Loosely-Coupled Integration of CSCW Systems

Roberta L. Gomes[*], Guillermo J. Hoyos Rivera[**], and Jean Pierre Courtiat

LAAS-CNRS,
7, Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
{rgomes, ghoyos, courtiat}@laas.fr

Abstract. As collaboration activities usually involve several people with different group tasks and needs, they are rarely supported by a single CSCW technology. Then different types of collaborative applications are usually applied in order to support group work. But in spite of being used to accomplish a common collaboration task, these applications are executed independently without getting any advantage of each other. The integration of such applications would allow them to dynamically interoperate, combining their different functionalities in a controlled way. In order to achieve integration, we propose LEICA[1], a loosely-coupled integration environment which allows collaborative applications to interact without loosing their autonomy. LEICA relies on Web services technology, event notification system, and collaboration policies for controlling the interactions between integrated applications.

1 Introduction

The increasing communication demands of geographically dispersed organizations combined with the technological advances in networking have given rise to the development of a great number of CSCW (Computer Supported Cooperative Work) systems. These systems aim to provide suitable forms of cooperation within a group of users to perform collaborative activities.

As collaboration activities usually involve several people, with different group tasks and needs, they are rarely supported by a single technology. Thus some CSCW systems try to combine different functionalities in order to support collaboration into a single environment. The main weakness of this approach is the challenge behind the anticipation of all the requirements of cooperative situations. This way a CSCW system is hardly suitable and sufficient for every collaborative activity.

As a result, current collaborative environments consist of a range of applications, working side by side but independently, without really getting advantage of each other. Allowing the integration of these applications could bring significant benefits to users. An integrated collaboration environment would allow the different functionalities of existing applications to be dynamically combined and controlled (enhancing

[*] Researcher supported by a scholarship from CNPq, Brazil.
[**] Researcher supported by scholarship 70360 CONACyT/PROMEP, Mexico.
[1] This research is partially supported by the European IST project *Lab@Future*.

flexibility and tailoring possibilities).

In order to achieve the integration of existing CSCW systems avoiding dealing with their low-level features, we propose LEICA, a "Loosely-coupled Environment for Integrating Collaborative Applications". Relying on Web services technology [1] and an event notification system (supporting the publish/subscribe paradigm [2]) different collaborative applications can interoperate by exchanging information within the context of a global *SuperSession*. The loosely-coupled approach proposed by LEICA overcomes two problems usually related to integration environments: (i) it does not require a true semantic integration of collaborative applications, (ii) once integrated to the environment, collaborative applications keep their autonomy.

The definition of collaboration policies controls the interactions among integrated applications, *i.e.* how the collaboration activity supported by one application will be affected by information received from other applications. In practice, these applications interact through the notification of events which may lead to performing specific action(s) in some of these applications.

The interaction degree among integrated applications depends obviously on the nature of the events they are able to exchange, and actions they are able to perform. Three main cases may be considered when integrating applications: a) open source applications, b) API-based applications, and c) applications without any API. Integration of open source applications can achieve the tightest interaction degree, since any internal event/action can be exported/performed; it might however imply great development efforts. Integration of API-based applications is straightforward, and interaction is limited to the provided API. Applications providing no API are constrained to interact only through application *start* and *stop* actions. LEICA's integration approach has been mainly driven by the second case (b). We believe that developers are certainly interested in creating specific and performable collaboration tools that can be used either stand-alone or integrated with other applications (through a flexible API), thus being able to get a great share of the market. This is for instance the case of Skype™ [3], a successful example of communication tool that has recently released its API.

The paper is structured as follows. Section 2 presents related work regarding the integration of existing CSCW systems. Section 3 overviews the general integration approach proposed by LEICA. Section 4 describes how a *SuperSession* is configured and explains how to specify collaboration policies. Section 5 presents the LEICA's architecture and how to integrate applications in practice. Section 6 draws some conclusions and presents directions of future work.

2 Related Work

In [4], Dewan addresses basic issues in interoperating heterogeneous CSCW systems that concurrently manipulate the same artifacts. However, it does not regard the interoperation of CSCW systems that, despite being involved in the same collaborative tasks, do not deal with the same artifacts (*e.g.* videoconference and shared whiteboard).

