Context-Aware Service Composition in a Home Control Gateway

André Bottaro[1, 2], Johann Bourcier[1], Clement Escoffier[1] and Philippe Lalanda[1]

[1] *GRENOBLE UNIVERSITY*
Laboratoire LSR-IMAG
38041 Grenoble, Cedex 9, France
{firstname.name}@imag.fr

[2] *FRANCE TELECOM R&D*
28, chemin du vieux chêne,
38240 Meylan. FRANCE
andre.bottaro@orange-ftgroup.com

Abstract

One of the main technical challenges of the pervasive computing is the ability to build applications with the capacity to adapt themselves to their environment. In this paper, we present an open architecture facilitating the development of home services. Context dynamicity and service ambiguity are dynamically managed by smart composing elements. The architecture is applied and tested in an attractive home scenario.

Keywords: Service orientation, autonomic computing, service–oriented component runtime

1. Introduction

Networks of devices will soon assist us in our daily activities using the notions of goals, context and knowledge to autonomously decide on the best actions to be undertaken [1, 2]. Indeed, numerous manufacturers are already using, or planning to use, electronic devices to provide services to their customers, especially in the home context. Many houses are already covered by wireless and wired network technologies and filled with electronic devices allowing the occupants to control their environment (being for comfort or for entertainment). But several business and technical challenges complicate the manufacturers' ability to develop and manage such innovative services. To begin, the business models aren't ready; it remains unclear how to commercialize such services and electronic devices or whether manufacturers should go to the marketplace alone or with specialized partners. The second concern is technical. Putting smart services in place is a challenging task. It requires implementing the cooperation between heterogeneous devices in complex environments in which topologies, communication protocols, security policies, and such are dynamic and differ from one customer to the next.

In order to allow the development of dynamic, context-aware applications innovative architectures based on the notion of service have been recently proposed [3, 4]. Service-oriented computing (SOC) is a paradigm that defines services as fundamental elements for application design. We believe that service orientation provides the level of flexibility that is required to build pervasive applications in the home context. However, a number of challenges still have to be tackled. In particular, mechanisms are needed to build truly dynamic applications that can integrate new conditions at runtime. Most current service-oriented applications are not dynamic after the initial binding phase. It is up to the developers to deal with context evolution and dynamic (re)composition.

The purpose of this paper is to address the above mentioned limitations. It presents an open computing infrastructure for the development of pervasive applications in the home domain. This infrastructure is based on an extensible service-oriented component runtime and on local autonomic managers [5] that deal with context-aware service compositions. This work is carried out within the ANSO[1] project which brings together major European actors interested in the development of pervasive home services.

The paper is organized as follows. First we provide background information about service orientation and the kind of application we are targeting. Then we present our approach for home computing. Section 4 details our service-oriented component runtime. Section 5 presents the local autonomic managers. This is followed by an illustration in the home context and a performance evaluation. Finally we conclude by confronting this work against the state of the art and pointing out major contributions.

---

1 This work comes from collaboration in the context of the ANSO project. The authors are given in the alphabetical order. ANSO is partially supported by the French Ministry of Industry under the European ITEA program.

2. Background

2.1. Service oriented computing

The central objective of the service-oriented approach is to reduce dependencies among "software islands," where an island is typically some remote piece of functionality accessed by clients. By reducing such dependencies, each element can evolve separately, so the application is more flexible than monolithic applications. SOC is based on three actors: service providers offering services, service consumers using services and a service broker containing references to available services specifications. Three kinds of interactions are defined among these three actors: service publication between the provider and the broker to offer services for use, service discovery between the consumer and broker to find desired services, and service invocation consumers and providers to actually use the service. To design complex service-oriented applications, it is necessary to compose services, which means that providers may require other services to provide their own service.

