Implementation of an AmI Communication Service Using a Federated Event System Based on Aspects[1]

Lidia Fuentes,
LCC Malaga University
lff@lcc.uma.es

Daniel Jimenez,
LCC Malaga University
priego@lcc.uma.es

Rene Meier,
DSG Trinity College,
rmeier@cs.tcd.ie

Abstract

Event-based communication can be considered naturally suited to support Ambient Intelligence and Ubiquitous Computing applications due to its asynchronous nature and due to loose coupling between application components. Event systems support different properties depending on the specific problem domain for which they have been developed. Using these event systems in a federated way, where events are disseminated across the boundaries of a single event system, has been possible in some areas. However, such federation has typically been realized as bilateral inter-working federation between designated pairs of event systems that rely on hardcoded architectures, which are inherently difficult to maintain when systems evolve over time. This paper presents an Ambient Intelligence platform, called AOPAMI, that uses aspects to enable truly multilateral federation between inter-working heterogeneous event systems. AOPAMI also solves the technology evolution problem using the Aspect Oriented Software Development paradigm.

1. Introduction

Ambient Intelligence applications are becoming the new revolution in computer science that is being supported by several organizations such as [1], [2], and [3] as well as by the work of individuals such as [4], and [5]. The next 10-15 years will likely be the era of the Ambient Intelligence (or AmI) and we will be surrounded by thousands of intelligent devices and applications that will help to fulfill everyday tasks. Today this kind of application is gaining acceptance due to the reduced manufacturing cost of the supporting hardware devices, the increase of users of portable devices such as mobile phones, PDAs and laptops and the development of new communication technologies such as WiFi or Bluetooth.

AmI applications are distributed and loosely coupled in nature and as a consequence, event systems are a natural choice for realizing communication. Unfortunately, there exist different event systems, each one providing solutions for specific domains and problems such as Siena [6], Steam [7] and CNS [8]. As a result, if we select one of these systems to implement an AmI application we might find that this application will be unable to interact with others applications that use different event systems.

Meier and Cahill [9] provide a classification of event systems that categorizes event systems according to a set of properties. Ryan et al. [10] has used this taxonomy and has shown that it is possible to implement a Federated Event System (FES). The FES provides a direct event translation between systems providing an adaptation mechanism for event system properties. This mechanism adapts the events produced in one system to a common and neutral representation, called the FES event model. This model enables the transformation from its representation to other event systems representations. As a consequence, this mechanism enables transparent access to the services provided by a diverse set of event systems, thereby creating a multilateral inter-working federation of event systems while maintaining the independence and coherence of the individual event systems.

However, this adaptation mechanism has some limitations. The first limitation is that it is not always possible to perform an adaptation of the events from one system to another. This limitation is due to the heterogeneity in the capabilities of the event systems and due to differences in event formats. Thus, only a subset of event system properties and features is available to be used in the FES. The second limitation is that it neither provides mechanisms to handle the evolution of event systems over time, for example, due to technological changes, nor considers possible changes to the event system configuration used at

[1] This work is partially financed by IST-2-004349-NOE AOSD-Europe and the Spanish Ministry of Technology and Science, CICYT, under grant TIN2005-09405-C02-01.

runtime, for example, due to modifications to the environment. In the first case, if we need to add support for a new event system or an existing event system changes, it will be necessary to modify and to recompile the entire application and then to redeploy the system on all devices in order to update the application. The workload associated with this task is expected to be considerable given the scale of such applications. In the second case, the application always uses a fixed set of event systems and it can not be adapted to change these at runtime. This feature allows, for example, to disable or to unload those event systems that are not anymore used, thus, saving significant computational resources. Finally, it also enables us to decide, even at runtime, which one event system the platform will use to forward the events. This decision can be taken at every moment depending on the environment variable values and state.

Based on our previous experiences refactoring applications using the Aspect Oriented Software Development (or AOSD [11]) paradigm [12] [13], we have developed a middleware platform, called AOPAmI. This paper shows the benefits of using the AOSD paradigm for addressing the heterogeneity, evolution and adaptation issues in FES applications. We show how the AOPAmI platform can be effectively applied to develop FES applications.

