Smart Environment Application Architecture

Zigor Salvador, Mikel Larrea, Alberto Lafuente
Department of Computer Architecture and Technology
The University of the Basque Country, Spain
Email: zigor@zigorsalvador.com, mikel.larrea@ehu.es, alberto.lafuente@ehu.es

***Abstract*—Pervasive computing is a field where many different entities come into play. In particular, development of useful applications in the field of pervasive healthcare requires dealing with substantial levels of complexity regarding the management of the pervasive environment where users and applications coexist. Service oriented middleware architectures simplify the task of creating those applications and turn the management of pervasive environments into a productive activity. We present our ongoing approach to the design of such a middleware architecture, known as the Smart Environment Application Architecture.**

I. Introduction

Pervasive healthcare, as is the case with most of the applications of ubiquitous computing [7], is based on the integration of a wide range of devices, resources and the infrastructure for their communication. Ideally, these constituent elements are well integrated with each other and become a robust, seamless and efficient distributed system on top of which various software applications can serve users in different and complementary ways.

Reality, however, is still far from this ideal vision, both regarding ubiquitous computing as a whole or speaking about the very particular realm of pervasive healthcare technologies. Bridging the gap between the technological reality and the social expectations is the main focus of our research, and we believe that one of the most important factors limiting the real-world deployment of pervasive applications lies on the integration methodologies, or lack thereof.

This paper analyses the main software requirements of applications which are being developed in the areas of pervasive healthcare and ubiquitous computing, presenting a middleware [1] architecture which integrates key aspects in the development of such applications. The inner workings of different elements from the architecture are discussed, and finally some conclusions are drawn with regards to the implementation and validation of the aforementioned architecture.

II. Pervasive Computing Applications

Providing application developers with the right software tools is vital in order to allow pervasive applications to live up to their full potential and effectively achieve their objectives. However, careful consideration of generic application requirements [4] and scopes is needed before these tools can be formulated and included into an architecture for the integration of pervasive resources and the applications which use them.

Research partially supported by the Spanish Research Council, grant TIN2006-15617-C03-01, and the Basque Government, grant S-PE06IK01.

In the context of this work, we will define a pervasive computing application as a high-level software entity, which exists inside the boundaries of a pervasive computing environment, is user-centric in its nature, and interacts with both the environment and the people in it to achieve one or more functional goals. Depending on the particular type of application and the features of its surrounding environment, applications will make use of certain types of resources to achieve their goals. We group these resources in three main categories, namely: control resources, context resources and interaction resources.

III. Pervasive Computing Resources

Control resources include all the controllable networked devices in the environment. These devices vary in form, size and purpose, but all share the ability to change the environment in one way or another. Many pervasive computing applications need to act on the environment on behalf of users, and consequently rely on remote control resources to fulfill that task. Examples of such resources include security locks, fire alarms, lighting controls, air conditioning systems, etc.

Context resources are formed by sensing devices [6] and other networked sources of environmental information. From wearable heart rate monitors to fall detection sensors, pervasive computing environments are full of devices with varying degrees of sensing capabilities. The data gathered from all the different sensing devices produces context information, which can be used by applications to produce smart behaviour and non-intrusive proactive reasoning.

Being built around users, pervasive applications often require human interaction in the form of feedback or commands to carry out their functional goals. Interaction resources which make this human-computer interaction possible are devices such as mobile phones, PDAs, computers, TV sets, audio systems and others, which can make use of one or multiple interaction modalities in accordance with their features and the limitations imposed by the environment and its context.

If the development of pervasive applications is to be productive, and the resulting applications are to be correct, developers need to be supplied with a stable set of tools which will simplify, unify and standardize the access to all of the underlying resources while keeping the flexibility, adaptability and extendability of the system at a reasonable level. A service oriented architecture [2] based on middleware has the potential to fulfill those requirements and produce a qualitative leap in the development of pervasive computing applications.

PERVASIVEHEALTH 2008, 30 Jan - 1 Feb. Tampere, Finland

DOI 10.4108/ICST.PERVASIVEHEALTH2008.3175

IV. Smart Environment Application Architecture

The software architecture for applications of smart environments, depicted in Fig. 1, presents a layered set of services or modules, where the modules in the bottom layer unify access to the various types of hardware resources described in Section III and the modules in the middle layer provide additional functionality by performing both horizontal interaction with other middleware services and vertical interaction with the application that sits on top of the stack and the hardware driver modules beneath them.

