

Building Home Monitoring Applications: From Design to Implementation into The Amigo Middleware

Wilfried Jouve¹, Noha Ibrahim², Laurent Réveillère¹, Frédéric Le Mouël², Charles Consel¹

¹INRIA PHOENIX / LaBRI, ENSEIRB, Bordeaux, F-33402, France

{jouve, reveillere, consel}@labri.fr

²INRIA ARES / CITI, INSA-Lyon, Lyon, F-69621, France

{noha.ibrahim, frederic.le-mouel}@insa-lyon.fr

Abstract

The proliferation of smart communication devices based on technologies such as Radio-Frequency Identification (RFID) makes ubiquitous computing evolving at a frantic pace. This evolution leads to the development of applications, called monitoring applications, that offer a host of new functionalities based on context information provided by tagged entities. In this paper, we introduce a software architecture dedicated to monitoring applications and define a mapping to the Amigo middleware which is dedicated to ambient computing. We illustrate our approach by developing two real-size applications for child monitoring and object reminder. In these experiments, our approach have demonstrated its usability and ease of programming.

Keywords: Software Design methodology, Middleware abstraction, Home Monitoring Applications, Object Tagging

1 Introduction

The proliferation of smart communication devices based on technologies such as Radio-Frequency Identification (RFID) and Ultra-wideband (UWB) [13] makes ubiquitous computing evolving at a frantic pace. These small devices, called tags, allow a location-tracking with high accuracy. They lead to the development of applications, called *monitoring applications*, that offer a host of new functionalities based on context information provided by tagged entities. However, developing enriched, real-size monitoring applications is quite a challenge.

Although existing middleware provide appropriate services (e.g., service discovery) for managing various heterogeneous devices of the home environment, they do not offer or even suggest a framework for developing monitoring applications. However, developing advanced, real-size monitoring applications require many software components to acquire context information from physical sensors, check monitoring criterias, and notify users according to their preferences.

This paper aims to define a framework for developing monitoring applications. We have introduced a software architecture for this class of applications and defined a mapping to the Amigo middleware [1] which is dedicated to ambient computing. Finally, we have applied our approach to the development of two real-size monitoring applications: child monitoring and object reminder.

The contributions of this paper are as follows.

- **Software architecture.** Based on a thorough analysis of monitoring applications, we have defined a software architecture that consists of three layers: (1) the bottom layer tracks entity locations acquiring raw information from sensors; (2) the middle layer checks criteria when an entity moves from one area to another and (3) the top layer warns supervisors based on notification preferences. These layers are linked by the concept of rule that defines monitoring criteria.
- **Middleware abstraction.** We have developed services on top of the Amigo middleware to map the various building blocks of our software architecture. This mapping raises the level of abstraction provided by Amigo to match the one required for developing home monitoring applications.
- **Real-size applications.** Following our approach, we have developed two real-size monitoring applications. First, the *child monitoring application* helps parents ensure the safety of their child depending on the room where (s)he is located in the home. Second, the *reminder application* prevents users from forgetting objects required for their planned activities.

The rest of this paper is organized as follows. Section 2 presents the home monitoring class of applications. Sections 3 and 4 present our approach including the software architecture and its mapping to the Amigo middleware. Section 5 assesses our approach by describing two case studies:

a child monitoring and an object reminder. Finally, Section 6 presents related work and Section 7 concludes.

2 Home Monitoring Applications

The goal of home monitoring applications is to monitor entities, that is, objects and people, inside the home environment. Typically, such an application acquires context information about entities from physical sensors, check monitoring criteria, and notify users according to predefined preferences. Various monitoring applications can be developed in an ambient home environment. As an example consider an object reminder application. This application prevents users from forgetting objects when leaving their home. Entity monitoring is another instance of the home monitoring class of applications. It consists of keeping close watch over people or objects inside the home.

2.1 Software requirements

The development of a home monitoring application requires many software components. First, entities must be tracked inside the home. Second, relevant criteria called monitoring rules must be checked. Finally, a user may need to be warned.

Tracking entities. Tracking entities inside the home requires the use of physical sensors [13]. Such location systems typically send events to notify applications of entities movements. However, these events are too generic and need some processing at the application level to match its requirements. For example, events providing Cartesian coordinates of entities have to be filtered if the application only needs to be notified of area changes. Areas are defined by a cut-out of the monitoring space in hierarchical locations such as rooms or floors.

Checking monitoring rules. Monitoring rules express constraints on entity locations. A monitoring rule is verified true when the monitoring entity enters a notification area (spatial location such as the kitchen) and some conditions (*e.g.*, time of the day, location of the supervisor) become true.