The rest of this paper is structured as follows. In section 2 we discuss the benefits of using a federated event system. Section 3 describes an example application. Section 4 presents the AOPAmI platform describing its main features. In section 5, we explain the application implementation process, indicating the problems faced when adapting existing applications to the AOPAmI platform. Finally, in the last section we present our conclusions and future work.

2. Federating Event Systems

Event based middleware is nowadays applied in a growing number of different application domains including finance, telecommunications, smart environments, health care and entertainment. The use of event systems allows the integration of heterogeneous applications in an anonymous and scalable way, due to the properties inherent to event systems [10]. These event systems provide a wide range of services to the applications that use them. However, the problem of integrating different event systems in order to use their services in a multilateral inter-working and federated way has not been widely considered by researchers. The use of a FES provides a powerful tool to build truly distributed applications that can use a wide range of services provided by the different event systems integrated in the federation.

For example, suppose we design a control application that detects invalid value ranges in multiple applications controlling a production chain. The applications throw control events every time that a task finishes, and we have different event systems for each application used in the production chain. If we want to avoid the problems derived from the integration of these event systems, our choice will be to use a FES. This FES integrates all the used event systems and enables the easy introduction of new event systems in the production chain. Otherwise, we will be forced to modify the implementation of the control application each time a new event system is added. The use of a FES is justified here, because it provides an elegant solution to the heterogeneity problem associated with the use of different event systems.

Another problem that can be solved used FES is system scalability. Suppose that the event systems do not provide content-based filter capabilities. But, they must send only those events whose contents match a specific criterion, for example, values that are out of range. Consider the importance of this problem if the communication channel used by the system only support a limited amount of events. As a consequence, we must minimize the number of events thrown by the system. An adequate way to achieve this is to develop a FES that provides filtering capabilities for all the event systems integrated into it.

As a conclusion, we must say that implementing a FES is complex, but it provides a transparent communication mechanism that does not interfere with the original event system behaviour. Additionally, the resulting system maintains all the benefits associated with the use of event systems, such as scalability and loose coupling between entities that use the FES.

3. Using Federated Event Systems

The distributed application example, which we have selected to evaluate our approach, is split in several parts that are executed on different hardware devices, as is illustrated by Figure 1. Together these applications form a typical FES application that exhibits the problems associated with distributed event based applications such as device heterogeneity, scalability and limitation on communication. To illustrate all these problems and how the FES solve them, we have taken the following example from the work of [10]. In this paper, we show how to adapt this example from the FES to AOPAmI as well as the advantages of using aspects.

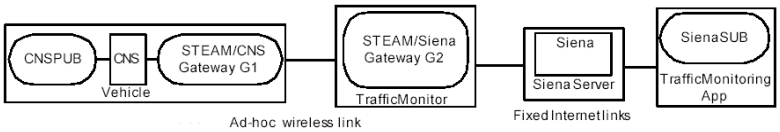

Figure 1 FES example

The first application, identified as Vehicle in the Figure 1, is located inside a moving vehicle and sends events containing information about the vehicle identifier, position (GPS) and speed at regular time intervals using an ad-hoc wireless connection. These events are gathered by one or more fixed traffic monitor applications, noted as TrafficMonitor in the figure. These events are forwarded to a traffic control center, indicated by SienaServer, using fixed communication connections. The delivered events and the information that they contain will be used by the final application, named TrafficMonitoringApp, in order to determine if the vehicle is circulating at an adequate speed or in the right direction.

The main problem with this scenario is that each application is using a different event system for disseminating events and that each application is implemented in a different programming language. Concretely, the Vehicle application is running a CORBA ORB implemented in Java. This application uses the CORBA Notification Service CNS to send location events to the nearby traffic monitor applications acting as event producers. Each instance of the second application type, the TrafficMonitor, runs a C++ application and the STEAM event system. This application receives STEAM events from the Vehicle application and sends them to the control center using the Siena event system. Therefore, this application acts both as an event consumer and as an event producer. Finally, the third application, the traffic control center, is implemented in Java and uses a Siena client connected a Siena server. The Siena server catches the events produced by TrafficMonitor applications acting as an event consumer.