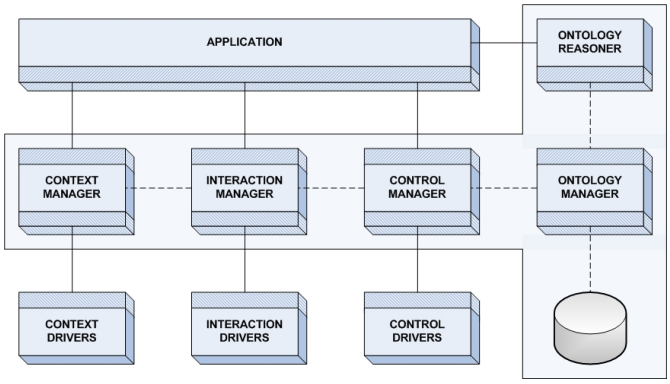

Fig. 1. The layered middleware architecture

The context, control and interaction driver modules of the architecture represent generic services that, once implemented, encapsulate a given hardware, technology or device (e.g., a X10 control driver, a Zigbee context driver or a Pocket PC interaction driver). Driver modules are conceptually grouped according to the nature of the resources they encode, and provide one of three unified interfaces to the modules above them. Driver modules can be very specific and therefore contain specific parts of code to access the hardware and control the flow of information between the application and the target resources.

The middle layer contains the managers or main functional modules, i.e. the context manager, control manager, interaction manager and ontology manager. The first three modules are closely related to a given type of resource and provide the application with a unified and managed access layer to the resource. The fourth module, the ontology [5] manager, is in charge of managing the persistent ontology database, importing ontological data from the driver modules, and setting up the initial instance of the ontology reasoner which can be leveraged by applications to produce smart behavior and proactive interaction with end-users.

This service oriented middleware architecture is based on distributed modules which can be deployed across a network where loose coupling is a key feature and multiple instances of both low level modules and applications can be found. The overhead of locating instances of low level services, managing their state, regulating access to them and granting the general consistency of the distributed system is shared by the main functional modules, and is thus transparent to both applications and low level modules. This way, application developers can concentrate in their applications, and resource driver developers can create efficient hardware drivers without worrying about the inner workings of the middleware.

V. Implementation and Validation

The empirical validation of the proposed software architecture is currently being carried out at the University of the Basque Country, through the implementation of an open-source middleware platform for the integration of generic pervasive applications known as Smart Environment Application Middleware or, more briefly, Seamware.

Seamware is fundamentally a layered, object oriented middleware extension to the Apache River [3] technology (formerly known as Jini Network Technology). This technology is a service oriented architecture that defines a programming model which both exploits and extends Java technology to enable the construction of secure, distributed systems consisting of federations of well-behaved network services and clients. Seamware uses the River technology to build a smart environment application middleware which is adaptive, scalable and flexible as typically required in dynamic computing environments, e.g., pervasive healthcare scenarios.

In Seamware, all of the functional modules presented in Section IV become autonomous, yet interrelated, Apache River services, which dynamically detect the existence of other Seamware services and clients and provide application developers with a unified way of accessing the underlying hardware and software resources they need in order to build their applications. Seamware focuses on providing an easy to use and easy to extend middleware platform that will enable the creation of pervasive computing application prototypes with a very modest investment in development time, suitable for both small development teams and didactic purposes.

Seamware is still in its early development stages but already shows some promise. Release to the general public is expected during the year 2008, followed by the introduction of a pervasive computing application and hardware prototype in order to validate the architecture with a real-world implementation.

References

[1] Philip A. Bernstein. Middleware: A model for distributed system services. *Communications of the ACM*, 39(2):86–98, 1996.

[2] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR, 2005.

[3] The Apache Software Foundation. Apache river jini implementation community (http://incubator.apache.org/river/).

[4] Zigor Salvador, Mikel Larrea, and Alberto Lafuente. Infrastructural software requirements of pervasive health care. In *AC 2007: Proceedings of the IADIS International Conference on Applied Computing*, pages 557–562, Salamanca, Spain, 2007. IADIS Press.

[5] Frank Van Harmelen and Deborah L. McGuinness. Owl web ontology language overview (http://www.w3.org/tr/owl-features/).

[6] Kristof Van Laerhoven. The pervasive sensor. In *UCS 2004: Second International Symposium on Ubiquitous Computing Systems*, pages 1–9, Tokyo, Japan, 2004. Springer.

[7] Mark Weiser. The computer for the 21st century. *SIGMOBILE: Mobile Computing and Communications Review*, 3(3):3–11, 1999.