# A Service Oriented Approach for Synchronous Collaborative RIAs Development

Tiago C. Gaspar
Federal University of Sao Carlos
Sao Paulo, Brazil
tiago_gaspar@dc.ufscar.br

Cesar A. C. Teixeira
Federal University of Sao Carlos
Sao Paulo, Brazil
cesar@dc.ufscar.br

Antonio F. do Prado
Federal University of Sao Carlos
Sao Paulo, Brazil
prado@dc.ufscar.br

## ABSTRACT

The concepts and technologies that define Web 2.0 have revolutionized and extended computer assisted collaborations. Collaborative applications with synchronous multimedia communication, rich interfaces and using the Web as platform are examples of such revolution. Experiences in that domain allow the identification of commonalities among these applications. A service oriented architecture might be a good choice to lower coupling between platforms and increase software reuse. Based on practical experience, this paper presents a software reuse approach. Web Services and GUI components are proposed to aid development of synchronous multimedia collaborative RIAs in a systematic manner.

## Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.8 D.2.13 [**Reusable Software**]:

## General Terms

Languages, Design

## Keywords

SOA, RIA, Collaboration, Web 2.0, Multimedia

## 1. INTRODUCTION

Web 2.0 [16] is a term that had origin in the concepts and technologies employed by a set of companies during the "Dot-Com bubble"in 2000. That set of companies excelled by employing services and applications that had some characteristics in common such as rich interfaces, Web as platform, harnessing collective intelligence, multi-device software and others. Usually, a Web 2.0 application does not need to comply with all these characteristics simultaneously, but at least a subset of them.

Human relations and knowledge production are changing quickly. That is partially due to new forms of information production and consumption created by the Web 2.0. Collaborations play a key role in such changes and are fundamental to content production. Although the collaboration practice has become popular, synchronous collaborations are not as popular as the asynchronous ones through blogs, social networks, wikis, and others. This is due to many factors, mainly because of technological barriers such as software and broadband network availability. Some of these factors are changing and it would not be a surprise if in a couple years most of our meetings with bank managers, help desk support or guitar teachers would be online through Web applications.

Nowadays, bandwidth for Internet is not a constraint any more for rich media content traffic. Sites like YouTube, Flickr and Justin.tv are examples of growing rich media content availability on the Web. Only in January 2009, 139 millions or 75% of US's Internet users had access to online multimedia content [4]. Despite the large amount of rich media content being produced and consumed today, most of collaborations in the Web are still text based.

Synchronous collaborations might evolve adopting richer media. The ideal scenario is to collaborate remotely in the same way it's done face to face, so that geographic barriers would become less restrictive. Computer aided collaborations offer possibilities not found in face to face collaborations like database information persistence, content search, content production support or remote collaboration. Some of these possibilities can be made available by Web applications that use technologies such as HTML, CSS, Javascript and AJAX. These kind of applications are known as RIAs (Rich Internet Applications) and they support user collaboration as much as desktop applications, although they are on the Web and available by browsers.

Synchronous collaborations usually have some elements in common such as audio, video or text interaction. These basic collaboration elements can be seen as synchronous interaction units. Each of these synchronous interaction units provides a way to exchange information between participants. GUI (Graphical User Interface) components and Web Services can support these synchronous interaction units allowing software reuse among different synchronous collaborative applications as they share basic characteristics.

The *Tidia-Ae* (Information Technology for Development of Advanced Internet - Electronic Learning) [20] project, in

which this works' authors contribute, is an initiative that aims the construction of an open source web electronic learning environment. The *Ae* is a portal that has a set of tools and functionalities to support learning activities. Part of this project is dedicated to research and develop multimedia synchronous collaborative applications with rich interfaces. *Ae* has been used as a learning platform in many universities. As a consequence, there is a demand to expand the existing set of applications.

However, developing rich interface synchronous collaborative applications can be costly and difficult. Compared to desktop applications, RIAs must fulfill non-functional requirements related to security, fast responses, elaborated user interfaces, browser history, and so on. Technological barriers lead development teams to spend long hours addressing these non-functional requirements. Heterogeneity of browsers, platforms and programming languages are problems concerning development teams' productivity. This complex scenario makes it harder to compose teams, predict project costs and maintain existent applications. Therefore, efforts in software reuse might be an important issue.

Mili et al. state that software reuse would improve the overall quality of a system if quality components were used in system's construction. With a reuse process, productivity increases in the same ratio as the process is automated and quality increases in the same extent as quality-enhancing processes are systematized [12].

Reducing software components coupling to platforms can improve software reuse as these components can be available to a larger number os systems. SOA (Service Oriented Architecture) is good alternative to expose these components as platform agnostic services. SOA is a paradigm for realization and maintenance of business processes that span large distributed systems. It is based on three major technological concepts: services, interoperability through an enterprise service bus and loose coupling [11]. From a collaborative perspective, where there are a large number of collaboration environments and many of the collaborative applications reside inside these environments, developing applications that are loosely coupled can be an advantage. In the e-learning area, for instance, there are many collaboration environments such as Sakai, Moodle or BlackBoard. Developing collaborative applications that adopts a SOA allows these applications to be compatible with any of these environments even though they might be in JAVA, PHP or .NET platforms.