Now we will explain how events are disseminated from the Vehicle application to the traffic control center application. In the first place, the Vehicle application sends a CNS event. This event is sent to other CNS enabled applications in the environment. Additionally, this event is sent through a gateway element identified as G1 in Figure 1.

This element converts the event format from CNS to FES and from FES to STEAM and delivers the adapted event to the second application. Then, the TrafficMonitor application receives the STEAM event and process it determining that it must be sent to the control center through a Siena server. In order to achieve this, the STEAM event is converted to the FES representation by the G2 gateway shown in Figure 1, and encoded again as a Siena event to be delivered to the Siena server. Finally, the Siena server receives the event and delivers it to all the Siena clients interested in it, in our case the TrafficMonitoringApp application.

4. The AOPAmI Framework

As we have indicated at the beginning of this paper, we have extended the AOPAmI framework in order to implement a FES. The AOPAmI platform (Aspect Oriented Platform for AmI) is an aspect-oriented middleware (AOM) platform. An AOM alleviates much of its complexity by allowing that concerns such as communication, coordination, location, persistence and security that crosscut the application to be modularized. This modularization hence facilitates the system evolution and makes it more robust to accommodate new application requirements. In AOPAmI we put special emphasis in addressing the evolution and adaptation issues.

Aspect technologies separate and encapsulate crosscutting concerns in modules called aspects. Aspects can only be invoked at some well defined execution points inside components called join points (e.g. component creation, disseminating a message or an event). The aspects code is weaved into the components that are crosscut by the aspect obliviously from the point of view of the components. The information about which aspects have to be weaved into a component and when, is specified in an aspect language using a set of composition rules. In our approach the aspect language is a Domain Specific Language or DSL for AmI applications (AOPAmI-DSL) that defines the platform architecture as is shown in [14]. The file holding the configuration of the AOPAmI platform, as is shown In Figure 2, is parsed by the platform when it is started. After this phase the platform instantiates the required base elements, identified by a series of InstantiateBeseElent tags in Figure 2, which we explain later. This configuration information is internally stored by the platform so it can be changed at runtime. We call base elements to both components and aspects. The difference between them is the way in which they are composed in the application. On the one hand, aspects modify the

component's normal behaviour by intercepting and modifying the disseminated events. On the other hand, components are not conscious about the existence of aspects that change their behaviour. In Addition, base elements can take on both the component and the aspect role in the application.

```
<platformArchitecture>
  <baseElements>
    <baseElement role="gateway">
      ...
      <impls selected="default">
        <impl id="default"
              mainclass="STEAM.STEAMGateway"/>
      </impls>
    </baseElement>
    <baseElement id="Coord">...</baseElement>
    <baseElement id="Location">...</baseElement>
    <baseElement id="Communication">...</baseElement>
    <baseElement id="CNStoFES">...</baseElement>
    <baseElement id="FEStoSTEAM">...</baseElement>
  </baseElements>
  <properties>
    <property id="deviceID" type="String" value="Vehicle"/>
    ...
  </properties>
  <compositionRules>
    <compositionRule when="EVENT" coordElement="Coord">
      <from><baseElementRef role="gateway"/></from>
      <events><event id="sendEvt"/></events>
      <composition>
        <evaluation>
          <concurrent><role name="Location"/></concurrent>
          <concurrent><role name="CNStoFES"/></concurrent>
        </evaluation>
      </composition>
    </compositionRule>
    <compositionRule when="BEFORE_SEND">
      <from><baseElementRef role="Coord"/></from>
      <to><baseElementRef role="Communication"/></to>
      <events><event id="sendEvt "/></events>
      <composition>
        <evaluation>
          <concurrent><role name="FEStoSTEAM"/></concurrent>
        </evaluation>
      </composition>
    </compositionRule>
    ...
  </compositionRules>
  <instanciateBaseElements>
    <instanciateBaseElement role="Coord"/>
    <instanciateBaseElement role="Location"/>
    <instanciateBaseElement role="CNStoFES"/>
    <instanciateBaseElement role="FEStoSTEAM"/>
    <instanciateBaseElement role="Communication"/>
    <instanciateBaseElement role="gateway"/>
  </instanciateBaseElements>
</platformArchitecture>
```

