Journal of Universal Computer Science, vol. 15, no. 9 (2009), 1945-1969
submitted: 15/8/08, accepted: 24/5/09, appeared: 1/5/09 © J.UCS

Supporting Awareness in Groupware through an Aspect-Oriented Middleware Service

Rita Suzana P. Maciel
(Faculdade Ruy Barbosa, Universidade Estadual da Bahia, Bahia, Brazil
ritasuzana@gmail.com)

José Maria N. David
(Faculdade Ruy Barbosa, Bahia, Brazil
jmndavid@gmail.com)

Michel Ridwan Oei
(Faculdade Ruy Barbosa, Bahia, Brazil
mitcheloei@gmail.com)

Adriano Augusto de Oliveira Bastos
(Faculdade Ruy Barbosa, Bahia, Brazil
adrianossa@gmail.com)

Leandro de Oliveira Menezes
(Faculdade Ruy Barbosa, Bahia, Brazil
cariocaleo@gmail.com)

Abstract: Solutions have been proposed to support awareness in groupware. Frequently, this requirement is fulfilled by similar functionalities that are implemented in different modules within these collaborative applications. These solutions usually represent crosscut concerns related to awareness by the use of object-oriented programming. As a result, tightly coupled components are generated as well as code redundancy and scattering. Flexibility and agility, related to awareness maintenance and evolution can be considered a challenge. This paper presents an awareness service named Aw2SOA, which was developed to support awareness functionalities in a Web-based Groupware Service-Oriented Architecture (WGWSOA) environment. The WGWSOA infrastructure is based on middleware services for collaborative applications. Aw2SOA is a middleware specific service which was implemented according to service-oriented architecture (SOA) principles and aspect-oriented programming (AOP) concepts. Case studies were carried out focusing on service integration activities as well as using groupware application development in order to evaluate this solution.

Keywords: awareness, SOA, groupware, middleware services, aspect-oriented programming
Categories: H.3.4, H.5.3

1 Introduction

Complex activities previously carried out by individuals have started to be done collaboratively - in groups - due to increasing demands for efficiency, productivity

and quality. Such groups are not always co-located at the same site. Consequently, they require complex remote interactions which may result in inefficiency. Groupware technologies support remote interactions to obtain an efficient cooperation process [Ellis *et al.* 91].

Making groups cooperate to reach a common goal is not an easy task. One of the problems that hampers collaboration, for example, is the lack of context among participants. To solve this, context can be provided by awareness, which consists of knowledge activities providing the context for individual activities [Dourish and Bellotti 92]. Social awareness, for instance, supplies information about the presence of each group participant, or about task-oriented when using notification results in order to provide information about the state of shared artifacts [Prinz 99].

Events such as sending messages to users, task conclusion or artifact edition in a shared workspace and validation policies are present at several points of groupware design. Such events are usually planned to support group communication, coordination and cooperation. Most can be considered crosscutting concerns in terms of functionalities, which are related to the events and are implemented in different modules (for example, components, objects and services), each with different abstraction levels. Even though some non-functional requirements are considered (e.g. portability, abstraction and scalability), it is possible to find similar functionalities implemented in different modules and related mainly to the functional requirement “provide awareness”. As a result, activities related to both maintenance and evolution requirements tend to complicate the effort involved to find different software elements to be altered.

Attempting to represent crosscutting concerns with the use of the object-oriented paradigm results in a tightly coupled code, as well as in the spread of functionalities in several parts of the application. In order to solve this issue as well as to modularize the crosscutting concerns and facilitating both code maintenance and understanding, aspect-oriented programming [Kiczales 97] [Kiczales 01] has been proposed.

A service-oriented approach has been deployed in order to enhance application development in several domains. In [Zaupa et al. 08] a process to support web services development in service-oriented architecture (SOA) context is proposed. In order to support mobile workers to collaborate on a solution based on service concepts is proposed in [Neyem et al. 08]. This solution aims to support ad-hoc network integration as well as to enhance interoperability among different collaboration scenarios. The complexity of interactions which have to be fulfilled by groupware concerning different collaboration modes (synchronous and asynchronous) is considered through notification service in [Geyer et al. 08]. Therefore, the service-oriented approach indicates a probable solution to support complexity that collaborative applications require.

The WGWSOA (Web-based Groupware Service-Oriented Architecture) is an architecture based on middleware services for collaborative applications [Maciel and David 07]. It enables interoperability between groupware developed according to the same or distinct middleware platforms. The Service-Oriented Architecture (SOA) approach is used to implement essential functionalities of applications according to service concepts which encapsulate application logic in different business contexts [Erl 05]. In this architecture, specific coordination, cooperation, communication and

awareness services were implemented. They support groupware design and implementation through composition and reuse, among other SOA principles.

According to this architecture, services can be designed as loosely coupled units. However, the use of SOA and object-oriented paradigm were not sufficient to fulfil awareness requirements in WGWSOA, considering the previously mentioned collaboration needs. Because of this, awareness support was only partially fulfilled, as a result leading to tightly coupled collaboration services.

This paper presents an awareness service named Aw2SOA [Bastos *et al.* 08], which was developed in order to support distributed groupware design. It is a middleware service that was implemented according to the SOA principles and aspect-oriented concepts. The paper is organized as follows: in Section 2, we present some related works. Section 3 describes the WGWSOA architecture, which supports groupware design through a middleware platform. The proposed solution is discussed in Section 4 and its evaluation follows in Section 5. Finally, Section 6 concludes the article.

2 Related Work

Awareness in groupware applications needs to be widely supported. In [Gutwin et al. 08] for example, some requirements to fulfill in distributed groupware concerning informal collaboration are presented. Groupware design needs to be considered differently, primarily in order to provide awareness appropriately.

Awareness services have been proposed in groupware literature. Some deploy technologies which support component-based development, which do not enhance interoperability among collaborative systems platforms accordingly. Other services have been designed as plugins to be adapted to the existing groupware, but again, interoperability among these platforms has not been properly supported.

Some solutions have been suggested in order to consider awareness in distributed collaborative applications. In Ariane [Vieira *et al.* 04], awareness support is implemented to monitor the actions generated by applications when manipulating artifacts that persist in SGBD. Overall application events are not monitored by Ariane, except for the actions carried out by persistent artifacts, removing from the application the function of notifying events considered as relevant to awareness support.

BigWatcher [Pinheiro *et al.* 03] is a framework designed to monitor and contextualize participants' work in group. Its main goal is to supply event awareness in the past, which can be recorded as well as presented in a flexible way. As a framework, this mechanism does not implement all the necessary functionalities. It requires that extensions should be done in every application needing their service deployment.

NESSIE [Prinz 99] is an environment designed to support awareness. This service offers a generic infrastructure for every application, together with sensors and indicators to notify events. It is based on the idea that through a server information about collaborative activities is captured by sensors. Sensors, in general, are associated with actors and shared objects which are parts of the activities. In the NESSIE environment it is possible to integrate real world indicators to foster

awareness about participants' presence in remote activities, independent of the screen limits.

Events in NESSIE do not need to be automatically created by sensors associated with a specific work situation. They can be manually created, and are used to both announce meetings and distribute relevant news. Sensors capture every event and configurable indicators are related to event consumption. The form in which the events are captured represents a challenge. Such events are generated exclusively when a client's change occurs, or they have to be sent directly to the NESSIE server via HTTP and CGI scripts.

AREA/POLIAwaC [Fuchs 99] is an integrated and global service designed to manage synchronous and asynchronous awareness information. Basic notification mechanisms are offered by AREA and can be used by designers to make actions visible to each participant in the environment.

Through a client's model specification, it is possible to support awareness among applications. Therefore users can be notified about other users' activities in different applications. The model consists of an interested profile (event description, scope, work situation, for example) and a privacy requirements list. The list names the actors who can receive a certain event.