Notifying users. The notification of the supervisor does not require very specific software components. However, it requires specific support to retrieve user preferences (which notification system to use), to acquire some privacy context information (is the user alone?) and to find renderers close to the user location (service discovery).

2.2 Hardware requirements

Location systems rely on physical sensors such as tags and tag readers. Therefore, the size of a notification area depends on the accuracy of the tag readers deployed. These tags are either passive or active. Using passive tags requires the deployment of tag readers in each monitoring areas, increasing the number of software components to coordinate. In addition, passive tag readers are often embodied by gates installed

at strategic places such as door frames, making the monitoring of an open area difficult. On the other hand, systems based on active tags such as Ubisense [13] only require few readers to obtain the entity locations by triangulation. By providing a uniform location of entities by Cartesian coordinates, this system enables reuse of software components and the definition of arbitrary notification area. In the rest of this paper, we focus on systems based on active tags.

3 Software Architecture

In this section, we define a software architecture for the development of home monitoring applications. Our software architecture consists of three layers: (1) the bottom layer, named *Context Management*, tracks entity locations acquiring raw information from sensors; (2) the middle layer, named *Rule Management*, checks criteria when an entity moves from one area to another and (3) the top layer, named *Notification Management*, warns supervisors based on notification preferences.

3.1 Context Management

The management of location events requires components able to filter, abstract and make context information available to monitoring applications. This context information is supplied by location systems. The context manager consists of two components: (1) a wrapper for event sources that standardizes event formats and publishing/registration mechanisms, and (2) an event interpreter to filter and abstract events to match the level of abstraction required by applications.

Location systems may send multiple events in a short period of time corresponding to the movements of several entities. In the context of the object monitoring application, the entities going outdoors are either the user or the objects (s)he is carrying with him/her. Therefore, once the application has received an event for the user, it must wait for events of objects that follow. To overcome this low-level consideration, we introduce a buffering mechanism to aggregate and abstract different events into one.

3.2 Rule Management

Because monitoring applications rely on monitoring rules, our software architecture includes components for loading these rules, managing subscriptions and notifying applications. These functionalities are achieved by two main components: the rule checker and the rule loader.

Rule checker. The rule checker manages monitoring rule subscription and evaluation. A set of events triggers monitoring rules that result into context changes. Consequently, the rule checker subscribes to a set of events and evaluates the monitoring rule based on incoming events. If the monitoring rule turns out to be verified, the subscribing applications are notified.

Rule loader. The rule loader makes monitoring rules active by subscribing to the rule checker. Before subscribing to a monitoring rule, the rule loader checks the fulfillment of a set of conditions defined for this rule. While the rule checker reacts to dynamic changes of the context, the rule loader only depends on static conditions such as the time of the day. In the context of an object reminder application, the rule loader depends on the list of activities, extracted from the home agenda, planned in the following few hours.

3.3 Notification Management

Once a monitoring rule is verified, a supervisor must be warned. The notification manager defines notification strategies according to user preferences and context information. These strategies rely on the type of the alarm (from warning to emergency), the location of the user and available renderers in this area, as well as on privacy context information. The notification manager can decide to use blinking lights, to display a textual alarm message or to trigger a siren. In addition, the volume of the siren and the blinking repetitiveness of the lights can be adjusted according to the nature of the alarm and user preferences.

4 Mapping the Software Architecture to the Amigo middleware

We first briefly introduce the Amigo middleware before presenting the mapping of our software architecture into this middleware.

4.1 Background

The Amigo project [2] focuses on the usability of a networked home system by developing an open, standardized, interoperable middleware. The networked home includes devices from Personal Computing, Mobile, Consumer Electronics and Home Automation domains. The Amigo middleware guarantees automatic dynamic configuration of these devices and services within the home system by addressing autonomy and composability aspects. Its goal is also to support interoperability between equipments and services within the networked home environment.

The Amigo base middleware provides a framework for the definition, discovery, and late binding of services. All the services for applications and users are defined, registered and discovered through the features of the Amigo base middleware. The design of Amigo is partially based on the OSGi framework technology that defines a standardized, component-oriented computing environment for networked services. The programming and deployment framework for Amigo services aims at guiding and facilitating service development, service lifecycle and enabling dynamic configuration of the Amigo system.