Figure 2 AOPAmI Architecture example

In AOPAmI, the relations between base elements are defined outside their source code, in the AOPAmI-DSL as a set of rules. Two of these rules are shown in Figure 2 enclosed by the compositionRules tag. As a consequence, the base elements do not maintain direct references among themselves. Therefore, we can provide different implementations for the same role name in the application in order to address the device specific requirements such as memory and execution constraints. For example, we can provide several different implementations of a base element that model different communication technologies such as WiFi or Bluetooth and refer to it using the same role name. In Figure 2, it is shown the definition of a gateway base element (baseElement tag) that provides only one implementation impl tag of this element. In order to add new implementations of this base element, we simply add new impl tags and identify then using the id attribute. Finally, we need to change the selected attribute in the impls tag in order to reflect which implementation will be used by the platform.

Using the AOPAmI platform we are able to modify the event model configuration depending on the application requirements and on the hardware restrictions. We can adapt this configuration even at runtime without stopping the application or modifying a single line of code. For example, we can change the set of event systems within the FES, adding or removing them from the application, or decide to which ones the system will use to disseminate events by simply modifying the composition rules shown in Figure 2. We can also accommodate all the different event types (untyped, typed and structured) to the AOPAmI event representation without changing the system implementation. Using AOPAMI, we are able to change the event propagation model if the adapted event system supports more that one. For example, we can model the push and pull event propagation models in the CNS event system using base elements. Switching between the two models will be as simple as changing the selected implementation of the required base elements and some rules. We can add new features to event models such as content and subject filters, real time restrictions or security by adding the adequate base elements and rules. Another advantage is that we are able to add new features incrementally to the system. Moreover, these features can be enabled or disabled in the application whenever it is needed if they are orthogonal.

5. AOPAmI and Federate Event Systems

In this section, we show how we have modelled the original FES application using the AOPAmI platform, the challenges that we have faced and the advantages derived from this implementation. In order to implement the distributed application presented in section 3 using AOPAmI, we have to implement two different applications.

5.1. Adapted Applications

The first application, the Vehicle application shown in Figure 3, is modelled in Java using JacORB [15],

which provides a CNS implementation. As JacORB and AOPAmI are implemented using Java, this adaptation is as easy as instantiate a new java object.

The second application, shown in Figure 3, is in charge to receive STEAM events and to send Siena events to the traffic control center. This application is implemented in C++ and, thus, we need an integration technology such as JNI Java [16] in order to create a bridge between both languages. But, two limitations arose from this solution.

The first limitation is that the target device must support a Java Virtual Machine (JVM). The second one is that only primitive Java types are supported as parameters and return values for the functions written in C++. Thus, in order to do not overload the AOPAmI platform with superfluous methods, we created a java class that acts as a launcher for the AOPAmI platform. This class implements all the methods needed by the TrafficMonitor application to disseminate events and to perform all the necessary parameter transformations.

5.2. Application Base Elements

Now we are going to describe the set of base elements used by the applications and the functionality that they model. In both applications we have followed the same model, but the base element implementations differs depending on the application's purpose. Figure 2 describes the base elements used by the Vehicle application using a series of baseElement tags.

The most important base element in both applications is the gateway. This element models the object that the application will use to disseminate events using the FES. We have two gateway implementations in the applications. The STEAMGateway, which is able to accept STEAM events and the SIENAGateway, which is able to accept Siena events. The definition of the first one can be seen in Figure 2. Notice that we use the gateway role name to refer to this element through the rest of the architecture description instead of using the real class name. For example, it is used in the first composition rule on Figure 2 to indicate that the rule will be applied only if the element throwing the event (from tag) is the gateway element. Notice that this base element is usually composed with other base elements as a component because it does not modify the normal application behavior as opposed to other base elements that we describe later. This composition is reflected in the composition rules that describe the platform architecture.