The AREA notification is sent to the application through an event description and a value representing the intensity of the event. The application has to consider this information and to present it to the user adequately and passively, according to his/her interest.

InterDoc (a reference architecture for Interoperable services in collaborative Document writing environments**)** [Maciel et al. 05] awareness service uses filters to capture application data and to send them to other servers, establishing an intermediary layer across the data flow. Despite the fact that the server is transparent for the application, it has facets and receptacles defined to support InterDoc servers (Users, Activities and Planning). However, in order to add new servers, modifications in the awareness server must be implemented.

If the presented approaches were used in a SOA, they would generate a tightly coupled solution between the applications and services. Consequently, they would not fulfill one of the basic principles of SOA: loose coupling.

In Aw2SOA, awareness is considered a crosscutting concern that has to be implemented independently from the application. Using AOP concepts, events are captured apart from other functionalities and can be defined in a simple manner by the application. For the notification process, it is suffice for the service to follow a protocol, sending any additional message or implementing any extension to use the service not being necessary.

3 WGWSOA: Web-based groupware service-oriented architecture

Distributed collaborative systems are applications that demand complex requirements which are difficult for current groupware architectures to fulfill [Maciel and David 07]. Such complexity is also related to the diversity of requirements of distinct environments that have to capture not only the semantics of the activities, but also the

information and the most common application procedures. Furthermore, there are some limitations regarding the design and implementation as well as the evolution of groupware functionalities in a web environment. Most limitations could be overcome if their functionalities were defined as middleware services.

Distributed groupware applications are usually built through middleware platforms support [Maciel *et al.* 2005]. In spite of manipulating objects (documents, meetings agenda, discussion lists, etc.) with the same semantics, it is not possible for these applications to access objects and data from another application, even though they are built on the same middleware platform. The complexity of this scenario increases when we need to interoperate collaborative applications built on distinct middleware, e.g. J2EE, CORBA and .NET. Due to the amount and diversity of solutions therefore, distinct groupware is not usually interoperable. The absence of interoperability is a problem not only for users but also for designers of these applications.

However, the existence of a great diversity of solutions makes interoperability implementation high cost task, due not only to the hardware and software platform heterogeneity that support them, but also to the amount of specific solutions (protocols, APIs, components, etc.) that some environments make available [Dewan and Sharma 99]; [Tietze 01]; [Li *et al.* 03]. In practice, for each new environment inserted in the collaborative project, it can be necessary to define a new development process to provide interoperability across these environments. The amount of specific solutions to fulfill the requirement is a challenge for every developer given the implementation effort involved.

Middleware solutions have also been proposed to solve problems related mainly to interoperability across distributed applications. For such solutions, however, it is necessary to know not only the aspects of application structure, but also the technology deployed. Middleware offers a set of services (common and specific) that are described – and made available – through a specific interface definition language. Such services enhance communication and distribution to support distributed application development [Schantz and Schmidt 01].

Common services supply functionalities to a high independent domain application level (concurrency control, transactions management, object location and activation, for example), while the specific services implement functionalities for a specific-domain application, e.g. collaborative systems (discussion lists and forums, chat communication, group and project management).

WGWSOA [Maciel and David 07] is an infrastructure designed to provide interoperability across collaborative systems built on the same platform, or under different middleware platforms; for example, .NET and J2EE. In the same vein, it aims to offer collaboration services capable of evolving according to dynamic groupware requirements. In this context, some issues related to awareness support and the technologies deployed to develop this service will be detailed next.

As illustrated in Figure 1, the middleware architecture WGWSOA has three layers: application, middleware and persistence. In the first layer, we can find groupware applications composed of WGWSOA services. The middleware layer is composed of common and specific services. The last layer is related to collaboration elements (communication, cooperation, coordination, awareness and wrapper).

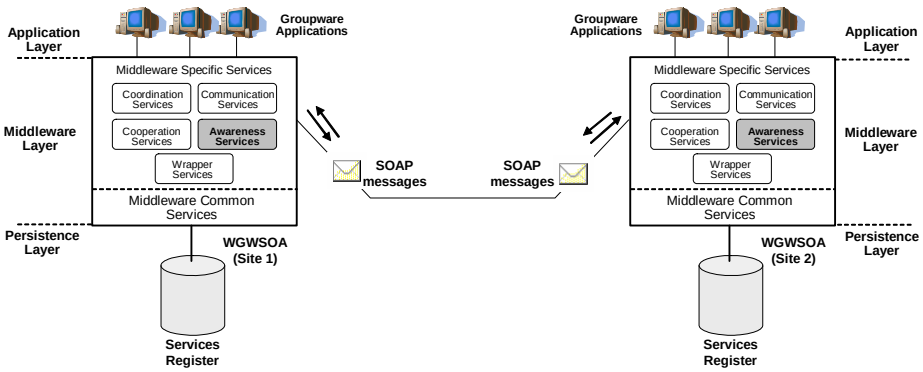

Figure 1: WGWSOA general architecture (adapted from [Maciel and David 07])

Middleware is both an execution and programming environment for services development. In order to interoperate groupware across distinct sites it is necessary to instantiate WGWSOA in each of the sites.

WGWSOA services are necessarily implemented as middleware services and their interfaces can be described with the use of IDL or WSDL languages. However, the latter can be used so that developers can take advantage of the protocols offered. Middleware services interfaces are usually described through their standards of specific languages (Interface Definition Language – IDL).

In spite of supplying programming language independence, such interfaces do not offer independence of middleware platforms. For example, J2EE/Java-RMI [Shannon 03] uses JavaIDL (Java Interface Definition Language), CORBA 3.0 [OMG 02a] uses CIDL (Component Implementation Definition Language) and CORBA 2.0 [OMG 02b] uses IDL. Additionally, each platform uses a specific communication protocol across their services, such as Java-RMI (Remote Method Invocation) for J2EE, and IIOP for CORBA. Thus, it is important to point out that applications built using the same architecture (or from the same framework) but implemented in distinct middleware are not always interoperable. It is necessary, therefore, that the architecture has also fulfilled this non-functional requirement.

Another important aspect is that, even though the services are implemented on a single middleware platform, the firewalls of internet sites do not allow these communication protocols to be used across web applications. Given this, WGWSOA deploys both the SOAP (Simple Object Access Protocol) to communicate across distinct implementations of the architecture as well as WSDL specific services description to provide interoperability among distinct middleware platforms that adopt web-service technologies.

Solutions based on services are not recent. Service-Oriented Architecture (SOA) enhances interoperability through its principles. Among them we can mention: contract specification, loose coupling, reuse, composition, autonomy and the stateless. Thus, a service-oriented architecture, when implementing such principles, can support interoperable service construction [Erl 05].

Groupware context service-oriented architectures have been proposed to facilitate the development of such applications. In [Fonseca and Carrapatoso 06] , for example, an architecture named SAGA is described which uses a model based on web services technology to enhance groupware development. However, the proposed architecture does not use a middleware service approach (e.g. namespace and discovery services) nor considers groupware interoperability applications across distinct platforms and heterogeneous technologies.

3.1 Common and specific services in a groupware context

The use of a domain specific service approach in order to support groupware development also facilitates reuse development [Turner 03]. As illustrated in Figure 1, WGWSOA has a set of common and specific services. Regarding the common services, it is necessary that the middleware that hosts WGWSOA, e.g. CORBA or J2EE, own both the name service as well as the distribution service.

Specific collaboration services can be divided into the following types: communication, coordination, cooperation and awareness. Communication services can be divided into synchronous and asynchronous. Such services can be composed of other services and extended to support different collaboration modes. For example, a Chat Service can be composed of an instant message exchange service to build a more complete communication service.

Table 1 exemplifies some services that are already implemented in WGWSOA.