Let us now present a mapping of our software architecture for home monitoring applications to the Amigo middleware. The service-oriented nature of this middleware requires the development of several Amigo services to match the level of abstraction provided by the building blocks of our software architecture. First, the context management layer is embodied by the the Amigo Context Management Service [3]. Second, the rule management layer is supported by the Amigo Awareness and Notification Service [8]. Finally, the notification management layer is supported by the Amigo service discovery mechanism [2]. Other software components for which Amigo does not provide any support are gathered into a set of monitoring services. These services include the rule loader of the rule management layer and a service to set up notification strategies for the notification management layer. Figure 1 illustrates the various Amigo services and application-specific services required to develop a home monitoring application.

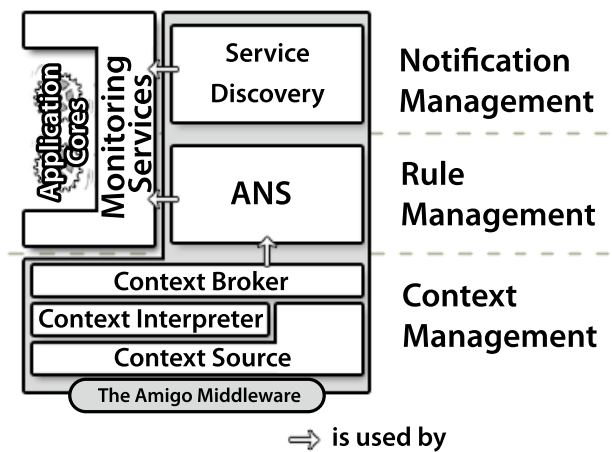


Figure 1. Home Monitoring Applications in the Amigo Middleware

4.2 Context Management Service

The aim of the Context Management Service (CMS) is to acquire raw data from sensors and other sources, to aggregate and abstract these data into context information, and to make this information available to other Amigo services and applications. The context management service is composed of three sub-services: the context source, the context broker and the context interpreter.

Context Source. The context source abstracts a data source such as a tag reader or a user agenda. The context source provides a uniform interface for services to subscribe to it and notifies services when the context changes.

Context Broker. The context broker is a repository of available context sources. It can be seen as a service discovery engine for context sources.

Context Interpreter. The context interpreter is a specific context source that aggregates and filters information from several context sources to raise the level of abstraction of the information. For example, the context interpreter interprets Cartesian coordinates provided by location sensors to notify services only of room changes.

4.3 Awareness and Notification Service

The Awareness and Notification Service (ANS) notifies application services of any significant context change. Application services must register ANS rules to describe which context information they are interested in. These rules are defined using the Event-Condition-Action Rule Language [7]. An example of ANS rules is displayed in Figure 2. It specifies that the swimming pool is a forbidden area for Pablo when he is not with his mother. ANS rules are obtained by translating monitoring rules.

```
UPON entertrue(isLocatedIn(Pablo, Swimming.Pool))
WHEN not(isLocatedIn(Pablo, Maria.location))
DO notify(Maria, 'Pablo is close to the Swimming.Pool')
```

Figure 2. An ANS rule example for monitoring a child

4.4 Service Discovery

To warn a supervisor once a monitoring rule becomes verified, monitoring services have to find appropriate devices in the the networked home to render an alarm. This rendering may vary according to user preferences and context information. Therefore, the Amigo middleware provides an Amigo-aware service discovery that enables semantic-level, context-aware service discovery. Service discovery commonly employs a Service Discovery Protocol that locates services satisfying a specific service description.

5 Case Studies

To assess our approach, we have developed two real-size monitoring applications based on our software architecture. The *child monitoring application* helps parents to make sure their child is safe when (s)he is alone in a room. The *reminder application* prevents users from forgetting objects required for planned activities. While having different goals, these applications are built on top of common services of the underlying Amigo based middleware. As an example, they both rely on the use of monitoring rules.

We now describe the hardware components, Amigo services and monitoring services used for our experiments. Then, we present the child monitoring and the object reminder applications.

5.1 Experiments

The tracking of entities is performed using active tags. The Ubisense platform [13] has been used to get real-time Cartesian coordinates of objects and people. It consists of four tag readers and a set of active tags, one for each tracked entity. Its integration into the Amigo middleware has been obtained by wrapping the Ubisense API as a context source. For our experiments, we have deployed an infrastructure including the Ubisense platform and the Amigo middleware, in the fourth floor of our laboratory building.

Because the physical sensors we used for tracking location of entities provide Cartesian coordinates, we integrate in our two applications a context interpreter, as presented in Section 4. This service provided by the Amigo-based middleware is parameterized by a map that defines areas of interest inside the home. Note that the buffering mechanism provided by the context interpreter is particularly useful for the object reminder application. Indeed, it enables to know which objects a user is carrying when leaving the home.