After the gateway elements, we found the adaptors. We need a pair of them to perform the event adaptation between a specific event system and FES and vice versa. Concretely, in the first application we found two of these named CNStoFES and FEStoSTEAM as it is shown in Figure 3. The first one transforms the CNS events received by the STEAMGateway base element to the FES event format. The second one transforms the FES events into the STEAM format using conversion Table 1. Other data associated to the event, as for example the event type, are also adapted automatically from one event system to another. Notice that these base elements are usually composed with other base elements acting as aspects, because they modify the normal application behaviour. Notice also that how base elements are composed is completely described in the composition rules shown in Figure 2 and that as a consequence; base elements have not knowledge of how they are being composed by the platform.

Table 1. FES to STEAM Conversion Table

AOPAmI FES Type	STEAM parameter Type
java.lang.String	S_EventParameterDeclaration.S_STR
java.lang.Integer java.lang.Short	S_EventParameterDeclaration.S_INT
java.lang.Float java.lang.Double	S_EventParameterDeclaration.S_DBL
JavaSLSLocation	S_EventParameterDeclaration.S_POS
java.lang.Long	S_EventParameterDeclaration.S_TIM
java.lang.Boolean	S_EventParameterDeclaration.S_INT
java.lang.Object	Not Supported by STEAM

If we examine carefully conversion Table 1, we notice the adaptation problem that arises here. The problem is that STEAM does not support all possible parameter types provided by CNS, for example Boolean or Object values. The opposite it is also true, because STEAM defined a local S_POS type that it is not supported directly by CNS. As a result, in some cases we must perform some parameter adaptations. For example, in AOPAmI we have defined a class called JavaSLSLocation that models the S_POS type. Other adaptation is from the CNS numeric parameters to the FES numeric types (Integer and Double). In this case some loss of precision may occur when adapting the numeric values. Finally, the conversion of Boolean values is an extreme case. We have no other choice that to convert them to integers. This is needed because STEAM does not support Booleans. Additionally, we need to remember the name of these parameters in order to convert then back to CNS.

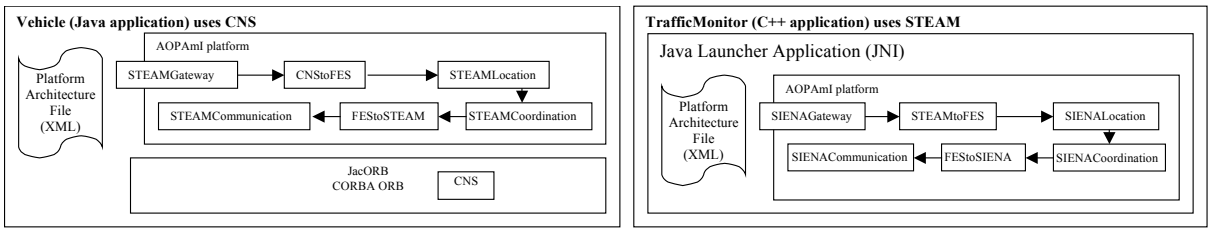

Figure 3 Vehicle and TrafficMonitor Applications

The second application uses another pair of adaptors named STEAMtoFES and FEStoSIENA that are composed similarly to the two previously presented. The conversion table from STEAM to FES is the inverse of Table 1. Finally, the conversion between FES and Siena is straightforward because there is a direct correspondence between Siena and FES formats.

The following important base elements used by the applications is the locator, named STEAMLocation and SIENALocation and identified as Location role name in Figure 2. These elements maintain a list of available STEAM and SIENA devices in the environment. When an event is disseminated through the STEAMGateway, this list is added to the event as a property that is used by the platform to send the event. Notice that we have composed this base element with the gateway element, as an aspect, using a rule instead of providing a hardcoded implementation of this property inside the coordination or gateway elements. Doing this, we provide a more flexible implementation and we make use of a platform mechanism to add information to the event as needed. This information can be used afterward by the coordination element to process the event throw by the gateway base element.