Specific Services		Examples
Communication	**Synchronous**	Instant Message, Chat.
	Asynchronous	Discussion groups, Discussion Forum, E-mail.
Cooperation		Repositories management (persistency), Shared workspace management (artifacts edition control, for example).
Coordination		Group management, Project management, Activities management.
Awareness		Event Notification, Filtering, Data Visualization (participation reports, data groupings, for example).
Wrapper		Protocols conversion, message packing, WGWSOA instances localization.

Table 1: Examples of Groupware Specific Services (adapted from [Maciel and David 07]

A wrapper service is responsible for communication among distinct WGWSOA instances. When an application on a certain site needs to execute a service at another site, it sends a request message over the web. To do this, it converts the message into a protocol that supports interoperability and in this case SOAP is the protocol.

4 Aw2SOA Awareness Middleware Service

Awareness can be defined as an event (event-based awareness) capable of occurring in a groupware and broadcasted to every interested participant. Events such as messages sent to users, task conclusion, artifact edition in a shared environment or policy validation can be found in most groupware designs. These events are related to the communication, coordination and cooperation activities, which can be characterized as a crosscutting concern.

Suppose that a collaborative application has interest in events related to a message exchange service, represented by the rectangle in Figure 2, and to the activity management service, represented by the triangle in this same figure. The first event possesses information about who sent the message and the targeted users, while the second relates an activity to the person responsible. Both occurrences are related to event generation, but in distinct contexts.

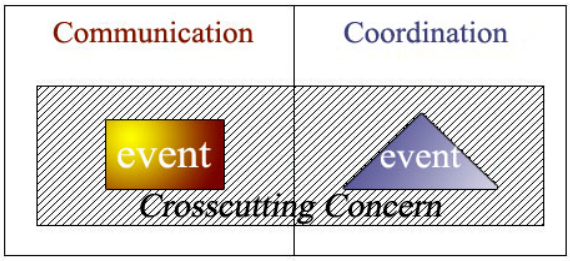

Figure 2: Events in different contexts

Aw2SOA (Awareness to Service-Oriented Architecture) is a specific service of WGWSOA. Its main goal is to support awareness about events launched by WGWSOA services as well as any groupware applications that uses WGWSOA in order to interoperate with each other. The Aw2SOA service gives support to actions that happen in the application as well as in every WGWSOA service not only to promote communication across such actions, but also to propagate “what”, “where”, “who” and “when” information, regarding generated events. This type of information can be asynchronously processed through WGWSOA database queries, as well as synchronously, through its notification service [Bastos et al. 08].

The process adopted in the awareness service presented here enhances information launched as events between applications and other services that are part of the WGWSOA middleware. Groupware applications and/or any WGWSOA services that implement awareness functionalities are classified as a producer, i.e. they generate events, or a consumer, when they are interested in manipulating event information. Both Aw2SOA event producers and consumers are correlated in a profile that guides every event notification. This profile supports event exchange in the WGWSOA environment, and maintains group memory so that the information occurred can be further queried [Bastos et al. 08].

4.1 Service Specification and the Conceptual Model

To support awareness in collaborative applications, functionalities were specified to be implemented in Aw2SOA service.

Figure 3 illustrates some of the main functionalities of the Aw2SOA through a use case diagram. The actors, being the external entities using Aw2SOA, are: (i) external WGWSOA applications, or groupware applications which are running in another WGWSOA instance; (ii) any other WGWSOA specific services.

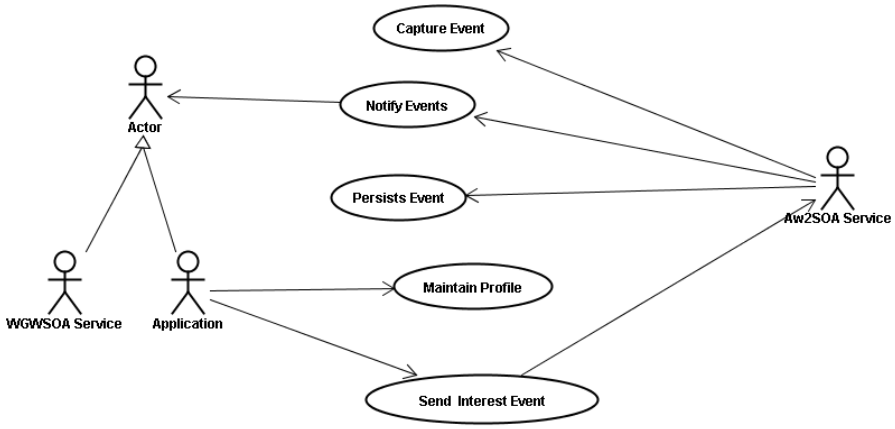

Figure 3: Aw2SOA use case diagram

The main use cases of Aw2SOA are to (Figure 3):
(i) maintain profiles in order to group all information related to the actor and event consumers and produces;
(ii) notify events, through application or services request, recent event occurrence are notified according to the consumer profile;
(iii) query events, to provide a search mechanism for past events;
(iv) capture event, intercepting events from any WGWSOA specific services method;
(v) send event of interest, regarding the delivery of information related to an event occurrence from a groupware application (related to the profile);
(vi) persist event, verifying event context according to the registered information in the profile and the storage in the WGWSOA repository.

In order to assure a loose coupling between applications and the Aw2SOA service, it was necessary to define distinct strategies for event production and a notification process for each of these actors. As a result, it was necessary to make the elements explicit which both generate and consume events. For each actor an interested profile is constructed.

The event generated register, regarding the service profile, is produced automatically at its first generation moment. This is done by Aw2SOA when registering a profile that does not exist. Nevertheless, the application profile has to be previously registered in the WGWSOA site repository (see Figure 1) by the

groupware designer. This profile register must be carefully elaborated considering that it is an agreement for distribution, notification and event persistence.

Figure 4 presents the AW2SOA conceptual model which illustrates each attribute related to the generated events. Every event is represented by its name, the actor name, the time of occurrence and the context in which it occurred.

According to Figure 4, the event is represented by Event class, containing the information about the event, i.e. the event name and its occurrence time, the actor name, which both produces and consumes events. Furthermore, this class encapsulates context elements, for example, what activity was delegated in a coordination service, or the participant name that sent the message. Actor represents the entities that generate events, while Application and Service classes can be an actor that produces and/or consumes events.

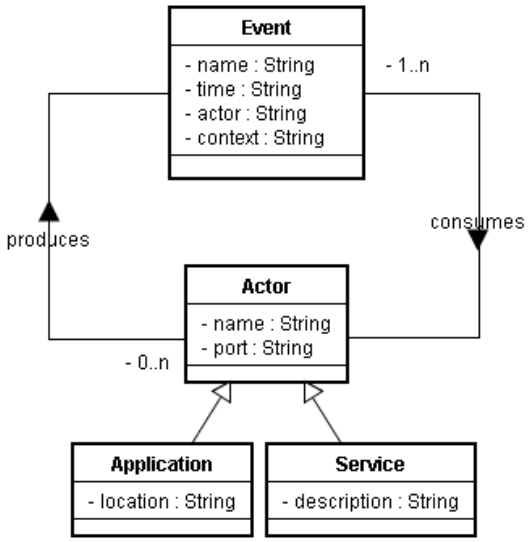

Figure 4: Aw2SOA conceptual model

While the entity attributes of the conceptual model are stored at the moment of event occurrence, the associations (consumers and producers) are defined through an interest profile. Therefore, each actor has to be associated to a profile, in which each generated and consumed event is described, as previously mentioned.

4.2 Service Architecture Overview

In order to give support awareness in collaborative distributed activities, the Aw2SOA service was composed of the following services: (i) events (ii) notification (iii) query (iv) persistence (v) profile manager and (vi) sensor. Figure 5 illustrates the Aw2SOA architecture and presents its services and their relationship.

