use of it since the *Replicated component* on which the control is initiated is a replication of the *Wireless component* and consequently has no handle on the *Remote component* which is deployed on a remote system. In order to control an incoming communication event, the *Remote component* has to be also managed by the administration system : the *Remote component* can then be seen as the *Wireless component* and this corresponds to the diagram in Figure 8.

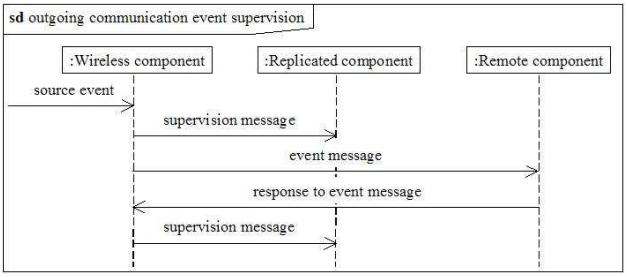

Figure 7. Outgoing Communication Event Supervision

3.3 Service Events

Definition

"A service event is a particular case of basic event, it is an event whose source comes from a service proper to the considered wireless system."

Timers, notifications of phone calls on mobile phones or system interruptions are examples of service events that one can have to handle. To access the service, the *Wireless component* requests it to the corresponding *Service component* which generates afterwards the corresponding service events.

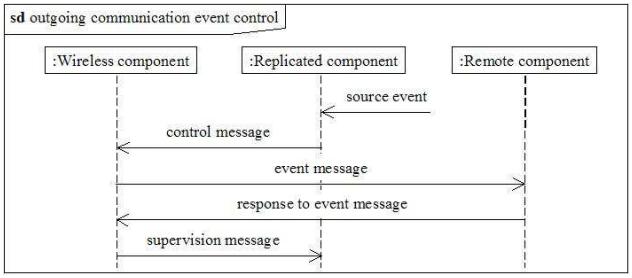

Figure 8. Outgoing Communication Event Control

Since service events are a particular case of basic events, they can be managed the same way (cf. Figure 9). However services like timers for instance generate numerous events which will be thus expensive in supervision messages. Depending on the situation, it can be interesting to replicate the service (*Replicated service*

component) on the administration systems in order to save these communications (cf. Figure 10). This can lead to statecharts divergence in the wireless component and in the replicated one,. The solution to this problem is out of the scope of this paper.

4. RELATED WORK

The closest works to ours are those of [16] which defines a middleware architecture allowing to administrate wireless devices. This architecture uses the externalization principle of state, structure and logic of the system to fully control the application at runtime. Our approach uses in fact that same principle but at a higher level of abstraction by using software components and statecharts which allows our approach to be integrated in a development method based on UML [15].

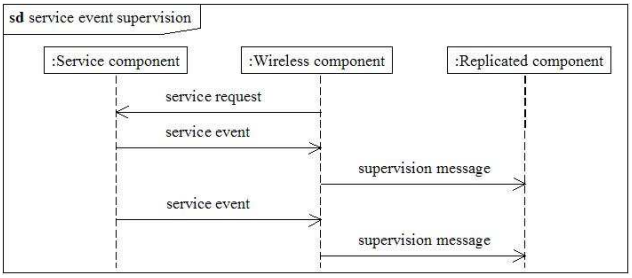

Figure 9. Service Event Supervision

In the domain of software components, the paper [7] also uses a model-based approach with Petri nets to administrate the behavior of components, but it only tackles supervision and nothing is done on control.

Other works take advantage of the use of statecharts in mobile systems but they are more interested in modeling mobility like in [1], [6], [12] or [14].

For technical aspects, the paper [5] shows the use of the JMX technology in order to administrate the life-cycle of the components of a service platform for wireless systems.