One of the most important base elements in any AmI applications is the coordination base element. Each AOPAmI application defines and uses one of these events to process the events received by the platform. STEAMCoord and SIENACoord are our example coordination elements identified in Figure 2 as the Coord base element. These elements can be implemented using a transition diagram or a hard coded decision tree. This decision is up to the programmer and depends on the device capabilities. In the examples we used the hardcode version because the coordination model is very simple. We only need to redirect the events to all the devices in the environment indicated in the property added by the Location base element. Notice that when an event is thrown by a base element and a composition rule is applicable, then the platform always evaluates a coordination element. An example of this is shown in the first evaluation rule on Figure 2 where the coordination element, indicated by the coordElement attribute, is evaluated as an aspect. Notice that this element can be used as a normal component when it sends messages to other base elements. This behaviour is depicted in the second composition rule (see from tag) shown in Figure 2.

Finally the last base element used by the applications is the communicator, identified using the Communication role name in Figure 2. Each application defines and uses one of such elements, identified as STEAMCommunication and SIENACommunication. The functionality of these base elements is sent the events to the target devices and usually they are composed as components in the application.

5.3. The Application Execution

After explaining which base elements are defined by the applications, we will show how they are used.

Firstly, when the Vehicle application is started, it instantiates and configures the AOPAmI platform using the platform architecture file shown in Figure 2. This file describes the base elements used by the application, as is shown in Figure 3. After AOPAmI has instantiated all the required elements, indicated by a series of instantiateBaseElement tags, the Vehicle application obtains a reference to the STEAMGateway base element. This object is equivalent to the gateway G1 object in the original example and it is used to disseminate events using the AOPAmI platform.

When the application sends a CNS event, this event is passed to the gateway base element. The gateway generates an AOPAmI event that it is intercepted by the platform. The platform checks for rules that can be applied on this situation. In this case a rule, the first rule shown by Figure 2, is found. We determine that the rule is evaluated only when an event is thrown because of the EVENT value assigned to the when attribute in the compositionRule tag. This rule also indicates that when a sendEvent event (events tag) is thrown by the gateway base element (from tag), then the platform will compose this base element with the Location and the CNStoFES base elements (evaluation

tag). The base elements will be evaluated as aspects in the order indicated by the rule, and they will probably modify the event contents and properties. Finally, the rule states that a coordination base element, indicated by the coordElement property of the rule, will be evaluated. This last base element corresponds in our example to the Coord element.

At the application level, by applying these two aspects we achieve two goals. Firstly, the platform adds a list of available STEAM devices to the event using the Location base element. And secondly, the platform converts the event format from CNS to STEAM using the CNStoSTEAM base element. Finally, after that, the coordination element (STEAMCoord) is evaluated the modified event. This element will decide to which STEAM devices send the event using the information provided by the event and by the platform. As a consequence, none, one or more messages are generated. Note that a message is an event, which has a destination. In our platform, this destination is other element. In this case, the destination is the STEAMCommunication element.

The new message is sent to the platform and the platform checks for the applicable rules. In the example describe by Figure 2, a rule is found that states that when a message sendEvent (events tag) sent by the Coord base element (from tag) to the Communication element (to tag) is found. Then, before the message is delivered to the Communication element (the attribute when takes the BEFORE_SEND value in the compositionRule), the message will be composed with the FEStoSTEAM base element.

At the application level the platform adapts the event format from FES to STEAM using the FEStoSTEAM element as an aspects that modifies the normal element composition. Finally, the event is delivered to the STEAM device or devices selected by the coordination element using the STEAMCommunication element.

The second application works similarly to the first one. After receiving a STEAM event, the TrafficMonitor application decides to forward this event using the SIENAGateway base element. But in this case we made the transformation from STEAM to SIENA event format. Finally, the event originally generated using a CNS event system is delivered and processed by a Siena event system.

Note that in both previous examples it is not necessary that the applications receive and handle events coming from other event systems. This functionality can be easily implemented in AOPAmI by adding the appropriate base elements and composition rules. For example, to receive STEAM events in the Vehicle application, we only need a STEAMtoFES and a FEStoCNS elements and a rule to compose them. This composition will take place when the STEAMCommunication element receives a remote event from other STEAM application.