According to the Facade design pattern, the Aw2SOA is composed of several services that make the descriptions of these services available in a single interface.

The Event service is responsible for the interest profile maintenance and supports input of every event in the WGWSOA. Therefore, this service is mandatory in order to make events perceivable by the applications developed in the middleware concepts. Its interface defines the attributes for both event creation and contextualization, such as the event name and the actor, a list of the names that identify event parameters and another list with the corresponding parameter values. The event service transforms event information into a XML (eXtensible forMat Language).

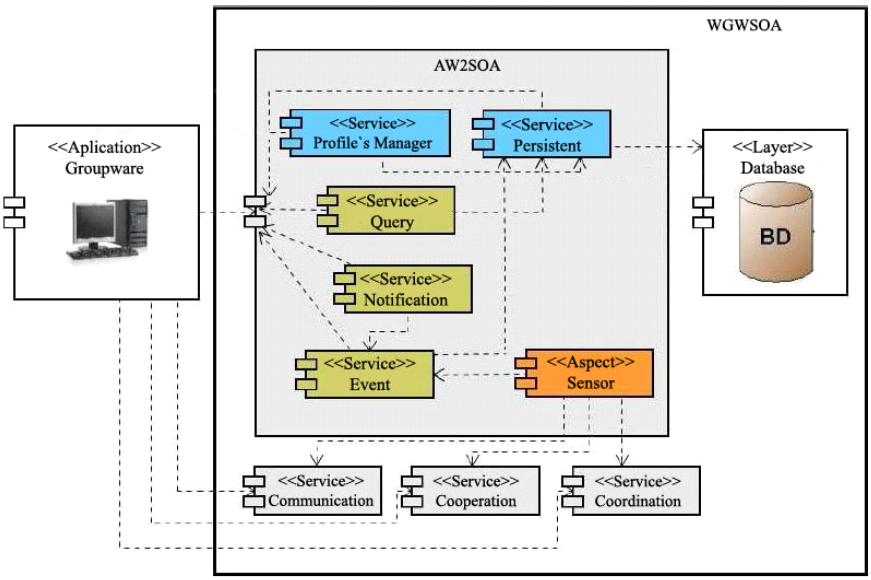

Figure 5: AW2SOA architecture overview

Information sent to the event service is validated according to the previously registered profile. This validation consists of comparing the received data with those contained in the WGWSOA database instance in order to assure the reliability of the generated XML document. If the validation process identifies some data incoherence, an error message is sent to the event actor. Otherwise, the XML document related to the event is created and temporarily stored in a recent event buffer for notification purposes. Finally, it is kept in the database so that it can be queried further.

The Notification Service provides awareness information. As a result, when applications are developed under WGWSOA concepts, they can be composed of this service. However, in order to perform this action, it is necessary to create the profile from the event selection. This service is responsible for both identifying and grouping the most recent applications or WGWSOA service events so that they can be consumed. The way used to notify follows the model “pull”, which obliges the consumer to request the event of interest so that they can be returned. From this request the profile is verified and the interested event types are obtained. Searching for the recent event is accomplished in a buffer that matches some events in

accordance with those described in the profile. After searching, the last request made by the consumer is stored in another buffer. This request will be used to avoid the same event being sent more than once. As a result, service integrity is guaranteed.

The Query Service supports other services or applications to query the WGWSOA database in order to retrieve events of interest asynchronously. It defines a way in which both applications and services could get information about events which have occurred in the past. To query it is necessary to inform only the data that will determine which the events are that should be required. These data can be the actor name, the event name or an occurrence interval, identified in the service interface description.

The Persistence service supports database access and manipulates XML files, withdrawing other services from this responsibility. While the query, notification and the sensor services are essential Aw2SOA services, the profile and persistence management services support other WGWSOA services.

The Profile Manager Service is responsible for maintaining the application profile, providing a way to create, delete and change profile data for each groupware. The profile data is kept in the WGWSOA repository of the site where the application is hosted. The profile can be used to inform which events can be consumed by other services or applications. It can also be queried through the WGWSOA administration tool.

The basic programming unit in AOP is the aspect that describes and implements non-functional application requirements through joint points of definition that indicate where a certain code should be embedded. The compilation process makes the aspects weave for the original application. Thus, sensors are implemented as aspects. Two advantages of AOP deployment are the ease of adding and removing non-functional application requirements, as well as the possibility of explicitly describing the joint points without changing the original application [Chavez and Lucena 01]. AOP deployment allows the extension of an implementation with sensor functionalities in non-intrusive way.

The Sensor service is responsible for capturing events that occur in WGWSOA specific services; it was implemented to be independent and transparent for any service. It aims to separate crosscutting awareness concerns and to generate event services. Therefore, Aspect-Oriented Programming (AOP) techniques [Kiczales 97] were deployed. This service is also responsible for registering interested profile and, after capturing events, the Event Service communicates with the WGWSOA persistence service, to save database information.

4.3 Service Design and Implementation Issues

For each event in Aw2SOA, the context could belong to different domains, e.g. (i) which activity was delegated in a coordination service? (ii) who received the message in a communication service? and (iii) which artefact name was modified in a cooperation service? To make event representation flexible enough the XML format was used. It gives the owner the responsibility of defining which information appropriately represents them. Figure 6 illustrates an event representation in XML format. According to the figure the first line contains both the version and encoding information. In the second line, the tag `<event>` is common to the every event which contains the event name, its owner and the occurrence time attributes. In line 4,