In [5] authors propose an integrative framework based on a three-level model: ontological, coordination and user interface. An internal knowledge of the collaborative application is needed so that its functionalities can be mapped into the three-level model in order to achieve integration. Accordingly, the integration of third party applications becomes a complex (even impossible) task.

In [6] authors present the CVW, a prototype collaborative computing environment defining a place-based system for integrating document and meeting-centric tools. Basic freeware collaborative applications have already been integrated, and new, special-purpose tools can be integrated, but the integration process is not straightforward – tools must be designed against a place-based API.

Systems like AREA [7] and NESSIE [8] have tried to propose a loosely-coupled integration for supporting cross-application awareness. Like LEICA, these systems are based on the exchange of activity relevant events. However these environments just aim to provide users with a common awareness of the whole collaboration activity. They do not provide any means for defining how an application should react when events are notified by other applications.

Another proposal also based on a loosely-coupled approach is presented in [9]. The authors define a framework where Web services are used to wrap collaborative applications in order to integrate them. Since they leverage open Internet standards, Web services overcome the interoperability issues usually associated with more general integration solutions like CORBA [10], DCOM [11] and EJB [12]. Besides, they are simpler to design, develop, maintain and use. Web services based integration is quite flexible, as it is built on a loose coupling between applications.

One of the drawbacks related to the Web services wrapping approach (in particular to the use of SOAP [13]) is that it represents an additional tier causing some overhead in processing exchanged messages [14,15]. Besides, depending on the architecture of the existing collaborative applications, the complete wrapping of these applications as Web services may imply great development efforts or even applications redesign.

Therefore, unlike the approach employed in [9], and following the recommendations of [14] and [15], we decided to use Web services for coarse-grained operations only. Thus, LEICA applies Web services as an initial mechanism for (i) registering newly integrated applications, (ii) setting and (iii) starting up collaborative sessions. Then, a different infrastructure is used to implement the event notification system in charge of interconnecting the collaborative applications during the execution of an integrated collaborative session. An overview of the proposed integration approach is presented in the following section.

3 General Integration Approach

The integration of a collaborative application to LEICA is achieved by attaching a *Wrapper* to this application[2]. This *Wrapper* comprises a Web services interface allowing the collaborative application to register itself with LEICA[3] as an integrated appli-

[2] *Wrappers* are attached to servers of client/server and multi-server applications, and to the peers of peer-to-peer (P2P) applications.

[3] Except for P2P applications, which use a *P2P Proxy* in order to register themselves.

cation. Through its Web services ports, the integrated application can interact with the *Session Configuration Service* (fig. 1).

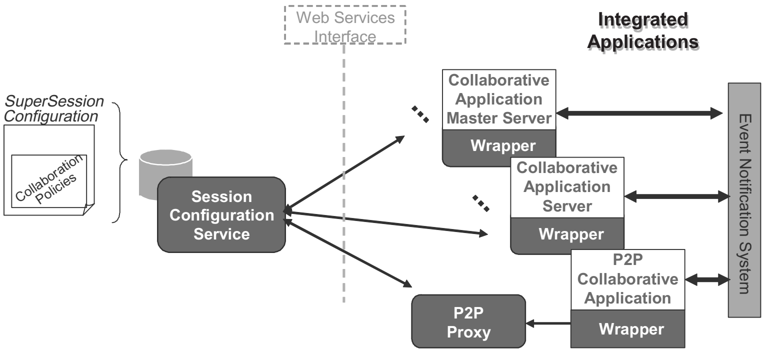
Web Services Interface
Integrated Applications
SuperSession Configuration
Collaboration Policies
Session Configuration Service
Collaborative Application Master Server
Wrapper
Collaborative Application Server
Wrapper
P2P Collaborative Application
Wrapper
P2P Proxy
Event Notification System

Fig. 1. LEICA general integration framework

The *Session Configuration Service* is a Web service used for (i) configuring new global *SuperSessions* and (ii) starting up *SuperSessions*. A *SuperSession* is an integrated collaborative session holding the whole collaboration activity. Within the context of a global *SuperSession*, different *specificSessions* can exist. A *specificSession* is then a conventional collaborative session defined within the context of one collaborative application (*e.g.* a videoconference session, a whiteboard session, *etc.*).