This paper proposes a reuse approach based on a SOA, and a set of GUI components, to develop synchronous collaborative RIAs. The purpose of the approach is to aid domain application developers to build applications reusing software artifacts in a systematic manner. The reuse approach is divided in to two parts: Domain Engineering and Application Engineering. Domain Engineering builds software artifacts to be reused such as a set of Web Services, GUI components and an architecture. Application Engineering guides application's assemble connecting existent Web Services and using UI components to provide key functionalities. All applications resulted from this approach share same architecture, although they might use different sets of Web Services or GUI components according to the it's requirements.

## 2. SYNCHRONOUS COLLABORATIVE DOMAIN

Face to face collaborations such as business meetings or classroom lessons can have a correspondence in the computer mediated collaboration domain. Interactive elements present in a meeting, for instance, such as sight, hearing, or a piece of paper can be represented in a computer mediated scenario with video, audio, and whiteboard representation respectively. One could use these computer mediated representations to build applications that support the same functionalities that are available in common collaborations without computer support. The idea is to bring these collaborations to the computer mediated domain so it could be possible to have a remote business meeting, attend a classroom or talk to a bank manager using the Web as platform.

Collaborations supported by computational systems may offer possibilities not found in regular collaborations without computer aid. Functionalities like database information persistence, content search, content production support and remote collaboration are a few examples. In order to make these functionalities accessible for most users, the web environment sounds appropriate due to the large availability of browsers.

The synchronous collaborative applications with rich interface domain is characterized by intense demand for efficient communication services, user management services and complex rich web interfaces. These characteristics can be very expensive to develop for each application. In order to lower costs and development complexity, Web Services and components are proposed throughout this paper. The components provide GUI support to a set of synchronous interaction units such as text, audio, video, whiteboard. The business logic are offered in terms of Web Services and supports tasks such as synchronous communication, authentication, authorization and session management.

From the applications users perspective, the applications constructed from this approach are OS independent and don't require specific software installation. To take advantage of synchronous interaction offered by these applications, the user must have softwares that are already available in most personal computers like a browser and *Flash* plugin. Other devices such as smartphones or PDAs that have browser and *Flash* plugin support are also able to make use of rich interfaces and synchronous interactivity.

From the applications' developers perspective, the proposed architecture and Web Services are also agnostic to specifics platforms. The architecture is designed to make the most of existent Web Services allowing application's business logic to compose it's self using these Web Services.

## 3. REUSE APPROACH

The proposed reuse approach was motivated by the development of synchronous collaborative applications for the Tidia-Ae project. The first application developed was an instant messenger tool with text, audio and video support. As others synchronous collaborative applications were been

designed and developed, it became clear that they all shared some set of functionalities. Some of these commonalities were identified during the analysis phase, such as a communication service for message exchange. Others were identified as late as the final stages of development like some GUI components. After the identification of commonalities, the common parts were refactored, documented, packaged in to software components and Web Services . As other applications were been developed, some of the existent services and components were improved and others created. After two years of this process and seven applications developed, a more stable approach, as described next, was reached.

The proposed reuse approach is based on the building blocks reuse approach with separate activities defined in [12]. The activities are divided in Domain Engineering and Application Engineering as shown in Figure 1.

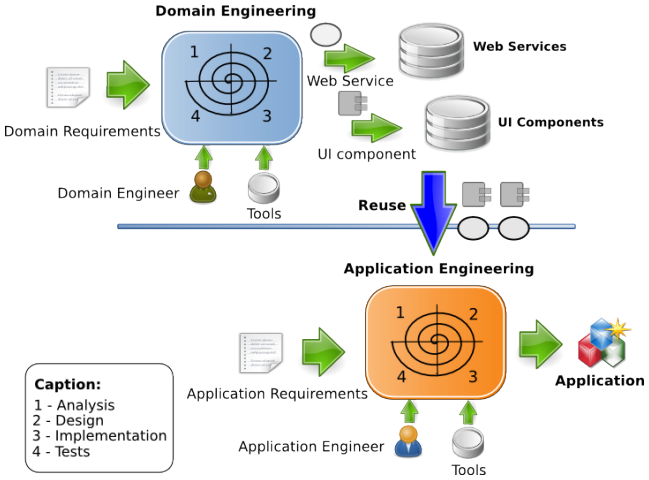


**Figure 1: Reuse process.**

Domain Engineering identifies common aspects of a domain and makes them available for posterior reuse by domain applications. The view is broad, centered not in one particular application but in a set of them. In this activity, it's important to foresee requirements and tendencies for future applications. Web Services resulted from the Domain Engineering are stored as UDDI registries [13] and, analogously, components are stored in a components library.

Differently from the Domain Engineering, that aims a broader universe, the Application Engineering addresses applications development reusing an architecture, Web Services and GUI components created in the Domain Engineering. Applications make use of these software artifacts in order to compose part of its functionalities.