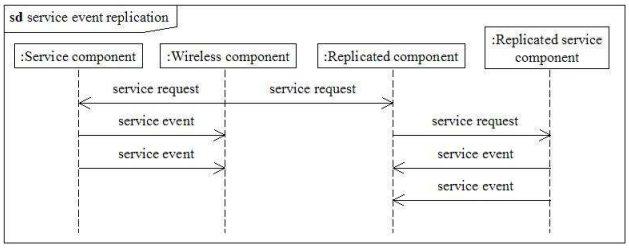

Figure 10. Service Event Replication

5. CONCLUSION

In this paper we presented an architecture to supervise and control wireless software components' behavior. The behavior of components is described with statecharts which are directly executed on the wireless system and replicated on the administration system.

This system currently works for the administration of the behavior of components individually. The objective is to extend it by taking into account the relationship between the components in order to have a finer control of the assembled application. As a matter of fact, analyze a deficiency on a component-based system or reconfigure a component system are trickier problems. This perspective of research relies on the formalization of a composition relationship based on the states of components, *i.e.* assembly rules on which administration rules themselves can rely.

6. REFERENCES

[1] Acharya, S., Mohanty, H., and Shyamasundar, R.K.: MOBI-CHARTS: A Notation to Specify Mobile Computing Applications. In: 36th HICSS, IEEE Computer Society (2003) 298

[2] Banavar, G., and Bernstein, A.: Software Infrastructure and Design Challenges for Ubiquitous Computing Applications. In: Communications of the ACM Vol. 45 No. 12, ACM Press (2002) 92–96

[3] Barbier, F., Belloir, N., and Bruel, J-M.: Incorporation of Test into Software Components. In: 2nd International Conference on COTS-Based Software Systems, LNCS 2580, Springer, Ottawa, Canada (2003) 25–35

[4] Edler, H., and Hörnstein, J.: Component+ Final Report. http://www.component-plus.org/pdf/reports/Final report 1.1.pdf (2003)

[5] Frénot, S., and Stefan, D.: Open-Service-Platform Instrumentation – JMX management over OSGi. In: 1st French-Speaking Conference on Mobility and Ubiquity Computing, ACM, Nice, France (2004) 199–202

[6] Grassi, V., Mirandola, R., and Sabetta, A.: A UML Profile to Model Mobile Systems. In: 7th International Conference, LNCS 3273, Springer, Lisbon, Portugal (2004) 128–142

[7] Grosclaude I.: Model-based Monitoring of Component-Based Software Systems. In: the 16th European Conference on Artificial Intelligence. ECAI, IOS Press, Valencia, Spain (2004) 1025–1026

[8] Harel, D.: Statecharts: a Visual Formalism for Complex Systems. In: Science of Computer Programming (1987) 231–274

[9] Horrocks, I.: Constructing the User Interface with Statecharts. Addison-Wesley Professional; 1st edition (1999)

[10] Kent, S.: Model Driven Engineering. In: 3rd International Conference on Integrated Formal Method, LNCS 2335, Springer, turku, Finland (2002) 286–298

[11] Kephart, J., and Chess, D.: The Vision of Autonomic Computing. In: Computer Magazine Vol. 36 No. 1, IEEE Computer Society (2003) 41–50

[12] Knapp, A., Merz, S., and Wirsing, M.: Refining Mobile UML State Machines. In: 10th International Conference on Algebraic Methodology and Software Technology, Springer-Verlag (2004) 274–288

[13] Kreger, H., Harold, W., and Willliamson, L.: Java and JMX, Addison Wesley (2003)

[14] Latella, D., Massink, M., Baumeister, H., and Wirsing, M.: Mobile UML Statecharts with Localities. In: Global Computing, IST/FET International Workshop,. LNCS 3267, Springer, Rovereto, Italy (2004) 34–58

[15] Mellor, S., and Balcer, M. Executable UML: A Foundation for Model Driven Architecture. Addison-Wesley (2002)

[16] Román, M., and Islam, N.: Dynamically Programmable and Reconfigurable Middleware Services. In: 5th ACM/IFIP/ USENIX International Conference on Middleware, Springer-Verlag, Toronto, Canada (2004) 372–396

[17] Siemens AG.: Java Wireless Messaging API (WMA) Specification Version 1.1. (2003)

[18] Sun Microsystems: Connected Limited Device Configuration, Specification Version 1.0a, J2ME (2000)