During the *SuperSession* configuration process, the *Session Configuration Service* dynamically contacts each integrated collaborative application in order to request: (i) which specific data is required to create *specificSessions* for this respective application (*e.g.* a videoconference tool might need an IP multicast address); and (ii) which kind of events it can notify, and action requests it can receive. This second information will be used during the collaboration policies definition process.

Collaboration policies are a set of rules following a condition/action model. These rules define how collaborative applications might react when events coming from other collaborative applications are notified. In other words, collaboration policies are the mechanism allowing to determine how an application should react when receiving information (events) notified by other applications.

Once a *SuperSession* has been configured, the *Session Configuration Service* can finally start it. To do so, firstly it contacts each integrated collaborative application requesting them to create the *specificSessions* defined in this *SuperSession*. Then, these collaborative applications are interconnected through an event notification service (as previously explained, from this point Web services are not used anymore).

As collaboration activities progress, collaborative applications exchange event notifications in a peer-to-peer fashion. Meanwhile, *Wrappers* are in charge of managing the collaboration policies. When the *Wrapper* of a collaborative application receives event notifications, it verifies if the notified events enable any policy rule concerning this collaborative application. If so, the *Wrapper* sends action requests to the respective application. Note that LEICA is not intended to support low-level physical events (e.g. mouse click/scrolling) or high frequency synchronization events (e.g. current position of moving objects). It aims to support activity relevant events that carry some semantics.

In the next section we detail the *SuperSession* configuration process, explaining how to specify the collaboration policies of a *SuperSession*.

4 *SuperSession* Configuration

In order to create a *SuperSession*, a two steps configuration process is carried out: (i) Session Management configuration and (ii) Collaboration Policies configuration.

4.1 Session Management Configuration

In the first configuration step, all data necessary to define the main elements of a *SuperSession* are provided. Two groups of information have to be specified:

- General Session Management information (*GSMinfo*). It carries management information such as scheduling, membership and general user roles.
- Integrated Applications information (*IAinfo*). It defines the list of integrated applications to be used during this *SuperSession*. For each collaborative application, a list of *specificSessions* is defined, where specific data required by this application for creating sessions is provided (*e.g.* a videoconference application will be provided with an IP multicast address).

4.2 Collaboration Policies Configuration

The second step of the *SuperSession* configuration process deals with the specification of collaboration policies. As briefly described in section 3, these policies are responsible for linking the collaboration activities supported by different *specificSessions* in the context of the global *SuperSession*. This is carried out through a set of policy rules, basically allowing the association of *n* event notifications to the execution of *m* actions (under certain conditions).

In order to specify the policy rules, a collaboration policies editor might be used. Policy rules are then created through the composition of GUI components, called *policy widgets*. Once the collaboration policies are created, the respective XML data is generated which is appended to the *SuperSession* configuration file. The rules' semantics associated with the XML syntax has been defined using the RT-Lotos formal description technique [16]. This semantics is implemented by a sub-module of the *Wrapper*.

Figure 2 illustrates the *policy widgets* used to create policy rules. These widgets can be connected through their connection points. The basic composition rules are: (i) policy rules are read from left to right; (ii) only widgets without any connection point on their left can appear on the left end of a policy rule; (iii) only widgets without any connection point on their right can appear on the right end of a policy rule.

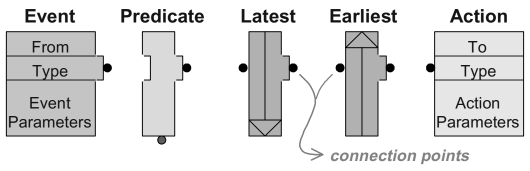

Fig. 2. *Policy widgets*

The *Event* widget represents an event notification. Each *Event* is associated with a collaborative application (field "From") and has a type (field "Type"). In the "*Event* Parameters" area it is possible to define matching patterns (filters) for parameters' values. The *Action* widget represents an action execution request. Each *Action* is associated with a collaborative application (field "To") and has a type (field "Type"). In the "Action Parameters" area, all the required parameters for this action type are specified. Note that all event and action types (and their parameters) are well-known since they are provided by each integrated application.