the tag `<parameter1>` represents the context description in which it occurs, and the attribute "value" contains the information related to its context.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2   <event name="Event Name" responsable="Actor Name" timeOccurence="Date and Hour">
3         <context>
4            <parameter1 value="VALUE"/>
5            <parameter2 value="VALUE"/>
6            <parameter3 value="VALUE"/>
7                        .
8                        .
9                        .
10           <parameterN value="VALUE"/>
11        </context>
12  </event>
```

Figure 6: Event representation of a XML document

The Notification service interface uses the profile name as a parameter to search for the recent events. When grouping the events that are already in a XML document, the service uses a tag `<events>` as a delimiter.

As a way of defining the launched events from WGWSOA Services, some associations were defined. The event name corresponds to the executed method name, the actor is equivalent to the package name in which the service is contained and its context is related to the method parameters. The tag `<name>` (line 10 `parameterN`) refers to the parameter names and their attribute "value" corresponds to the parameter value.

The event attributes related to WGWSOA specific services are defined by their IDLs, in which the method name corresponds to the event name and the parameters to its attributes. The sensor service makes a profile register automatically; events attributes related to groupware applications must be manually informed. As a result, developers need to register interested consumers and produce events. In order to facilitate the registration process, an application that uses profile manager service, named profile manager, was implemented.

Figure 7 illustrates a graphical interface for the event structure registration by the profile manager. The profile register process is divided into two steps: the generated events, and then the consumed events. The profile mentioned above was designed considering that the same application generating the events can also have interest in consuming them. First, it is necessary to register the generated event structures, so that the event name is identified as well as its attributes. The last step consists of registering events that the application needs to consume.

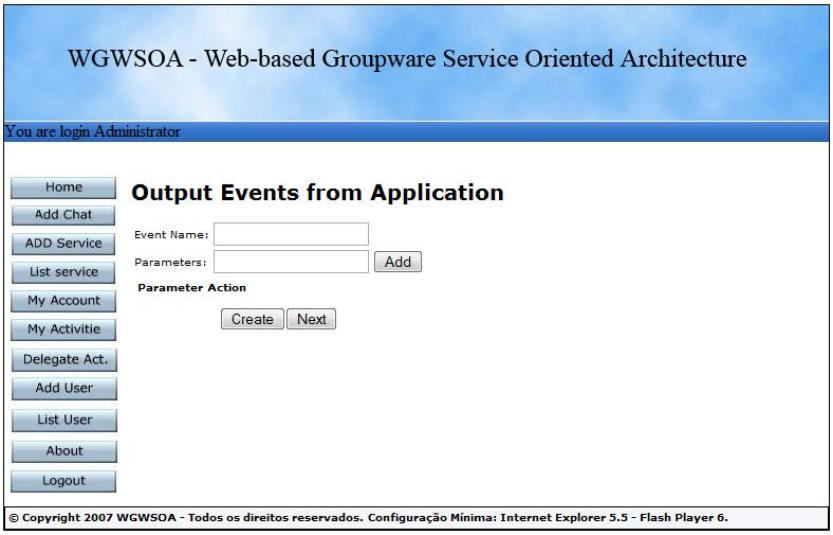

Figure 7: Event structure register

The awareness process is divided into four stages: production, storage, notification and query. In the production stage information which will be used in the next stages is produced. In the storage stage event information is stored both in a buffer and a database. In the notification, the recently occurred events are distributed among interested participants, by request. In the query, events that have already occurred can be recovered from database. Consequently, event information can be asynchronously consumed to query the database and synchronously to support the notification service.

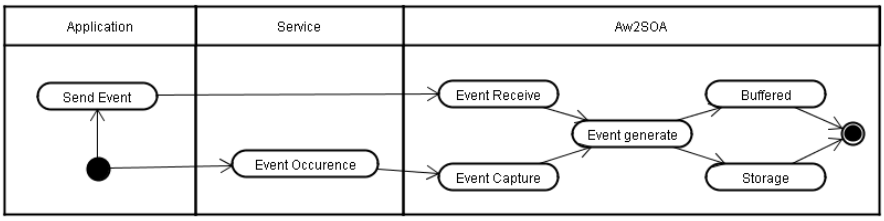

Figure 8: Activity diagram of the event production stage

Event production occurs through the next steps, according to the diagram in Figure 8.
(i) Event occurrence: represents the executed actions through WGWSOA services;
(ii) Send Event: represents the executed actions through applications;
(iii) Event Receive: represents reception action from application;
(iv) Event capture; in the services the events are captured through a sensor;

(v) Event generate validates the event according to the actor's profile generating it. After that, it is converted into a XML file (Figure 8).
(vi) Buffering: recent events are maintained in a buffer to optimize requests and to define life time limit for notification;
(vii) Storage generated events are stored in the middleware repository, which keeps the history of actions, allowing further query and recovery of information.

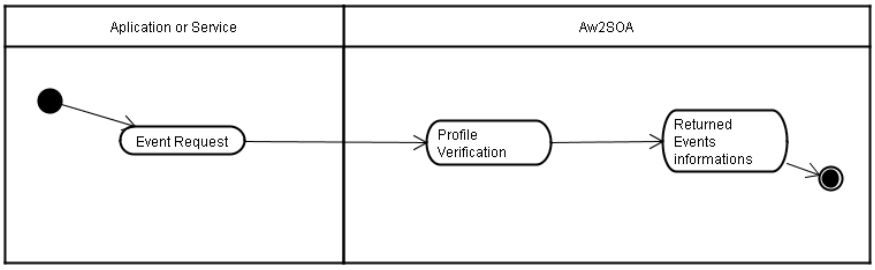

Figure 9: Activity diagram of the event notification stage

Event notification occurs as illustrated in Figure 9.
(i) Events request: consumers' events of interest can be requested directly from Aw2SOA after the profile has already been created.
(ii) Profile verification: is the consumer´s profile associated to their interest in every generated event by the actors. When the requisition of an event occurs, the profile indicates which of the recent events contained in an event buffer should be returned.
(iii) Returned Events Information: after recently generated events have been collected from the buffer they are grouped as XML document. Afterwards, they are sent to the interested users.

Finally, event consumption is carried out in the applications and/or services where the contained information could be used to contextualize group activities or to accomplish communication among every involved entity.

The technologies used for the architecture implementation were based on the J2EE platform, using DOM (Document Object Model) specification for XML manipulation. Sensor Service was implemented as aspects using AspectJ [Kiczales *et al.* 01] plug-in for eclipse platform. JPA (Java Persistence API) was used in the data persistence layer and RMI (Remote Method Invocation) technology was used to support every service implementation. For communication support across the different WGWSOA web instances, the protocol SOAP was used.

Considering the continuous evolution of middleware, a new collaboration service design should be taken into account. As a consequence, it is not possible to identify the methods that will be captured (join points) by the sensor previously. To solve this problem and also to pass the responsibility of the captured method definition on to the domain expert, Java Annotations resource was used together with the aspect. Java Annotations are metadata that can add information to the code without dependence addition as well as modification in the code logic (For example: `@Deprecated` - Annotation which indicates the method or class depreciation). Thus, a new degree of filtering not contemplated in the aspect can be defined.