From these concepts, SOC applications can exhibit interesting characteristics, such as:

- Loose coupling: a consumer does not need to know anything about the service implementation.
- Late binding: a consumer uses a broker to find desired services at runtime.
- Dynamic resilience: service rebinding may occur at any time. A service consumer cannot rely on the same service implementation being returned by the broker between invocations.
- Location transparency: providers and consumers are oblivious to the underlying communication infrastructure (e.g., local versus remote, specific protocols, etc.) [6].

Service orientation allows the development of modular and inherently dynamic applications. It is however generally based on low level techniques and requires a deep expertise on the programmer side. In addition, developers need to manage service dynamism within the business logic. Developers also need to manage aspects related to context-awareness. In this paper, we propose to use an extensible service-oriented component runtime to develop pervasive application. This component runtime allow the separation between the application logic and the context-aware service interactions.

2.2. Home applications requirements

This section presents an example illustrating the main requirements of home computing. The ambient messaging application presented hereafter highlights the needs in terms of dynamicity and service selection. It will be also used at the end of this paper to illustrate our approach.

The purpose of the ambient messaging application that we are investigating with France Telecom is to display information and services to the end user on the most appropriate rendering device. For instance, when the home inhabitant is watching TV, the system may direct any communications on the TV by displaying a message on the TV screen. If the user is having a nap on the couch, the state of the communications is viewed through colours and brightness on a lamp emitting a low-level light. Similarly, when he is in the kitchen, he may listen to the communications through a text-to-speech system if no display is available. If he comes at his desk, the communications are set in usual Instant Messaging user interfaces on the computer.

It clearly appears that, depending on a dynamic context, the ambient messaging application has to select and use the best services to assess the situation and to display information. The home context consists of (i) the user context made of his location, preferences and activity, (ii) the device context made of its location and its capacities, (iii) and the external physical context such as place properties (weather parameters, physical location, etc.) [7].

Home devices and services connect and quit the home network at any time. This is due to various reasons including device management operations, mobility, power saving measures, etc. For example, smart phones and PDA are connected to the home network as long as they remain physically in the home. Service availability dynamic detection has to be automated so that applications can benefit from it. A final point that we would like to mention is referred to as service ambiguity. Service ambiguity occurs when several service providers meet a same requirement. The system faces this problem when several services are available with the adequate contextual properties at runtime. It leads to composition unpredictability: the system is then given to using distinct service providers in identical situations. Service selection mechanisms must be able to precisely select the best set of resources among the available ones.

3. Our approach

3.1. Overall architecture

As presented in [6, 8, 9], we are working on an open service-oriented computing architecture for the home. This architecture comprises network-enabled devices and computing platforms connected through field buses.

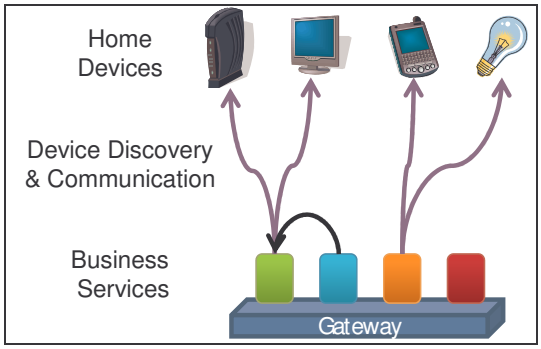

Figure 1. Platform-centric architecture

Devices are pervasive elements integrated in the house (screens, loud speakers, controllable shutters or heaters for instance) providing basic services to sense and act upon the environment. Many devices are today implemented as service providers and requesters using technologies like UPnP or IGRS for instance – see www.upnp.org and www.igrs.org). The smaller devices, in terms of computing resources, communicate through *ad-hoc* wired or wireless protocols. Proxies have to be created on a service platform to make them visible as services. Computing platforms often play the role of network gateways. Such gateways are already present in many houses (telecommunication Internet gateways, TV-connected set-top-boxes or utility service gateways) and it is likely that future homes will host several, heterogeneous such computing platforms. Gateways provide the resources needed to run higher level services making use of the connected devices.