Figure 3 shows a simple collaboration policy rule associating directly one action with one event. This policy rule is enabled when the specified event is notified. It specifies that: if the application "CA1" notifies an *Event* of type "T1" with parameter "a:M*" (a string starting by "M"), then an action request of type "T2" must be sent to application "CA2". The '%' character is a reference operator, indicating that the *Action*'s parameters "d" and "y" have their values copied from the *Event*'s parameters "b" and "c".

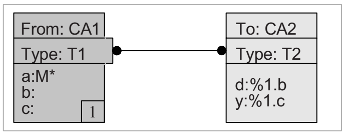

Fig. 3. A simple collaboration policy rule

A *Predicate* widget allows the association of conditions to enable policy rules. A *Predicate* can appear alone or attached to every *policy widget* but an *Action*. It contains a predicate that is specified in Java™ language syntax. *Predicates* can impose time constraints, as well as conditions based on the current *SuperSession* state. When a *Predicate* is attached to an *Event* (or to a *Latest*) it can also reference the parameters of the respective *Event* (or the parameters of the *Events* connected to the *Latest*).

The *Earliest* and *Latest* widgets allow the composition of different *Events* for the specification of a policy rule. When *Event*s are grouped through an *Earliest*, the policy rule is enabled when one of the specified *Events* is notified. When *Events* are grouped through a *Latest*, the policy rule is enabled after all events have been notified.

Figure 4 shows two examples of policy rules using *Latest* and *Earliest* widgets. In the left example, the policy rule is enabled when both specified events are notified (for the first one, the attached *Predicate* must be evaluated to true). Then the *Predicate* associated with the *Latest* widget is also evaluated. If it is true, then the two specified action requests are sent. In the right example, there is an *Event* and a *Predicate* grouped through an *Earliest*. Policy rules like this one aim at waiting the fulfillment of certain conditions to be enabled (*e.g.* regarding the *SuperSession* state); however if a particular event is notified before this condition is satisfied, then the policy rule is also enabled.

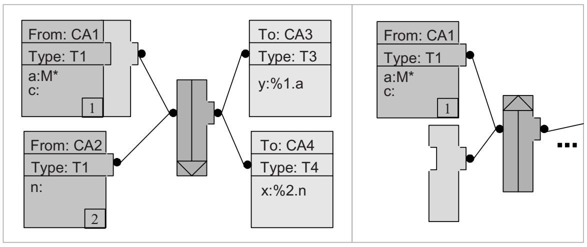

Fig. 4. Policy rules using *Latest* and *Earliest*

An upcoming problem is related to the fact of having more than one *Event* in the same policy rule. Thus, an automatic sequence number is attributed to each *Event* in order to be used as identifier while referencing event parameters. Another constraint related to event parameters referencing appears when *Events* are grouped through an *Earliest*. As it defines a non-deterministic behavior (there is no way of knowing which of the *Events* will enable the policy rule) parameters from *Events* of different types grouped through an *Earliest* can not be referenced by a *Predicate* attached to this *Earliest*.

As illustrated in figure 5, different *Earliest* and *Latest* widgets can be combined in order to create compound rules.

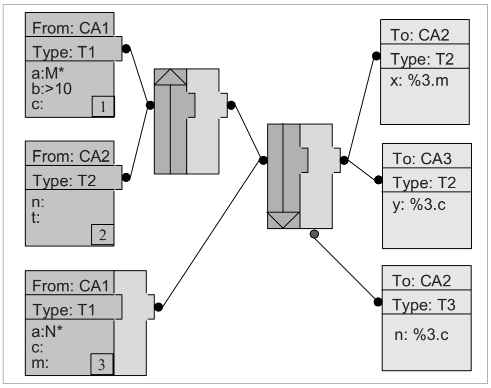

Fig. 5. A compound collaboration policy rule

5 LEICA's Architecture

Following the integration framework presented in section 3, we describe here the LEICA's architecture and different implementation aspects of a prototype currently under development. Java™ has been chosen as underlying technology for implementation. To precisely describe each architecture component, let us consider the five necessary steps to achieve the execution of a *SuperSession*.

5.1 Integrating a Collaborative Application

CSCW systems may present different distribution architectures, varying from centralized to replicated architectures. Centralized architectures are usually implemented according to the client/server or multi-server approaches. Replicated architectures are mainly implemented following the P2P approach. These three different approaches are considered to determine how to integrate collaborative applications to LEICA.