Notifications are carried out by computer screens, computer speakers, PDAs, UPnP lights and video cameras. This set of devices allows various notification strategies to be tested. Textual notifications are rendered by screens and PDAs while audio notifications are rendered by speakers. These notifications can be completed by images and video streams from video cameras and by blinking lights. The supervisor can also be notified by email if (s)he is not at home.

Monitoring rules are defined by using Web interfaces. Monitoring applications are developed within the Amigo OSGi programming framework and are deployed in the Amigo OSGi deployment framework where they can interact with the Amigo middleware.

5.2 Child Monitoring

The child monitoring application helps parents make sure their child is safe when located in another room of the house. As an example, consider Maria and Jerry, two parents who want to be notified when their child, Pablo, is leaving safe areas of the home. The development of this application requires the use of several services and hardware components, as described above. It also requires the generation of monitoring rules to express the constraints on Pablo's location. Examples of these rules are depicted in Table 1. The first three rules specify that a supervisor must be warned if Pablo enters a room which is neither the bedroom or the living room. The supervisor to warn depends on the time of the day. In our example, notifications must be send to Pablo's father during the morning, to his mother during the afternoon and to both of them during the night. The last rule specifies that Maria must be warned if Pablo enters the swimming pool area except if she is currently with him.

Entity	Notification area	Supervisor(s)	Condition
Pablo	\neg (Bedroom \vee Living room)	Jerry	Time \in 8:00am - 12:00am
Pablo	\neg (Bedroom \vee Living room)	Maria	Time \in 12:00am - 8:00pm
Pablo	\neg (Bedroom \vee Living room)	Maria \wedge Jerry	Time \in 8:00pm - 8:00am (Day+1)
Pablo	Swimming pool	Maria	Pablo.location \neq Maria.location

Table 1. Monitoring Rules for supervising Pablo

5.3 Object Reminder

The object reminder application prevents users from forgetting objects required for activities planned in their agenda. To achieve this task, objects carried by users are monitored and compared with objects associated to agenda activities. When a user goes through the home entrance and an object has been forgotten, the user is notified. Reminder rules describe objects required for a given activity of the user agenda. Rules are subscribed when the user comes back home and wonders about objects required for planned activities while out of the home.

	Purpose	Location	Transport	People
08:00am 12:00am	Work	University	Tramway	Peter \wedge James
...				
06:00pm 07:00pm	Sport Practice	Stadium	Car	Maria
...				
09:00pm 11:00pm	Watching football game	Stadium	Car	Steven \wedge John
11:00pm		Home	Car	

Table 2. Excerpt of Jerry's Agenda

The structure of the user agenda facilitates the creation of reminder rules. More particularly, once a reminder rule is associated to an activity, this rule is propagated through the user agenda to activities sharing common features. In the same way, new activities are also linked to previously created reminder rules. We now describe how reminder rules are associated to activities described in the user agenda. The user agenda is a list of activities (*e.g.*, *Sport Practice*), each of them defined in a time slot. An activity is fully described by four components: the purpose (*what*), the location (*where*), the transport (*how*) and people involved in the activity (*who*), as shown in Table 2. For example, Jerry takes his *Car* to *Practice Sport* at the *Stadium* with *Maria*. Such decomposition aims at reusing activity elements through multiple activities. For example, the *Stadium* location is also involved in the *Watching football games* activity.

A reminder rule is the association of an object with one or several activity components. If an activity consists of components involved in a reminder rule, the object referred to by this reminder rule is automatically declared as required for this activity. This enables the propagation of reminder rules. The *parking card* is associated with the *Stadium* location and the *Car* transport via a reminder rule defined in Table 3. This reminder rule is automatically bound to the activity of *Sport Practice* and the activity of *Watching foot-*

ball game (see Table 3) since they both contain the *Stadium* location and the *Car* transport, as activity components.

Entity	Notification area	Supervisor	Conditions
Jerry	Home entrance	Jerry	(Parking Card).location = Home entrance

Table 4. Monitoring Rules for the parking card

When loaded by the ANS, reminder rules are translated to monitoring rules. As an example, the reminder rule related to the *parking card* (Table 3) is translated to the monitoring rule shown in Table 4. In our example, Jerry is both the monitored entity and the supervisor and there is only one notification area: the home entrance. If Jerry does not carry his *parking card* with him, he is immediately notified when leaving the home. This monitoring rule is loaded when Jerry comes back home before 9:00pm.

6 Related Work

Tag-based technology opens up a host of new applications, from merchandise tracking [4] to people safety and monitoring [9, 14, 10]. It also brings new issues in context information management [6, 15].

