An SCA-Based Middleware Platform for Mobile Devices

Daniel Romero, Carlos Parra, Lionel Seinturier and Laurence Duchien
University of Lille 1, INRIA Lille, Nord Europe, Laboratoire LIFL UMR CNRS 8022
Parc Scientifique de la Haute-Borne 40, Avenue Halley
59650 Villeneuve D'Ascq- France
{daniel.romero, carlos.parra, lionel.seinturier,laurence.duchien}@inria.fr

Rubby Casallas
Software Construction Group, University of Los Andes
Carrera 1 No. 18A 10
Bogotá- Colombia
rcasalla@uniandes.edu.co

Abstract

In pervasive environments, users can potentially access a variety of services through their mobile devices. However, in order to use the new services, we need to adapt the functionality of these devices. To achieve it, we propose to load a bootstrap into them that is able to communicate with the services. However this bootstrap has to be adapted due to the diversity of services offered in the environment, which are heterogeneous regarding aspects such as communication and discovery. Our bootstrap has two layers: the application layer and the middleware layer. This paper focuses in the middleware layer. We propose an architecture based on the Service Component Architecture (SCA). The architecture eases the reconguration of the components at runtime to support different communication mechanisms and service discovery protocols. Besides, using SCA, we can add new functionality to the middleware platform that can be provided by remote applications (SCA or not).

1. Introduction

In pervasive computing, several elements integrated in the environment provide services and information to be used through user's devices. Nevertheless, heterogeneity and dynamicity of mobile computing can restrict the interaction. Software running on these devices need to be adapted to communicate with the services. Thus, we need to offer mechanisms to enable interaction across mobiles in these environments.

To extend the device functionality, we aim to load, on every device, a bootstrap that is able to communicate with the services. However, this bootstrap has to be lightweight and adaptable to match the diversity of services, which are heterogeneous regarding aspects such as communication and discovery.

Our bootstrap has two layers: the application layer and the middleware layer. In the application layer we face several adaptation challenges, e.g., how to react properly according to preferences established by users and, how to respond to the context changes. Concerning the middleware layer, we are interested on how to support different communication mechanisms and service discovery protocols (SDP). Furthermore, we have to consider how to manage the context, which is a fundamental aspect in pervasive environments.

In this paper, we focus in the middleware layer, although we give some considerations associated with the application layer. We propose an architecture for a lightweight context-aware middleware based on Service Component Architecture (SCA). Four main modules compose our middleware: Kernel, Context Manager, Service Manager, and Adaptation Manager. Combining the services and component models, we expect to obtain clear concern separation that eases the loading of functionality when it is required, without overloading the devices.

Our bootstrap is intended to be used in environments, such as malls, where there are several users and diverse services can be offered. For example, users could query the meals offered by the restaurants through their mobile devices. They also may be interested in sharing a car and finding someone that offers the service.

The paper is structured as follows. Section 2 presents the background of this work: SCA and COSMOS, a framework

that we use to manage the context information. In Section 3, we discuss the challenges associated with a middleware in a pervasive environment. In Section 4 we describe the architecture of the middleware platform. Section 5 presents the related work and we finish with some conclusions.

2. Background

2.1. Service Component Architecture (SCA)

Service Component Architecture [1] is a set of specifications for building SOA (Service-Oriented Architecture) applications. In SCA, the basic construction blocks are the components, which require and provide services. SCA is designed to be language independent. The model is also neutral regarding the communication protocols used between remote components. In the same way, services can be described using different service description languages. With SCA it is possible to combine components which were built with different technologies. Furthermore, components behavior and interaction can be configured to different contexts, using the policy framework defined by SCA.

2.2. COntext entitieS coMpositiOn and Sharing (COSMOS)

COSMOS [3] is a component-based framework for managing context in pervasive environments. The framework retrieves context information, from entities such as sensors, and processes it according to the defined policies. These policies are described as hierarchies of context nodes using a dedicated composition language. The architecture of these context nodes is implemented taking advantage of the Fractal hierarchical component model [5]. To support the dynamic reconfiguration and evolution of the policies, COSMOS has a component-based architecture that exploits four design patterns (Composite, Factory method, Singleton and Flyweight). These patterns facilitate an efficient resource management. An important advantage of COSMOS is the possibility to describe a policy using other policies.