When integrating client/server or multi-server collaborative applications, a *Server Wrapper* must be added to the servers. In the case of a P2P collaborative application, a *P2P Wrapper* is used. As shown in figure 6, the difference between these two *Wrappers* deals with the Web services interface, not present in the second case. Since P2P applications are usually dynamically executed in the users' hosts when they get connected, they cannot be permanently available as Web services. To overcome this problem, a *P2P Proxy* is used.

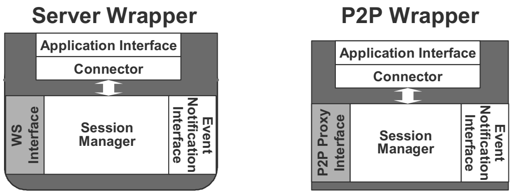

Fig. 6. The LEICA *Wrapper*s

The *Wrapper* is a Java component to be tied to the collaborative application through an *Application Interface*. This last is an abstract class to be extended in order to implement the communication interface with the collaborative application itself [4]. Through this interface the collaborative application notifies the *Wrapper* of "what is happening" inside its collaboration context (*i.e.* make event notifications), and receives all s*pecificSessions* set up and action requests.

The *Session Manager* implements the core functionalities of the *Wrapper*. It is in charge of (i) receiving and handling *specificSession* configuration data; (ii) managing the collaboration policies as it receives event notifications; and (iii) sending event notifications to other collaborative applications.

[4] JNI (Java™ Native Interface) is used for integrating non Java based collaborative applications.

5.2 Registering a Newly Integrated Application

To register with LEICA, as illustrated in figure 7, the *Wrapper* publishes its services in a *Private UDDI Registry* [17]. In multi-server applications, a *Master Server* is designated to register the application. In P2P applications, registering is made through the *P2P Proxy*.

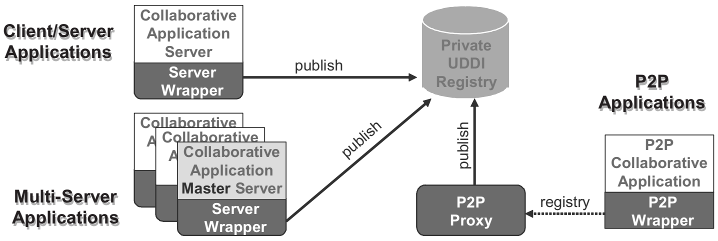
Client/Server Applications
Collaborative Application Server
Server Wrapper
publish
Private UDDI Registry
P2P Applications
Collaborative Application Master Server
Server Wrapper
publish
publish
Multi-Server Applications
P2P Proxy
registry
P2P Collaborative Application
P2P Wrapper

Fig. 7. Registering collaborative applications to LEICA

For implementing the *UDDI Registry*, we use jUDDI [18], a Java implementation that complies with UDDI 2.0. *Wrapper*'s *WS Interface* and the *P2P Proxy* use UDDI4J [19] (Java API for UDDI interaction) to interact with the *UDDI Registry*.

5.3 Creating *SuperSessions*

Figure 8 schematizes the necessary steps for creating a new *SuperSession*.

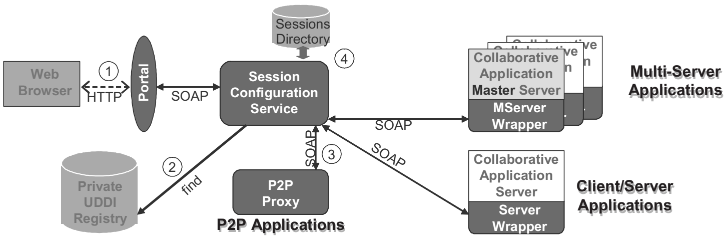
Sessions Directory
4
Web Browser
1
HTTP
Portal
SOAP
Session Configuration Service
Collaborative Application Master Server
MServer Wrapper
Multi-Server Applications
SOAP
SOAP
3
SOAP
2
find
Private UDDI Registry
P2P Proxy
P2P Applications
Collaborative Application Server
Server Wrapper
Client/Server Applications

Fig. 8. Configuration of new *SuperSessions*

1. A Web portal is used to access the *Session Configuration Service* and start the creation process.
2. The *Session Configuration Service* accesses the *Private UDDI Registry* to find out which are the integrated collaborative applications.
3. The *Session Configuration Service* contacts integrated applications to get information about which specific data are needed for configuring *specificSessions*, and which kind of events it can notify and action requests it can treat. Based on this information, *GSMinfo*, *IAinfo* and collaboration policies can be defined.
4. The *SuperSession* configuration file is finally generated and stored.