```
1 public aspect AspectSensor {
2     pointcut interceptEvent():execution(public * br.frb.wgwsoa.services.*.*(..));
3     before():interceptEvent(){
4         try {
5             String methodName = thisJoinPoint.getSignature().getName();
6             Class classType = thisJoinPoint.getSignature().getDeclaringType();
7             Method method = this.getMethod(classType, methodName);
8             System.out.println("AspectSensor.before()111");
9             if (method.isAnnotationPresent(Awareness.class)){
10                System.out.println("AspectSensor.before()222");
11                this.sendEvent(thisJoinPoint, method);
12            }
13        }
14        catch(NoSuchMethodException e){
15            e.printStackTrace();
16        }
17    }
```

Figure 10: Example of Aw2SOA crosscutting concerns

To generate events WGWSOA services have to belong to the service package and their methods should contain the annotation `@Awareness`. Figure 10 illustrates `AspectSensor` which captures method occurrence through the point cut `interceptEvent` (line 2) in `before` advice (line 3) the methods that possess the annotation `@Awareness` are filtered. The necessary information for event generation is obtained through reflection and sent to the Event service (line 10).

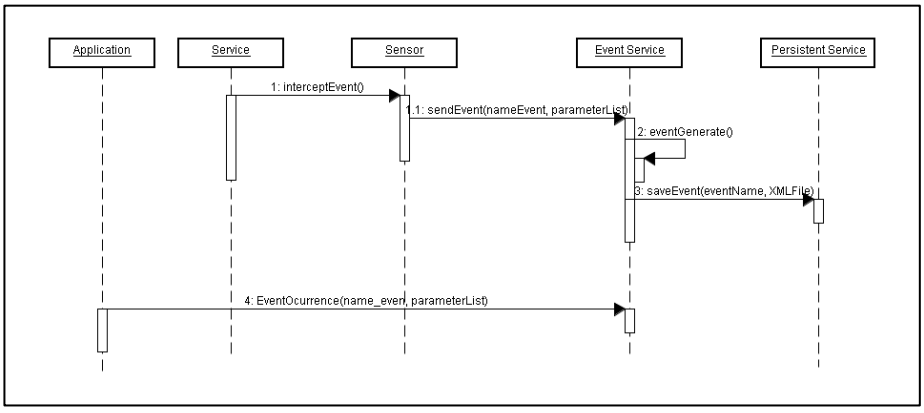

Figure 11: Sequence Diagram of the production Process

The event production process occurs as illustrated in Figure 11. According to the sequence diagram, if an application needs to send an event, it has to request to the Aw2SOA the execution of `eventOcurrence()` method. This method receives, as parameters, the event name and a value list `(parameterList)` representing the event context. From these values the event service constructs a XML document (see Figure 6) to further maintain it in a buffer and, finally, record it in a database through the saveEvent() method execution from the persistence service.

Generated events from services are captured by sensor through `interceptEvent()` pointcut. After this, the acquired values are sent to the event service through the `sendEvent()` method execution. As previously described in the application process, a XML document is constructed from the parameters sent by this method, and the event is also maintained in a buffer as well as in a database.

According to Figure 12, the notification process initiates not only from a service request but also from an application request through the `eventRequest()` method execution from the notification service. As a parameter to this method, the actor (application or service) profile name is sent. As a result, the notification service recovers from the database all the values which are associated to the profile that represents actors' interests.

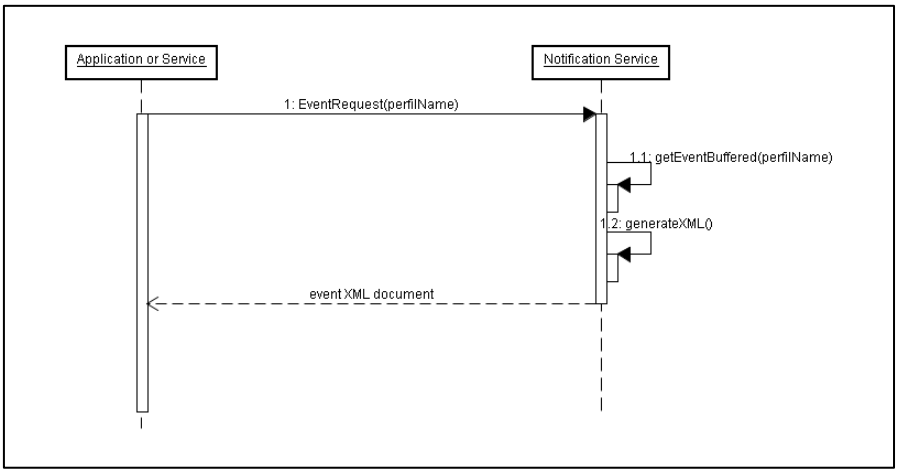

Figure 12: Sequence Diagram of the Notification Process

According to the profile information, the notification service can group recent events from the buffer. Such events are used by `generateXML()` method to construct XML documents, and return them to the actor.

5 Evaluation

Experiments were carried out throughout the distinct case studies to reflect groupware development and usage scenarios. These experiments were designed to evaluate two issues: (i) if the Aw2SOA event model can support WGWSOA services design, and (ii) if the modularization offered by the Aw2SOA for awareness support can reduce the impact in relation to every modification in other services.

The main objective of the first issue was to evaluate the event definition form, from the profile creation to its treatment in different contexts of each collaboration element (coordination and communication). A groupware developer can register his/her profile, send events to WGWSOA instances and make queries about events of interest. Meanwhile, he/she can evaluate the level of reuse, flexibility and abstraction

offered by Aw2SOA. The second issue was to identify if aspect oriented paradigm usage could enhance awareness functionality encapsulation, not changing the business logic of other services.

In order to evaluate the above issues three case studies were carried out: the first to evaluate the proposed approach in the event context generated in WGWSOA services, in the second case study the targeted context belonged to the generated events starting from a groupware application and in the third case study distinct WGWSOA instances were considered in the event generation process.

5.1 Awareness support for coordination services

The goal of this case study is to extend awareness support in four WGWSOA coordination services. These services were developed to support the measure of the following rates in a forum application (i) conflicts rate ("Conflictmeter", `Conflictmeter.setConflicts`) (ii) participation rate ("Participameter", `Participameter.setParticipations`) (iii) contribution rate ("Contributiometer", `Contributiometer.setContributions`) and (iv) impact rate ("Impactmeter", `Impactmeter.setImpacts`). According to the script described in Section 4.2 - Figure 6, the method of each service was selected in order to represent the events.

```
1   @Awareness(names={"userLogin","application","conflictUserLogin","level1"})
2   public boolean setConflicts(String userLogin, String application, String conflictUserLogin, int level1){
3       Conflictmeter.open();
4       ResultSet rs = Conflictmeter.doSelection(userLogin, application, conflictUserLogin, level1);
5       try {
6           if(rs.next()){
7               Conflictmeter.do(userLogin, application, conflictUserLogin, conflictmeterCount, level1);
8           }else{
9               Conflictmeter.Conflicts(userLogin, application, conflictUserLogin, level1);
10          }
11      } catch (SQLException e) {
12          e.printStackTrace();
13          return false;
14      }
15      return true;
16  }
```

Figure 13: Annotation in a service to measure conflict rate ("Conflictmeter")

From the identified methods, the services were added to the WGWSOA package "service", and the annotation @Awareness was inserted, see Figure 13, in order to support the method execution capture. At the moment of the first event capture, the profile related to the service generation was automatically created, and the package where the service implementation is located was defined as the actor name parameter. The attribute names were: conflictmeter, contributiometer, impactmeter, participameter.

The chosen methods had different parameters. As a result, they possessed distinct contexts for each event, as illustrated in Figure 14 from the tag <context> of the generated events.