3. Challenges of a Middleware Platform in Ubiquitous Computing

Pervasive environments contain a large and diverse set of services. Nevertheless, these services are heterogeneous regarding aspects such as communication and discovery. This restricts their use. Hence, to ease the process of building context-aware applications, we have to tackle the following challenges:

1. How to retrieve context information: this information can be internal and external. By internal information we mean the resources associated with the devices (e.g., memory, battery and screen real estate) and the user preferences. External information refers to services and data provided by the entities such as sensor, servers and other user's devices. Hence, it is necessary to deal with different data representations that should be processed, consolidated and distributed in order to be useful for both the applications and the middleware layer.

2. How to discover and consume services using potentially diverse protocols: the middleware layer should support different discovery protocols, and provide a mechanism to filter services according to the device characteristics and user preferences. However, it is also necessary to consider that multiple communication ways are required. Hence, the middleware platform has to support different communication configurations using paradigms such as RPC (Remote Procedure Call), event-action or message-response. In the same way, it should be possible to select the routing algorithm and decide if the communication has to be synchronous/asynchronous, one or two ways, conversational, etc.

3. How to get the flexibility required to deal with environmental changes: flexibility refers to customization and addition of functionality at runtime. The middleware platform must be customizable because it will be deployed at devices of different kinds. The new functionality depends on the context changes. These changes are associated with the device itself, but also with services that can appear. Then, configuration and dynamic adaptation are essential functionalities that the middleware platform has to provided.

4. How to deal with mobility: in pervasive computing, the users move between different environments constantly. It causes variations on service availability. In order to deal with these variations, the middleware layer should be able to replace the services and to try to reduce the impact of such changes in user activities.

Considering all these challenges and in order to exploit the services provided in the environment, we need to accomplish dynamic adaptation, at two levels and for devices with several limitations. In the remainder of this paper, we focus mainly in the flexibility required in communication and service discovery although we consider briefly the other issues.

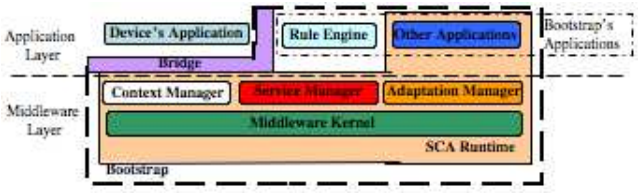

Figure 1. Middleware Platform Architecture

4. Middleware Platform Architecture

This section presents our middleware platform. It enables, through the user's device, the usage of resources present in pervasive environments. Furthermore, it can be extended with new functionalities. We propose to exploit SOA and Component-Based Software Engineering (CBSE) [9] features to achieve the required flexibility (in terms of adding and removing functionalities) both at middleware and application levels. In our approach, we use the main features of the two approaches like loose coupling, reusability, composability and service discovery, i.e., we use SCA. It means that all the middleware components can be reused and their functionality can be exposed like SOA services.

Figure 1 shows the middleware architecture. It is divided in the application and middleware layers. In the middleware layer resides all the functionalities required to deal with the resources present in the environment. The application level allows the user to exploit these resources. As we said before, our middleware platform is part of a bootstrap that is installed in the user devices. The description of the bootstrap generation and loading process can be found in [8]. A description of each component is presented below.

4.1. SCA Runtime

As said before, we will use an SCA based architecture. That means we need a SCA runtime. Since we have restrictions associated with device's capacities, the runtime has to be lightweight but with the basic functionality required to execute SCA applications.

4.2. Kernel

This component encapsulates functionality associated with communication and resource management. The communication service is composed by components, each one supporting a different communication mechanism. These components have associated a file descriptor (additional to their SCA descriptors) that specifies the communication model that is supported by them, i.e., the routing algorithm, the data encoding, the paradigm (e.g., RPC or message passing) and the mode (e.g., asynchronous, two ways, conversational with state). The communication service components can be added and removed at runtime. The resource management service uses policies to determine how it will manage some resources (as threads, memory, battery, etc.). The policies are established according to the device features, when the server generates the bootstrap. The Kernel component facilitates also the incorporation of new functionalities, such as persistence, context migration, and security, when it is required.

4.3. Service Manager

This component makes possible the filtering and invocation of services detected by the Context Manager (see section 4.5). The service filter is defined in the server side according to the device characteristics and it is customized on the mobile device according to the user preferences. When possible, the selection should be made considering the QoS. The Service Manager supports different SDPs. The functionality associated with each protocol is provided by a set of components. Each component of this set has a specific protocol responsibility such as discovery, invocation and advertising. In this way, at the beginning, the bootstrap will have only the functionality associated with discovery. The other functionalities will be loaded only if it is required. Each service discovery has a mapper associated. The mapper translates the service descriptions to the model used by the Service Manager. The components associated with invocation and advertising have to use the communication services provided by the Kernel component.