To implement all these Web services interactions, we use Apache Jakarta Tomcat 5.0 and Apache SOAP 2.3.1.

5.4 Running *SuperSessions*

Figure 9 illustrates how a *SuperSession* is started.

1. A Web portal is used to start a *SuperSession*.
2. The *SuperSession* configuration file is retrieved and parsed. The collaborative applications to be used in this *SuperSession* are identified.
3. The *Session Configuration Service* contacts integrated applications to set up s*pecificSessions* and to send them the collaboration policies of this *SuperSession*.
4. The *Server Wrapper*s of each collaborative application are interconnected through the *Event Notification System*. From this point, Web services are not used anymore.

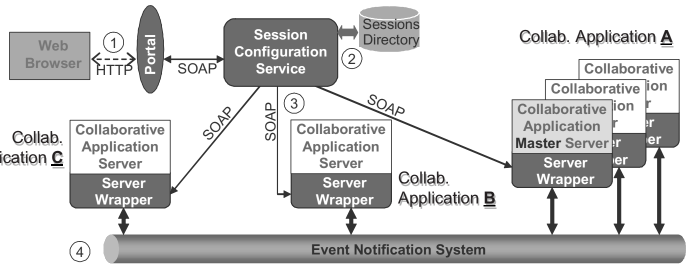

Fig. 9. Starting up a *SuperSession*

In order to implement the event notification system keeping the loosely-coupled nature of LEICA, the publish/subscribe paradigm [2] has been chosen. Publish/subscribe interaction scheme is well-adapted to loosely-coupled environments. In general, subscribers register their interest in patterns of events and then asynchronously receive events matching these patterns, regardless of the events' publishers.

Each *Wrapper* analyses the collaboration policies in order to discover: which type of events it needs to publish, and which type of events it needs to subscribe to. A *Wrapper* just needs to publish *Events* that could enable policy rules, and subscribe to *Events* that could enable a policy rule defining *Actions* to its associated application.

In order to implement the publish/subscribe paradigm for notifying events, we use Scribe [20], a Java based large-scale, peer-to-peer, topic-based publish/subscribe infrastructure. Scribe also provides efficient application level multicast.

5.5 Connecting to a *SuperSession*

In order to connect to a *SuperSession*, a *LClient* application is executed. Figure 10 shows how the connection of a new user is treated.

1. The *LClient* contacts the *Session Configuration Service* and it receives the *GSMinfo*, *IAinfo* and collaboration policies defined to the chosen *SuperSession*.
2. The *LClient* plays the role of a local launch point for P2P and client applications. As it needs to execute the collaboration policies in order to know when P2P/client applications must be launched, it joins the event notification system to receive event notifications.

3. Suppose that, initially, this user is to be connected just to two *specificSessions*, concerning the collaborative applications "B" and "D". Then, the *LClient* runs the client application "B" and the P2P application "D".
4. The *Wrapper* of the P2P application "D" connects to the event notification system and executes the same publishing/subscribing process described in the previous subsection.

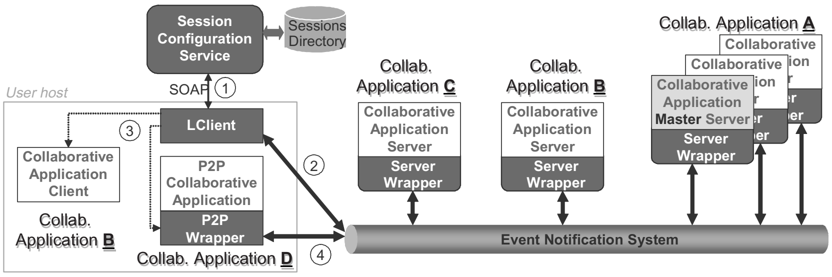
Session Configuration Service
Sessions Directory
SOAP
1
User host
LClient
3
Collaborative Application Client
Collab. Application B
P2P Collaborative Application
P2P Wrapper
Collab. Application D
2
4
Collab. Application C
Collaborative Application Server
Server Wrapper
Collab. Application B
Collaborative Application Server
Server Wrapper
Collab. Application A
Collaborative Application Master Server
Server Wrapper
Event Notification System

Fig. 10. User connection to a *SuperSession*

LClient is a Java application using Apache SOAP 2.3 to contact the *Session Configuration System*, and Scribe's classes to connect to the event notification system.