Many monitoring applications have already been developed. Borriello *et al.* [5] have evaluated the use of RFID tags to build a reminder system. The gate reminder [11] is a home appliance that notifies users about missing objects and messages when leaving home. Other monitoring applications include supervising patients applications [14] and elderly people supervising applications [9]. However, none of these applications are based on either a well identified software architecture or a dedicated middleware. Indeed, their design and implementation have been made from scratch disregarding re-use. In contrast, our approach defines common software components as well as relations between these components and the underlying Amigo-based middleware. The genericity of our approach to developing monitoring applications has been validated by different examples.

Middleware-based approaches (*e.g.*, Gaia [12]) have been developed to address a number of key issues of ubiquitous computing such as mobility, service discovery and distributed applications. These approaches provide specific abstractions and mechanisms to the developer. However, these abstractions and mechanisms are too generic to match the requirements of home monitoring application. In contrast, our ap-

Entity	Activity components	Occurrence
parking card	Location = Stadium \wedge Transport = Car	Always
Star Wars DVD	People = Peter	Once
MP3 Player	(Purpose \neq Teaching \wedge Location = University) \vee Purpose = Jogging	Always

Table 3. Jerry's Reminder Rules

proach introduces a framework dedicated to the development of home monitoring applications on top of a customized middleware.

7 Conclusion

In this paper, we argued that the scope and underlying technologies of the home monitoring domain make software development of applications an overwhelming challenge. To take up this challenge, we have introduced a software architecture dedicated to home monitoring applications. We have developed services on top of the Amigo middleware to map the various building blocks of our software architecture. This mapping raises the level of abstraction provided by Amigo to match the one required for developing home monitoring applications. We illustrated our approach by developing two real-size applications for child monitoring and object reminder. In these experiments, our approach have demonstrated its usability and ease of programming. As an example, the object reminder has only been required one week of development by a programmer.

Acknowledgement

The work reported here is supported by the European Commission as part of the IST-IP Amigo project under contract IST-004182.

References

- [1] The IST European project Amigo, <http://www.hitech-projects.com/euprojects/amigo/>.
- [2] Amigo consortium, deliverable d3.1: Detailed design of the amigo middleware core: Service specification, interoperable middleware core, 2005.
- [3] Amigo consortium, deliverable D4.1: Report on specification and description of interfaces and services, 2005.
- [4] Y. Bendavid, S. F. Wamba, and L. A. Lefebvre. Proof of concept of an RFID-enabled supply chain in a B2B e-commerce environment. In *ICEC '06: Proceedings of the 8th international conference on Electronic commerce*, pages 564–568, 2006.
- [5] G. Borriello, W. Brunette, M. Hall, C. Hartung, and C. Tangney. Reminding about tagged objects using passive RFIDs. In *UbiComp'04: Proceedings of the International Conference on Ubiquitous Computing*, 2004.
- [6] S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma. Managing RFID data. In *VLDB '04: Proceedings of the International Conference on Very Large Data Bases*, 2004.
- [7] P. D. Costa, L. F. Pires, and M. J. van Sinderen. Architectural patterns for context-aware services platforms. In S. K. Mostefaoui and Z. Maamar, editors, *Second International Workshop on Ubiquitous Computing, Miami*, pages 3–18, Portugal, April 2005. INSTICC Press.
- [8] R. Etter, P. D. Costa, and T. Broens. A rule-based approach towards context-aware user notification services. In *IEEE Conference on Pervasive Services 2006 (ICPS'06)*, Lyon, France, 2006.
- [9] L. Ho, M. Moh, Z. Walker, T. Hamada, and C.-F. Su. A prototype on RFID and sensor networks for elder healthcare: progress report. In *E-WIND '05: Proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, pages 70–75, 2005.
- [10] IBM. Danbury hospital and ibm team up to create nation's first safe and efficient emergency department. *IBM news*, 2005.
- [11] S. W. Kim, M. C. Kim, S. H. Park, Y. K. Jin, and W. S. Choi. Gate reminder: a design case of a smart reminder. In *DIS '04: Proceedings of the 2004 conference on Designing interactive systems*, pages 81–90, 2004.
- [12] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, pages 65–67, 2002.
- [13] P. Steggle and S. Gschwind. The Ubisense smart space platform. In *Adjunct Proceedings of the Third International Conference on Pervasive Computing*, 2005.
- [14] L. Sullivan. Alzheimer's patients get help at home from wireless networks. *Information Week*, 2004.
- [15] F. Wang and P. Liu. Temporal management of rfid data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1128–1139. VLDB Endowment, 2005.