3.2. Gateway architecture

This gateway will be able to run high level services and applications. As presented in the previous section, there is a need for building context aware applications. To enable the construction of such application, we have developed a computing framework, depicted in Figure 2, allowing context-aware service composition.

This framework is based on the following elements:

- A Service-Oriented Component Runtime providing the underlying execution framework. This framework, based on the notion of container, is extensible. A container encapsulates a business service and contains an extensible set of handlers that can be called whenever the business service is invoked.
- A Context Service aggregating different context sources. This service provides all the necessary information regarding the current execution context. The framework provides an API to access the context. In this work, we focus the attention on the whole adaptation process, and we rely on existing context monitoring systems [10, 11]. In our approach, we assume that a service representing the context is present, and we use well known ontology (common vocabulary) [12] to interact with this service.
- A set of autonomic handlers that can be attached to business services. Handlers make use of the context service to decide whether dynamic re-composition is needed. In this case, the handlers are in charge of selecting and binding the most appropriate services.

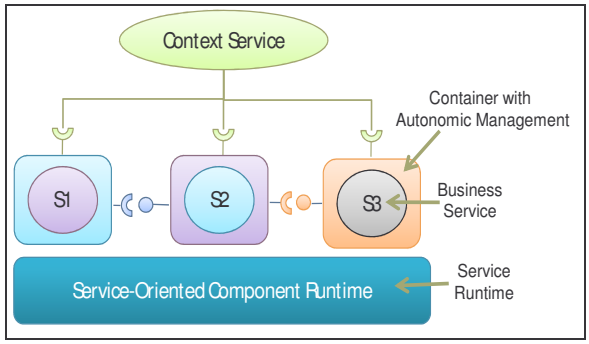

Figure 2. Gateway computing framework.

We believe that this architecture enables the construction of pervasive applications. Building such application on top of a service-oriented component model makes possible the externalization of the dynamic context execution management. The application logic code is not polluted with service interactions. Thus coding a pervasive application becomes possible to every application developers. The following section presents in more details the different elements of this architecture.

4. Autonomic computing framework

4.1. Service-Oriented Component Runtime

As introduced in [13], service-oriented component runtime help developers to build SOC applications.

The motivation to use components to implement service specifications emerges from the need to separate SOC mechanisms from the logical code implementing the service behaviour. The desire is to delegate SOC mechanisms to a component container, which will interact with the service broker at run time to find available services and to publish the component's provided services.

In [13], the general principles of a service-oriented component model were introduced, which are:

- A service is provided functionality.
- A service is characterized by a service specification, which describes some combination of a service's syntax, behavior, and semantics as well as dependencies on other services.
- Components implement service specifications, which may exhibit implementation-specific dependencies on services.
- The service-oriented interaction pattern is used to resolve service dependencies at run time.
- Compositions are described in terms of service specifications.
- Service specifications provide a basis for substitutability.

IPOJO [14] is a prototype of a service-oriented component framework. One of the main goals of iPOJO is to keep service-oriented component development as simple as possible, which means keeping the component as close to a "plain old Java object" (POJO) as possible. The code of a component should focus on business logic, not on SOC mechanisms or non-functional requirements. To reach this goal, iPOJO provides a component container that manages all SOC aspects, such as service publication, service object creation, and required service discovery and selection. Moreover, iPOJO containers are extensible. Indeed, service interactions are not the only non-functional preoccupation used in pervasive applications. Pervasive applications can need non-functional concerns as persistence, security, logging, etc.

IPOJO is implemented on top of OSGi [15]. The OSGi service platform defines a framework to dynamically deploy services in a centralized environment. The OSGi framework automatically manages aspects of local service deployment, such as Java package dependency resolution, but leaves service dependency management as a manual task for component developers. We choose the OSGi service platform for three main reasons:

- It is service-oriented,
- It is dynamic,
- It is applicable to a large range of domains from mobile phones to application servers.