6 Conclusions and Future Work

This paper has presented LEICA, a loosely-coupled environment for integrating collaborative applications. Existing collaborative applications can be loosely integrated using Web Services as integration technology. In the context of a *SuperSession*, a global collaboration activity is supported where different integrated applications are used in a parallel and coordinated way. Based on the specification of collaboration policies, LEICA defines applications' behavior in response to event notifications.

The current prototype implementation confirms the fact that open source collaborative applications achieve richer interaction levels since we have all the needed flexibility for binding *Wrappers* to applications. However, as new software applications are increasingly coming out of the box with API specifications (sometimes based on Web services), their integration tends to be straightforward.

Regarding Web services, an important effort has been deployed in order to propose solutions for optimizing the transmission and/or wire format of SOAP messages. In this perspective, the current event notification system of the LEICA could also become a Web services-based system without the performance problem actually inherent to SOAP.

Concerning the definition of collaboration policies, no verification of policies' consistency conciseness has yet been performed. Possible solutions based on the use of formal techniques description to guarantee policies' consistency will be studied in a near future.

References

1. Web Services Activity (2005): http://www.w3.org/2002/ws/
2. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.-M.: The many faces of publish/subscribe. ACM Computing Surveys, Vol.35. ACM press (2003) 114-131
3. Skype website. http://www.skype.com/
4. Dewan, P.,: An experiment in interoperating heterogeneous collaborative systems. 6th European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark. Kluwer Academic Publishers (1999)
5. Iqbal, R., James, A., Gatward, R.: A practical solution to the integration of collaborative applications in academic environment. 5th International Workshop on Collaborative Editing Systems, hosted by the ECSCW'03, Helsinki, Finland (2003)
6. Spellman, P. J., Mosier, J. N., Deus, L. M., Carlson, J. A.: Collaborative virtual workspace. International ACM SIGGROUP Conference of Supporting Group Work. ACM Press, Phoenix (1997) 197-203
7. Fuchs, L.: AREA: a cross-application notification service for groupware. 6th European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark. Kluwer Academic Publishers (1999)
8. Prinz, W.: NESSIE: an awareness environment for cooperative settings. 6th European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark. Kluwer Academic Publishers (1999)
9. Fox, G. et al.,: A Web services framework for collaboration and videoconferencing. Workshop on Advanced Collaborative Environments, Seattle, Washington (2003)
10. Orfali, R., Harkey, D.: Client/server programming with Java and CORBA. Wiley, NewYork (1998)
11. Distributed Component Object Model (DCOM) (2005): http://www.microsoft.com
12. Roman, E., Ambler S.W.: Jewell T Mastering Enterprise, JavaBeans. Wiley, NewYork (2001)
13. W3C (2004): Simple Object Access Protocol (SOAP). http://www.w3.org/TR/soap
14. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services (chapter 5). In: Web Services – Concepts, Architectures and Applications, Springer-Verlag (2004)
15. Chiu, K., Govindaraju, M. Bramley, R.: Investigating the limits of SOAP performance for scientific computing. 11th IEEE International Symposium on High Performance Distributed Computing. IEEE press (2002)
16. Courtiat, J.P., Santos, C.A.S, Lohr, C., Benaceur, O.: Experience with RT-LOTOS,a temporal extension of the LOTOS formal description technique. Computer Communications, Vol.23, Num.12, July (2000) 1104-1123
17. W3C (2005): Universal Description, Discovery, and Integration (UDDI) http://www.uddi.org.
18. Apache (2005): jUDDI webpage. http://ws.apache.org/juddi/
19. IBM (2005): UDDI4J webpage. http://www.uddi4j.org/
20. Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.: Scribe: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC), Vol. 20, Num. 8. IEEE press (2002)