```
<context>                              <context>
    <userLogin value="ramon"/>             <userLogin value="rodrigo"/>
    <application value="Forum"/>           <application value="Forum"/>
    <conflictUser value="paula"/>           <level1 value="3"/>
    <level1 value="5"/>                     <level2 value="2"/>
    <level2 value="3"/>                 </context>
</context>
```

Figure 14: Event context examples

The service composition was completed in approximately six hours by two developers. The Aw2SOA event model deployment allowed the removal from WGWSOA services functionalities related to the storage and information retrieval. As a result the number of lines of code was reduced by around six hundred. However, it was observed that this event model does not allow that events of two overloaded methods to be appropriately captured, because they possess the same name in the XML document.

5.2 Awareness support for communication service

In this case study, the activities were carried out through the development of a Chat service for a collaborative authoring application. The main requirements of this service were (i) to allow message exchange among users of a same room (ii) to record each message and (iii) to exhibit an on-line user list in each room.

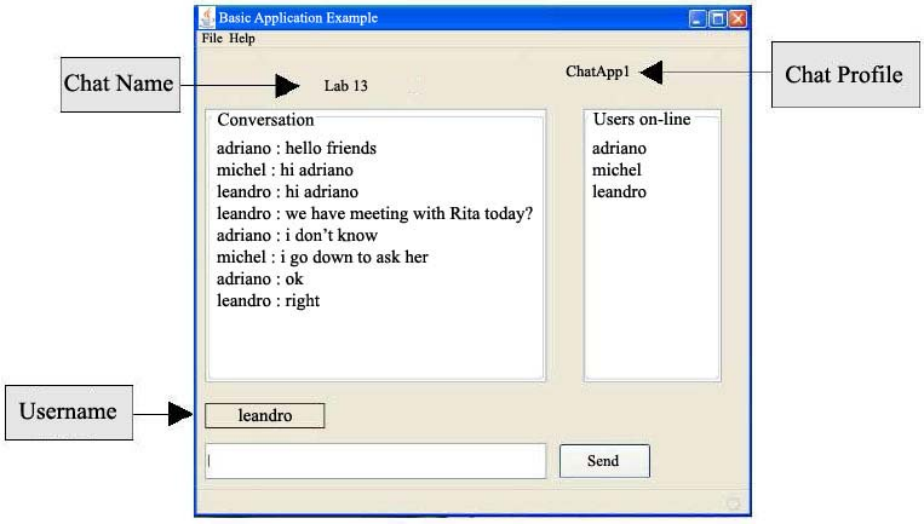

Figure 15: Chat graphic interface

Initially the Chat interface was designed, as illustrated in Figure 15. After that, the event was defined and named newMessage. Then it could be generated and consumed by the application. In this context, the following values could be found (i) the user who sent the message (ii) the room in which the user was present and (iii) the text message. Furthermore, two additional WGWSOA services composed the Chat to support the exhibition of a user list: UserList and UserLogin. For this scenario, three Aw2SOA profiles were created: one for Chat, which defines newMessage as a generated and a consumed event, other one for UserLogin, defining event generation for both login and logoff, and UserList, consuming these events from the UserLogin service (Figure 16).

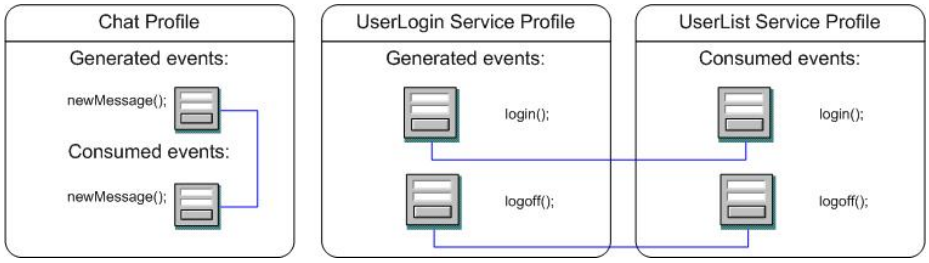

Figure 16: Generated and consumed events in Chat application

After the Chat implementation through Aw2SOA support, the application developers added other functionalities not only through the event consumed models but also through the query service. Other functionalities implemented included presentation of previous conversations and user list with time notification of when the messages were sent and when users were on-line, as well as awareness about users' actions, for example, “lazy” or “writing”.

From the data collected in the experiment as well as the service interface evaluation, it is possible to assert that the service abstraction degree of Aw2SOA can be classified as “detailed”, according to Erl (2005). We can affirm this because rules about both the profiles and logic definition of the returned event type were sufficiently explicit.

Except in the first case study, in both experiments it was necessary to manage profiles and manipulate events from the pre-established model. In the second case study the need to change the profile as a requirement to be implemented in Chat was a challenge, considering also that it was not foreseen during the application specification.

The service-oriented approach is suggested in literature to enhance reuse. Implemented as a WGWSOA service, the Aw2SOA has fulfilled this requirement in the several situations in which it was used. In all the experiments, there was no need to modify any of the services involved in the experiments. In addition, the aspect-oriented approach, used to separate awareness crosscutting concerns, fulfilled its purpose of withdrawing from the service logic the need to implement such functionalities. However, some protocols needed to be followed to facilitate service usage, such as (i) the place where the interfaces should be implemented, and (ii) the

annotation @Awareness usage. The former was necessary considering that the capture points were not defined in the sensor, but in the correct locations of capture. Moreover, both the applications and service developers should manage the defined profiles.

5.3 Awareness support through interoperable services

According to Figure 15, the WGWSOA enables communication between applications hosted in distinct sites. The element responsible for the exchange of messages across different WGWSOA instances is the Wrapper service. In this context event consumption of the delegation activity service will be analyzed (`ActivityManagerService`) through Forum application, which is hosted in a WGWSOA remote instance. Specifically, the goal of this scenario is to add to the Forum the functionality of activity presentations that were delegated to the user at the moment that he/she accesses the groupware.

ActivityManagerService has three event types in its generation profile: (i) createActivity, which indicates the creation of a new activity and contains its main data, such as the description and user creator (ii) delegateActivity, which indicates that the activity was delegated for a specific user and (iii) closeActivity, which indicates the conclusion of the activity.

In order to carry out the defined functionality, the Forum has to be a consumer of delegateActivity event, because, in its context, both the activity name and the user who should execute this activity are already defined. To achieve this, it was necessary to access the Profile Manager of WGWSOA instance containing ActivityManagerService in order to register the interest in this event. After that, the Wrapper service was used as a bridge for the Aw2SOA remote instance execution. By this instance, events were required not only synchronously, through notification, but also asynchronously, through queries. Figure 17 illustrates the described interaction.

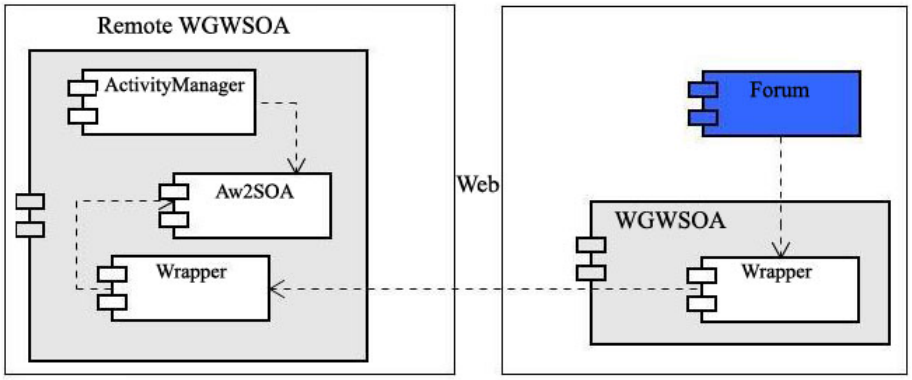

Figure 17: Collaborations to access remote Aw2SOA

Throughout the experiments, it was observed that the events arrived from the Forum in a short time. However, both the data reading and handling of the XML format added a short delay to the application. For example, when approximately five

hundred events were requested, the delay was almost fifteen seconds. This could have occurred because the DOM specification had loaded the XML document to memory. Other approaches, e.g. SAX (Simple API for XML) could be used to solve this issue.

As a result, we can also conclude that the defined event model was capable of being transmitted through communication technologies on the web. This fact has enhanced event transfer across WGWSOA remote instances.

6 Conclusions

Awareness is a functional requirement which is a concern for groupware application. It makes it possible for context activities to be noticed by group participants. Issues concerning this requirement are usually implemented by several architectural units. As a result, the design does not enhance reusability, maintainability or code evolution. In spite of the fact that AOP attempts to solve these challenges, this approach is not widely used when designing awareness elements in groupware. They are frequently designed only by the use of components in the object-oriented paradigm.

A service-oriented approach is suggested in groupware literature to solve reuse and evolution requirements. In a SOA environment, groupware functional requirements can be fulfilled by only one service composed of a group of services. However, representing crosscutting concerns in this architecture is a challenge to enhance reusability as well as loose-coupling.

The objective of this paper was to describe an awareness service, named Aw2SOA, designed to support the development of distributed groupware on the Web. To enhance composability, statelessness, autonomy, reusability and loose-coupling, a middleware service-oriented architecture (WGWSOA) was used. As a result, the Aw2SOA is a composition of other services.

The aspect-oriented paradigm was the adopted approach used to separate crosscutting concerns and event service generation. The use of the object-oriented paradigm to represent such concerns does not enhance loose coupling between services, but promotes functionality redundancy and code dispersion in several parts of the application. Therefore, in order to modularize this crosscutting concern, enhance service maintenance and evolution, an aspect-oriented paradigm was used. We can affirm that in the context of Aw2SOA, aspect-oriented programming can preserve and enhance primary SOA principles in the WGWSOA environment. This could be observed because it was possible to capture generated and consumed events in several other architecture services maintaining them loosely coupled. Thus, supporting awareness in collaborative applications can now be implemented with the use of this service preserving this requirement not only between WGWSOA's services but also between this architecture and the applications which use it.