IPOJO is named after the phrase "injected POJO", since the general approach of iPOJO is to inject POJOs with handlers to manage non-functional behavior. Indeed, POJO classes are manipulated (byte code injection) to become iPOJO. Based on this injection mechanism, iPOJO manages all service interactions and injects selected service providers inside POJO fields. Moreover, iPOJO provides an extensibility mechanism allowing the development of handlers separately from iPOJO core. A handler manages a non functional requirement, and is plugged automatically at runtime on the component container.

IPOJO is hosted as a subproject of the Apache Felix incubator project.

4.2. Autonomic handlers

4.2.1. Principles

To automate the management of the dynamic execution context, we propose to enhance the component containers with a context-aware manager. The role of this autonomic manager is to monitor the execution context and to react on the component by changing its bindings or acting on its lifecycle. In our vision, a manager is developed in a generic way and is driven at runtime by high-level goals and policies.

The decision-making process of a context-aware manager is presented in Figure 3. This process is composed of three main parts, detailed in the following sections, and is configured by a service binding policy. This policy contains service queries, expressed with context properties. The first step of the decision making process aims to create a system representation. To achieve this goal, the autonomic manager senses the environment. When the system representation is updated, the autonomic manager analyses this representation and produces an action plan containing the set of actions to undertake. The execution of this plan is made in a last step called reaction.

We have implemented the proposed context-aware service composition manager as an iPOJO handler. Our handler is configured inside the component description. This configuration contains the service binding policy describing for each service requirement the contextual queries and a ranking policy. An example of such policy is presented in section 5.1.

4.2.2. Monitoring

The monitoring part of an autonomic manager listens to context events and updates the system representation. In order to listen only to relevant events impacting the system representation, an autonomic manager extracts the relevant context information from the binding policy. Then, the autonomic manager can filter context event to select only needed changes.

Pervasive Applications, based on services, must also handle dynamic service availability: service publications, modifications and departures can occur frequently. Tracking services in pervasive applications consists of dynamically maintaining the list of the available service providers satisfying service requirements. The tracking mechanisms are commonly specified in service middlewares through active and passive discovery concepts. Active discovery enables the component to send an active request at starting time in order to initialize the list of available services. Active requests are also relevant as soon as the requirements are modified in order to refresh the list. Passive discovery consists in the update of this specific service list in reaction to service events generated by the underlying system mechanism whenever a service is published, modified or un-published. By listening to these events, an autonomic manager updates the system representation to maintain the set of consistent service providers.

4.2.3. Situation analysis

When the system representation is updated (and only in this case), the analytic phase of the autonomic management begins. This phase aims to create the list of actions to be undertaken to guarantee service continuity. Figure 3 shows the analysis process of our managers. The whole process is driven by a representation of the underlying system. On this representation, the autonomic manager can select the best service to be bound with the managed element.

The diagnostic begins by confronting each requirement expressed in the binding policy to the list of available service providers. If at least one mandatory requirement is not fulfilled, the manager plans to invalidate the component and to unbind every used provider. Else, the manager plans to validate the component if it is not already valid. Then if a requirement has several suitable providers, the autonomic manager ranks them according to a ranking policy to choose the best one. Precisely, the manager calls a ranking method to sort acceptable services in a total order. The ranking method is contained inside the component implementation class. It allows developers to program a complex ranking method. Then it plans to re-compose services, i.e. unbind the no more used providers and bind the new providers.

Our requirements are expressed in contextualized terms. The evaluation of requirements aims to dynamically compare queries against service provider properties.