4.4. Adaptation Manager

The Adaptation Manager determines if it is necessary to load new functionalities in order to use a service provided in the environment. A central server provides these functionalities using generated components. The Adaptation Manager has to determine if the adaptation is required in the application or middleware levels, or in both, according to the service requirements.

4.5. Context Manager

The Context Manager identifies situations based on the context changes. By context changes, we mean mainly the availability of services, the variations in the devices resources, and other events in which the user may be interested. To do that, we use COSMOS. We argue that COSMOS is a good option because it is implemented using a component model that facilitates the reuse of policies and because it efficiently deals with memory footprint, instance management and resource consumption, all fundamental concerns in the type of devices that our middleware platform is intended.

5. Related Work

Several middleware approaches have been proposed to deal with pervasive environments. In [2] authors propose a solution to support SDP interoperability using event-based parsing. Each SDP has associated a parser, that translates the messages in events, and a composer, that makes the reverse process. However, they do not consider the loading of new functionalities to support other service discovery protocols.

The component-based service discovery framework proposed in [6] makes possible the development of middleware platforms that support different SDPs. The resource usage is minimized through component reuse. With the proposed architecture, it is possible to implement different SDPs, nevertheless, the issue of dynamic adaptation is not tackled.

ReMMoc [7] is a reflective middleware that can interact with heterogeneous services. It provides dynamic reconfiguration in terms of bindings and service discovery protocols. In order to do that, all functionality associated with the different communication mechanisms and service discovery must be in the device. In our case, the functionality should only be loaded when it is required.

AdaptiveBPEL [5] is a policy-driven middleware for the exible composition of Web Services. This approach is more focused on the automatic service integration, according to the applications needs, than in the middleware adaptation to enable the use of multiple kinds of services.

TinyLIME [4] is a middleware that supports the development of sensor network applications. This middleware platform provides data recovering from sensors considering aspects such as mobility and freshness. In our architecture, the Context Manager component has this responsibility. However, TinyLIME does not offer mechanisms to use the set services present in the environment.

6. Conclusions

In this paper, we have introduced a middleware platform for pervasive environments. With this platform, we want to enable users, through mobile devices, to access different resources of the environment such as services. In order to make it possible, the platform supports multiple communication models and service discovery protocols.

We have chosen SCA as an approach to build our middleware platform because we would like to take advantage of SOA and component paradigms. Using this approach, the user can be consumer and provider of services. We have also identified context management, adaptation, and mobility as important challenges in a context-aware middleware platform. These challenges drive our future work. In order to detect the services that can be used with user's devices and to enable the context migration, we have to work in a way to support service filtering. We should do it considering the different semantics used to describe the services and the QoS. We should also explore how to define, at runtime, policies in COSMOS that will make possible to take actions according to the current user needs. Moreover, it is required to determine how to deal with security and persistence issues. Finally, although there are several context-aware middleware, we consider that they are not flexible enough to let the users exploit all the potential of pervasive environments. With a service oriented middleware we expect to accomplish this flexibility.

References

[1] Service component architecture specifications. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications, 2007.

[2] Y.-D. Bromberg and V. Issarny. Service discovery protocol interoperability in the mobile environment. In T. Gschwind and C. Mascolo, editors, *SEM*, volume 3437 of *Lecture Notes in Computer Science*, pages 64–77. Springer, 2004.

[3] D. Conan, R. Rouvoy, and L. Seinturier. Scalable processing of context information with cosmos. In J. Indulska and K. Raymond, editors, *Proceedings of the 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'07)*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224, Paphos, Cyprus, jun 2007. Springer.

[4] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. Tinylime: Bridging mobile and sensor networks through middleware. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 61–72, Washington, DC, USA, 2005. IEEE Computer Society.

[5] A. Erradi and P. Maheshwari. Adaptivebpel: Policy-driven middleware for flexible web services composition. In *MWS 2005 Workshop at EDOC 2005*, pages 5–12, Enschede, The Netherlands, 2005.

[6] C. A. Flores-Cortés, G. S. Blair, and P. Grace. A multi-protocol framework for ad-hoc service discovery. In *MPAC '06: Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006)*, page 10, New York, NY, USA, 2006. ACM.

[7] P. Grace, G. S. Blair, and S. Samuel. Remmoc: A reflective middleware to support mobile client interoperability. In *CoopIS/DOA/ODBASE*, pages 1170–1187, 2003.

[8] C. Parra and L. Duchien. Model-driven adaptation of ubiquitous applications. In *1st International Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 08)*, pages 97–102, Oslo, Norway, June 2008.

[9] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.