Throughout this project, other solutions were evaluated, e.g. the Observer design pattern. According to this solution, the applications would be registered as event observers, and the method definition would be captured through aspect inheritance. However, it presented some challenges, for instance, the need for a common updating interface for the observant applications (considering that they can deploy different programming languages), and also the need to have knowledge of aspect-oriented paradigm concepts by the service developers in order to both define and maintain join points.

Initially, the Aw2SOA was implemented and tested in some groupware prototypes. It was then integrated into the WGWSOA current implementation. It has also been used in groupware implementations.

Case studies were carried out in order to evaluate the proposed solution. New services were integrated with the WGWSOA services and groupware applications were developed. By applying the experiments described here, it was possible to see that the strategy of providing awareness support in the service layer of a SOA infrastructure as well as through the aspect-oriented approach can enhance reusability and loose coupling in groupware development.

7 Future Work

Further investigations should be carried out to reveal new challenges regarding the approach used. The WGWSOA is currently being implemented in a distinct and heterogeneous platform regarding the current instance, thus Aw2SOA should also be evaluated in this new context.

Annotations were used in order to support the service development according to the aspect-oriented paradigm. Therefore, it is fundamental that the proposed approach by Aw2SOA be implemented and evaluated in other different platforms than those used by the service presented here.

Furthermore, other case studies should be carried out not only to evaluate the support offered by the awareness service for groupware development, but above all regarding collaboration issues.

Acknowledgements

WGWSOA project has been partially supported by Faculdade Ruy Barbosa (Salvador/BA - Brazil) and FAPESB (Fundação de Amparo à Pesquisa do Estado da Bahia). We are grateful to all WGWSOA research team participants for their contributions. Special thanks to Silvio J. Q. Pereira, Victor O. Reyes, Diego Bastos, Guilherme David G. Toledo, Lucas G. Bittencourt, Humberto G. Sposito, and Bruno M. Costa for their participation in the case studies and for their contributions during the experiments. We would also like to thank Edeyson Gomes for his insights and comments.

References

[Bastos et al. 08] Bastos, A., Oei, M., Menezes, L., Pitangueira, R. S., David, J. M. N.: "Aw2SOA: An Aspect-Oriented Awareness Service for Distributed Groupware. *In: Proceedings of the 12th International Conference on Computer Supported Cooperative Work in Design*, Xi´an, 2008, 404-409.

[Chavez and Lucena 01] Chavez, C. and Lucena, C. J. P.: "Design-level Support to Aspect-oriented Development", *In: Workshop on Advanced Separation of Concerns at OOPSLA'2001*, Tampa Bay. Workshop on Advanced Separation of Concerns at OOPSLA' 2001, 2001.

[Dewan and Sharma 99] Dewan, P. and Sharma, A.: "An Experiment in Interoperating Heterogeneous Collaborative Systems", *In: Proceedings of European Conference on Computer-Supported Cooperative Work (ECSCW'99)*, Copenhagen, Sept., 1999, 11-20.

[Dourish and Bellotti 92] Dourish, P. and Bellotti, V.: "Awareness and Coordination in Shared Workspaces", *In: Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, Toronto, Ontario, 1992, 107-114.

[Ellis et al. 91] Ellis, C. A., Gibbs, J. and Rein, G. L.: "Groupware: some issues and experiences", *Communications of the ACM*, v.34, n.1, 1991, 38-58.

[Erl 05] Erl, T.: "Service-Oriented Architecture - Concepts, Technology and Design", 2005, Prentice Hall PTR.

[Fonseca and Carrapatoso 06] Fonseca, B. and Carrapatoso, E.: "SAGA: A Web Services Architecture for Groupware Applications", *In: Groupware: Design, Implementation and Use. Proc. of 12th International Workshop on Groupware (CRIWG'06)*, Medina del Campo, Spain, 2006, 246-261.

[Fuchs 99] Fuchs, L.: "AREA: A Cross-Application Notification Service for Groupware". *In: Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work (ECSCW'99)*, Copenhagen, Denmark, Kluwer Academic Publishers, Sept., 1999, 61-80.

[Geyer et al. 08] Geyer, W., Filho, R. S. S., Brownholtz, B. and Redmiles, D. F.: "The Trade-Offs of Blending Synchronous and Asynchronous Communication Services to Support Contextual Collaboration". *In: Journal of Universal Computer Science*, vol. 14, no. 1 (2008), 4-26.

[Gutwin et al. 08] Gutwin, C., Greenberg, S., Blum, R., Dyck, J., Tee, K. and McEwan, G.: "Supporting Informal Collaboration in Shared-Workspace Groupware". *In: Journal of Universal Computer Science*, vol. 14, no. 9, 2008, 1411-1434.

[Kiczales et al. 97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J., Irwin, J.: "Aspect-Oriented Programming", *In: Proc. of European Conference on Objective-Oriented Programming (ECOOP)*, Finlândia, Springer-Verlag LNCS 1241, 1997.

[Kiczales et al. 01] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. G.: "Getting Started with AspectJ", *Communications of the ACM*, v.44, n.10, 2001, 59-65.

[Li et al. 03] Li, D., Li, R., Yu, Y., Yang, Y.: "Using Familiar Single-Users Editors for Collaborative Editing". In: *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*, Hawaii, January, 2003, 10 p.

[Maciel and David 07] Maciel, R. S. P. and David, J. M. N.: "WGWSOA: A Service-Oriented Middleware Architecture to Support Groupware Interoperability", *In: Proceedings of the 11th International Conference on CSCW in Design*, Melbourne, Australian, April 26-28, 2007, 556-561.

[Maciel et al. 05] Maciel, R. S. P., Ferraz, C. G. and Rosa, N. S.: "An MDA Domain Specific Architecture to Provide Interoperability Among Collaborative Environments", *In: 19º. Simpósio Brasileiro de Engenharia de Software*, Uberlândia, 2005, 120-135.

[Neyem et al. 08] Neyem, A., Ochoa, S. F. and Pino, J. A.: "Integrating Service-Oriented Mobile Units to Support Collaboration in Ad-hoc Scenarios", In: *Journal of Universal Computer Science*, vol. 14, no. 1, 2008, 88-122.

[OMG 02a] OBJECT MANAGEMENT GROUP - OMG, "CORBA/IIOP Specification Version 3.0", December 2002

[OMG 02b] OBJECT MANAGEMENT GROUP - OMG.: *CORBA Component Model v3.0 full specification.* OMG Adopted Specification (formal/02-06-05), 2002.

[Pinheiro et al. 03] Pinheiro, M. K., Lima, J. V. and Borges, M. R. S.: "A Framework for Awareness support in Groupware Systems", *Computers in Industry*, Irlanda, v. 52, n. 1, 2003, 47-57.

[Prinz 99] Prinz, W.: "NESSIE: An Awareness Environment for Cooperative Settings". *In: Proceedings of the Sitxth European Conference on Computer-Supported Cooperative* Work *(ECSCW'99)*, Copenhagen, Denmark. Kluwer Academic Publishers, Dortrecht, NL, Sept. 12-16, 1999, 391-410.

[Schantz and Schmidt 01] Schantz, R. and Schmidt, D.: "Middleware for Distributed Systems: Evolving the Common structure for Network-centric Applications", *Encyclopedia of Software Engineering*, 2001, Wiley & Sons.

[Shannon 03] Shannon B.: "Java 2 Platform Enterprise Edition Specification v. 1.4", 2003, *Sun Microsystems*.

[Tietze 01] Tietze, D.: "A Framework for Developing Component-based Co-operative Applications", *Ph. D. Dissertation (Computer Science)*, 2001, Technischen Universität Darmstadt, Germany, 178 p.

[Turner et al. 03] Turner, M., Budgen, D. and Bereton, P.: "Turning Software into a Service", *IEEE Computer*, 2003, 38-44.

[Vieira et al. 04] Vieira, V., Mangan, M. A. S., Werner, C. M. L. and Mattoso, M. L. Q.: "Ariane: An Awareness Mechanism for Shared Databases", *In: X International Workshop on Groupware (CRIWG'04)*, San Carlos. Groupware: Design, Implementation and Use. Berlin / Heidelberg : Springer-Verlag, v. 3198, 2004, 92-104.

[Zaupa et al. 08] Zaupa, F., Gimenes, I. M. S., Cowan, D., Alencar, P. and Lucena, C. J. P.: "A Service-oriented Process to Develop Web Applications", *In: Journal of Universal Computer Science*, vol. 14, no. 8, 2008, 1368-1387.