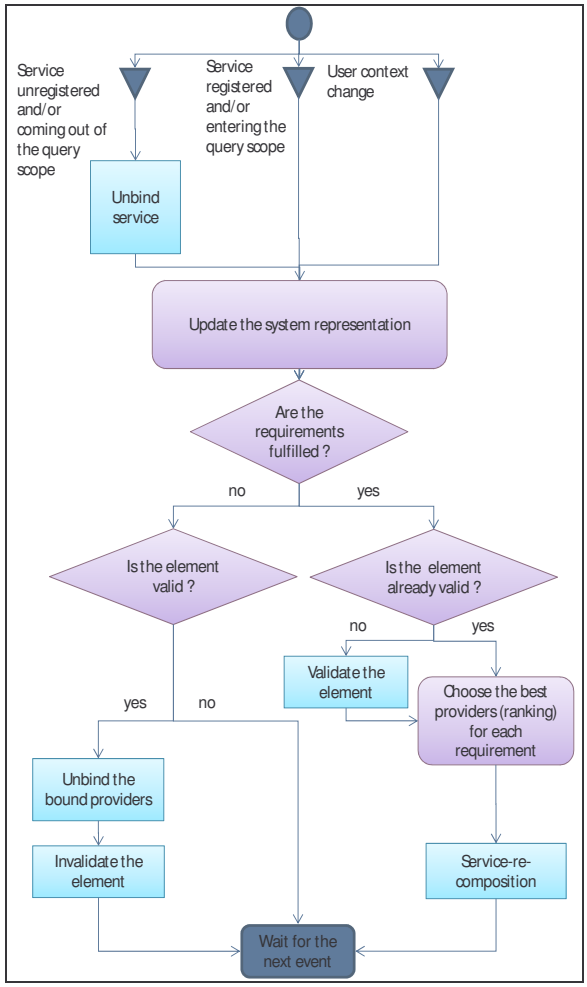

Figure 3. Decision-making process.

4.2.4. Reaction

According to the action plan, the autonomic manager needs to validate or invalidate the managed element and / or re-configure service bindings.

As depicted on the Figure 4, validating a component implies the publication of provided services and the announcement of the new services to be discoverable. Then, other components could use provided services. Invalidating an element implies the reverse action. By un-publishing services and by announcing their departure, every element using these services and

associated resources must release them. The provided services are no more discoverable.

The autonomic manager can decide to recompose service binding. By changing the binding, the manager changes the service provider used by the component. The manager must release the no more used service providers and bind the non-already bound ones. If a selected provider is no more available, i.e. if it leaves during the decision-making process, the element is invalidated and waits for a new available provider.

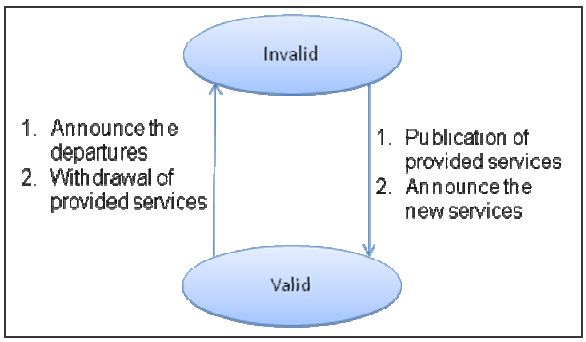

Figure 4. Autonomic Manager Action on validation and invalidation

5. Application

5.1. Ambient instant messaging

The scenario described in section 2 is implemented above our architecture. The application is designed as depicted in Figure 5. A context service gathers all the relevant information coming from sensors in order to assess:

- the location and the user willingness to communicate of the user. User willingness to communicate is key dynamic information to be taken into account by the system. Unfortunately, it is hard to measure this parameter. This criterion is linked to unfaithful information such as user activity and user preferences [16].
- the location of every mobile device (see the PDA description).

The Instant Messaging (IM) client is divided into 4 types of components. The IM application is the central component that is connected to the IM server on the Internet through out-of-band protocols (here Jabber/XMPP protocols). This component runs the IM application logic. This component requires the best IM User Interface (UI) service to diffuse the message.

Our autonomic manager is attached to the IM application component. It dynamically reacts to contextual information coming from the context server in order to select the adequate IM UI provider.