6. Conclusions

This work has investigated the possibility and benefits of applying the AOSD paradigm using the AOPAmI platform to develop a FES. Starting from the example and the set of properties identified in the work of [9] and [10], we have been able to implement a functional AOPAmI FES prototype. This prototype has combined three different event systems namely Siena, STEAM and CNS establishing a communication channel among them.

The use of AOPAmI has clearly added some benefits to the FES application, such as adaptability, modularisation and reusing. A consequence of this is that using AOPAMI, we will be able to extend this system by identifying new event system properties not considered previously. These properties then can be modelled and integrated in the FES as new base elements and rules adding even more functionality to existing and new applications.

Additionally, both the base elements and the platform architecture definitions developed for this example can be reused in other applications by simply adapting the platform configuration to these systems. Indeed, we reuse the Platform Architecture file shown in Figure 2 in both the Vehicle and the TrafficMonitor application examples. Moreover, by modifying the platform architecture file, we can develop more complex event system federation configurations adding new functionality to existing applications.

An additional issue that was raised when adapting the original application was that the AOPAmI platform was originally designed to develop J2ME applications. But, when using Siena and JNI we where forced to create an extended version of the platform able to use the J2SE features due to the requirements of these event systems.

Finally, we have shown that AOPAmI is able to deal with the integration issues of different programming languages (Java and C++) in an elegant way. In contrast, other AmI platforms such as PCOM [17] or EMI2 [18] only consider a fixed set of technologies and development languages and applications are hardcoded, which limits the application development.

As future work, we intend to integrate other event systems in our FES modelling additional properties as new base elements and refine the already developed

ones to improve their reusability. Additionally, we are working on a set of tools to automate Platform Architecture creation, test and verification.

7. References

[1] ISTAG web site: http://www.cordis.lu/ist/istag.htm

[2] IBM Pervasive Lab web site: http://www-128.ibm.com/developerworks/wireless/library/wi-pvc/

[3] MIT Media Lab web site: http://www.media.mit.edu/

[4] J. Bravo, and X. Alamán, Ubiquitous Computing and Ambient Intelligence, ISBN:84-9732-442-0, Thomson, 2005.

[5] V. Issarny et Al., "COCOA: COnversation-based Service COmposition in PervAsive Computing Environments", Proceedings of the IEEE International Conference on Pervasive Services (ICPS), Lyon, France, 2006.

[6] Siena web site: http://serl.cs.colorado.edu/siena/

[7] R. Meier, and V. Cahill, "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks", Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02), Vienna, Austria, 2002.

[8] Object Management Group, CORBAservices: Common Object Services Specification - Notification Service Specification, Object Management Group.

[9] R. Meier, and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems", The Computer Journal, vol. 48, 2005, pp. 602-626.

[10] C. Ryan, R. Meier, and V. Cahill, "Federating Heterogeneous Event Services," In Proceedings of the 3rd International workshop on Distributed Event-Based Systems (ACM/IEEE ICSE/DEBS'04), Edinburgh, Uk, 2004.

[11] Aspect Oriented Software Development (AOSD) web site: http://www.aosd.net

[12] L. Fuentes, D. Jimenez and M. Pinto, "An Ambient Intelligent Language for Dynamic Adaptation", In Proceedings of Object Technology for Ambient Intelligence workshop (OT4AmI), Glasgow, Uk, 2005.

[13] L. Fuentes, D. Jimenez, and M. Pinto, "Development of Ambient Intelligence Applications using Components and Aspects", Journal of Universal Computer Science, Volume. 12, Issue 3, 2006, pp. 236-251.

[14] L. Fuentes, and D. Jimenez, "An Aspect-Oriented Ambient Intelligence Middleware Platform", Proceedings of 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC), Grenoble, France, 2005.

[15] JacORB web site: http://www.jacorb.org/

[16] JNI web site: http://java.sun.com/j2se/1.5.0/docs/guide/jni/

[17] C. Becker, et Al, "PCOM – A Component System for Pervasive Computing", 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom'04), Orlando, USA, 2004.

[18] D. López et Al., "EMI²lets: A Reflective Framework for Enabling AmI". I Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI'2005), Thomson, ISBN 84-9732-442-0, Granada, Spain, 2005.