The scenario involves several devices. These devices are instances of the same UI component as depicted in the XML metadata file below. The file reflects the following description of the devices:

- A living room television with an attractive lighted keyboard in the lounge. User willingness is expected to be high.
- An ambient lamp able to convey information thanks to the variation of the brightness, the pulse frequency, and the colour of the light. A low user willingness to communicate is enough to use this incomplete user interface.
- Loud speakers with Text-To-Speech software in the kitchen and in the living room. User willingness is set to "medium" since listening to audio messages ask for some user attention but since there are no speech recognition, the user is not expected to answer to the messages.
- In the bedroom, a computer with a standard and complete user interface. User willingness is expected to be high.
- A wifi PDA with a standard and complete user interface. Its location is dynamically set. User willingness is expected to be high. User rating is set to "low" since the user would prefer to chat with any other device which is not as constrained as the PDA.

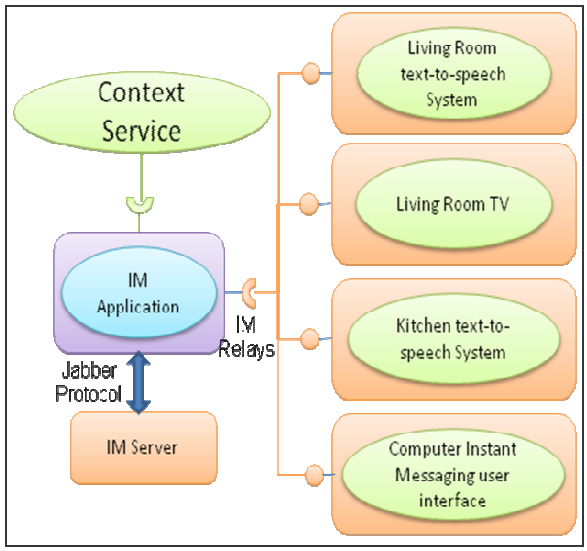

Figure 5. Architecture of the AIM application.

The following snippet shows the binding policy of the IM Application. First, based on iPOJO field

injection mechanism, a service requirement must contain the field in which the selected provider will be injected. Then the query is expressed in terms of contextual variables: the location of John. To finish, a ranking method is set to sort providers if several suit the filter. Inside the IM Application code, the developer can directly use the "relay" field. This field will always target the best user interface. If there is no suited provider, the dependency will becomes invalid, and the component will be stopped until a suited service provider appears. During this period of times John will not be reachable for other users. This is a quite simple policy, but we are currently working on how to express more complex policies, such as taking the closer media in the home if there is no media in the desired room.

```
<ipojo
xmlns:cadep="cadep.CadepHandler" >
 <component className="aim.App">
  <cadep:dependency
    field="relay"
    filter=
     "(location=${Context:john.location})"
    ranking-method="rankRelays"
   />
 </component>
 <instance component="aim.App"
name="IMApplication"/>
</ipojo>
```

Figure 6 : AIM Application metadata

5.2. Evaluation

To test our approach, we set up a scenario using the ambient instant messaging application described below:

1. John enters the home and first goes in the lounge to have a nap on the couch. A control point is attached to John. John willingness to communicate is low. The ambient lamp is chosen by the system and conveys light information.

2. John wakes up after 20 minutes. He goes to the kitchen to eat something and drink a soda. John hears the messages through the text-to-speech system.

3. John wants to answer to an important message of his friend who is chatting. He returns to the lounge and sits in front of the TV. John willingness is high, the TV is chosen.

4. During the active chat with his friend, he realizes that he misses information on the Internet. So he goes to the bedroom to use the computer. The computer is connected to the IM system.

5. He goes out of the home. He continues to read his messages on his PDA.

We compare two ambient instant messaging applications with the same scenario. The first one uses OSGi standard programming model and the second one uses iPOJO with our autonomic manager.

The first comparison between the two applications refers to their size. The context-aware part of the application without iPOJO has 252 lines against 133 inside the application using iPOJO. The difference comes from the code managing the context-aware composition.

A second comparison refers to the service selection time. We have implemented a benchmark measuring the service selection time in the two applications. This benchmark follows the five steps of the scenario. It changes user location and then sends a message to the user. This message is displayed on the most appropriate media. We measure the time needed to display a message in the two different implementations of the application.

To obtain accurate results, our scenario is played 10000 times, and we measure the average times. This evaluation was done on a 5 five years old laptop with a Pentium 3 at 1 GHz and 512 Mo of memory. Our benchmark shows an overhead of 9% in the application with our iPojo manager. This overhead comes from the invocation of the ranking method that uses reflection, and from the injection mechanism.

The overhead brought by the use of the iPojo framework seems reasonable in comparison to the development advantages. The benefits are naturally the re-use of robust automatic mechanisms offered by iPojo handlers and the simplicity of POJO developments applied to complex applications.

6. Related Work

Service-Oriented Computing is a new development paradigm enjoying of a good popularity nowadays. Thanks to the Web Services success, the Service-Oriented Computing has become very famous. However, development models and the need for dynamic execution frameworks are not deeply studied. Service-Oriented Component Runtime begins to appear with OSGi technology, Service Binder [13], Service Component Architecture [17] and Spring-OSGi [18]. These component models aim to help developers to implement service application. OSGi technology provides a dynamic service platform. The programming model provided by OSGi specification is very basic and the developer needs to manage all service interactions within the business code. Service Binder was the first service-oriented component model proposing to manage service dynamics outside the

business code. However, Service Binder is not extensible and its programming model is intrusive for the developer. Service Component Architecture (SCA) targets mainly Web Services. It proposes a standard development model to develop service applications. SCA does not address dynamic availability and does not manage dynamic composition. Spring-OSGi is a new project, proposing to use Spring component model to develop service application on the OSGi platform. As with iPOJO, this new service component model uses POJOs and tries to address dynamic service availability. Spring-OSGi uses an aspect framework to inject service dependencies. However, the developer needs to explicitly manage exceptions when an unavailable service is used. Spring-OSGi also does not manage dynamic service properties and component factories.

The emergence of smart and communicating devices has lead to a new computing paradigm commonly called pervasive or ubiquitous computing. Pervasive computing makes an extensive use of context. There have been many works over the last decades in the research field of context-aware and context-sensitive application. Most of these works focus on how to gather, store and retrieve context information with the use of models, ontology and semantics [19, 10, 11, 12]. These very interesting works tackle only a subpart of the problem of context-aware applications. Indeed, building such applications involves not only the retrieval of context information but also the adaptation of applications behaviour. Hereafter we present some works taking into account the context–aware adaptation.

In [20] and [21], frameworks called Safran and Camido are presented as enabling designers to create component based context-aware applications. Both works propose a container which handles the context-aware adaptation process. We think that the general service trading mechanisms at the basis of our framework is more given to represent the pervasive elements of the targeted environments. Thus, the loose aspect of component coupling is emphasized in our work.

7. Conclusion

Home-context applications are generally difficult to build. Indeed, developers need to manage the context and the service dynamics in addition of their business code. In this paper, we present our architectural approach to Pervasive Computing. We believe that service-oriented computing is a good challenger to build pervasive application. Our approach is based on a service-oriented component framework. Moreover, we propose to delegate the management of the contextual service composition inside component container thanks to an autonomic manager. The manager tackle critical home-context application problem as context dynamicity and service ambiguity.

Our work distinguishes itself from the state of the art by proposing an open architecture and the corresponding runtime that automatically track devices in the environment and adapt the application to fit the current context. The use of a service oriented component model allows our architecture to takes into account new runtime conditions such as the arrival, the modification and the departure of devices on the network, and the change of application service requirements bound to contextual events.

We implemented this architecture and tested applications above the OSGi framework and the Apache iPOJO service component model. This work has been demonstrated during the past review of the ITEA ANSO project. It has also been demonstrated at the IEEE Service Computing Contest 2006 and at the IEEE Consumer Communication and Networking Conference 2007.

We are currently investigating two main perspectives of this work. The first one concerns the service-oriented component runtime. We are studying how to improve service-component runtime with hierarchical service composition mechanisms. The second one is about the autonomic architecture. We are investigating a hierarchical architecture for autonomic managers in the home control domain. This architecture enables more powerful reconfiguration and optimizes the cohabitation of several applications on the gateway.

8. References

1 M. Weiser, “The computer for the 21st century”, Scientific American, 265(3):66-75, September 1991.

2 A. Ferscha, “Pervasive computing and communications”, Beyond The Horizon Thematic Group, Information Society Technologies, 2005.

3 M. N. Huhns and M. P. Singh. "Service-Oriented Computing: Key Concepts and Principles". IEEE Internet Computing, vol. 9:pages 75–81, Jan./Feb. 2005.

4 C. Marin, P. Lalanda and D. Donsez, “A MDE approach for power services development”, Internatinoal Conference on Service Oriented Computing, Amsterdam, december 2005.

5 J. O. Kephart and D. M. Chess, "The vision of autonomic computing", IEEE Computer, vol. 36, no. 1, 2003.

6 André Bottaro, Anne Gérodolle, Philippe Lalanda, "Pervasive Service Composition in the Home Network", 21st International IEEE Conference on Advanced

Information Networking and Applications (AINA-07), Niagara Falls, Canada, May 2007
7 F. Ramparany, J. Euzenat, T. Broens, J. Pierson, A. Bottaro, R. Poortinga, "Context Management and Semantic Modeling for Ambient Intelligence", Proceedings of the First Workshop on Future Research Challenges for Software and Services (FRCSS), April 2006.
8 P. Lalanda and J. Bourcier, “Towards autonomic residential gateways”, IEEE International Conference on Pervasive Services (ICPS 2006), June 2006.
9 J. Bourcier, C. Escoffier, P. Lalanda, “Implementing home-control applications on service platform”, 4th IEEE Consumer Communications and Networking Conference (CCNC’07), Las Vegas, January 2007.
10 A. Ward, A. Jones, A. Hopper, "A New Location Technique for the Active Office". IEEE Personal Communications 4(5) (1997).
11 D. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, K. Swenson, “Web Services Context (WS-Context) Version 1.0”, July 2003.
12 H. Chen, T. Finin, and J. Anupam "The SOUPA Ontology for Pervasive Computing", International Conference on Mobile and Ubiquitous Systems: Networking and Services, Boston, August 2004.
13 H. Cervantes, R. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model", 26th ACM International Conference on Software Engineering, Edinburgh, May 2004.
14 C. Escoffier, R. Hall, P. Lalanda, "iPOJO: An extensible service-oriented component framework” IEEE Service Computing Conference (SCC 2007), Salt Lake City, July 2007.
15 OSGi Alliance, “OSGi Service Platform Core Specification Release 4”, October 2005.
16 J. Christensen, J. Sussman, S. Levy, W.E. Bennett, Tracee Vetting Wolf, Wendy A. Kellogg, "Too much Information", ACM Queue, Vol. 4, N°. 6, July-August 2006.
17 F. Curbera, D. Ferguson, M. Nally and M. L. Stockton, "Toward a Programming Model for Service-Oriented Computing" in Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05), December 2005.
18 Interface21, “Spring OSGi Specification (v0.7),” http://www.springframework.org/osgi/specification, 2006.
19 G. Chen, D. Kotz, "Context-Sensitive Resource Discovery", Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, p. 243-252, Fort Worth,Texas, March 2003.
20 P.-C. David, T. Ledoux, "An Aspect-Oriented Approach for Developing Self-Adapting Fractal Components", Proceedings of the 5th International Symposium on Software Composition (SC2006), March 2006.
21 N. Behlouli, C. Taconet, G. Bernard, "An Architecture for Supporting Development Execution of Context-Aware Component applications", Proceedings of the 3rd IEEE International Conference on Pervasive Services (ICPS’06), June 2006.