## 3.1 Domain Engineering

Domain Engineering is characterized by development of artifacts in order to provide reuse to domain applications. The development model of Domain Engineering follows the evolutionary prototype paradigm aided by the spiral development model proposed by Boehm [2]. In each increment, four phases takes place: Analysis, Design, Implementation and Tests.

Developing several applications for this domain pointed that most of these applications require some common functionalities. These common functionalities are represented by a set of Web Services and GUI components as can be seen in Figure 2.

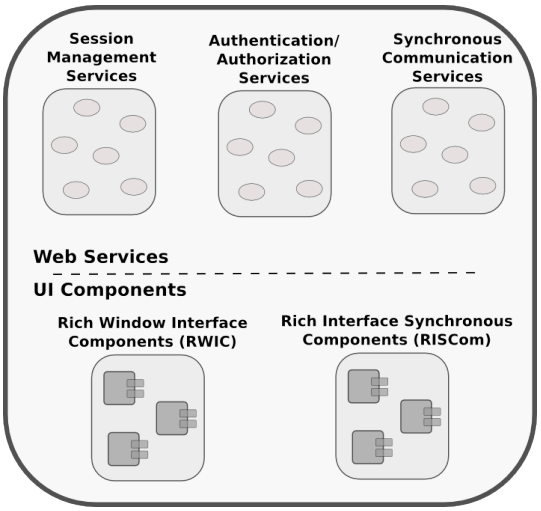


**Figure 2: Synchronous Collaborative Web Services Domain**

The following sections describe in details the available Domain Engineering's Web Services and GUI components.

### *3.1.1 Authentication/Authorization*

Collaborative applications must authenticate and authorize it's users in order to provide secured and controlled access. Often, in corporate or educational environments there are authentication/authorization services already deployed such as LDAP or Microsoft's Active Directory . SOA allows these services to be exposed and used seamlessly by different applications. Makes sense that synchronous collaborative RIAs also access these services to provide authentication and authorization functionalities. If the developer chooses to use proprietary authentication/authorization services, these services can be exposed in the same manner, using the same WSDL interface.

Table 1 lists the most relevant authentication/authorization services in the synchronous collaborative applications domain.

**Table 1: Authentication/Authorization services**

| Authentication/Authorization services | |
|---|---|
| **Service** | **Parameters** |
| authenticate | login, password |
| addUser | login, password, name, email, ..., role |
| isAuthorized | operation_id, role |
| isAuthorized | operation_id, login |
| authorize | operation_id, role |
| authorize | operation_id, login |

Authentication/Authorization Web Services provides a WSDL interface published as UDDI registries and supports SOAP messages. SOAP allows messages to pass through firewalls

using HTTP communication protocol. Corporate or education environments are frequently complex due to secure related issues. Using SOAP to make requests can simplify network setups and increase interoperability.

### 3.1.2 Session Management

Session management is essential to synchronous collaborative applications. That might not be true to asynchronous collaborations. In a social network or blog there's no need to know if a participant is online, therefore schedule a session is not an issue. But in synchronous collaborations it is important to know which participants are members of a session and if they are available or not to receive synchronous messages.

A synchronous collaborative session is about scheduling a meeting between a group of participants. Synchronicity requires that all participants are collaborating simultaneously in a defined time, during a defined period. Another important aspect is defining collaboration roles. Different synchronous collaborative applications might need different roles to fulfill it's needs. In a simple meeting application, two roles might be necessary: a Maintain and an Access role. The Maintain role can create a meeting, include/exclude users and choose applications that will be used in a particular session. A classroom application might need a different setup with 3 roles: Teacher, Teacher Assistant and Student. Each role having different sets of permissions.

Synchronous collaborative applications accesses session management functionalities analogously to Authentication/Authorization Web Services using SOAP, WSDL and UDDI.

Table 2 shows the most relevant session management services in the synchronous collaborative applications domain.

**Table 2: Session Management services**

| Session Management services | |
|---|---|
| **Service** | **Parameters** |
| addSession | site_id, title, description, begin_date, end_date, participant_ids |
| getSession | session_id |
| removeSession | session_id |
| updateSession | session_id, title, description, begin_date, end_date, participant_ids |
| listSessions | site_id, partipant_id |
| addSessionTool | session_id, tool_id, tool_description, tool_topic |

Most of the services in Table 2 are straight forward, except for the last one. *addSessionTool* service requires a *tool_topic* parameter in order to reserve a communication channel to exchange messages. Synchronous collaborative RIAs might be composed of several tools such as a text tool for text communication and an audio/video tool for multimedia communication. All tools are ultimately rendered in the client, so it must know how to exchange messages to its peers. A tool topic allows a tool to send and receive messages to similar tools active on the other client's machines.

### 3.1.3 Synchronous Communication Service (SCS)

Synchronous communication functionalities are perhaps one of the most common and important commonalities of this domain. Every synchronous application needs to exchange messages between two or more participants. In order to provide that functionality, a set of communication Web Services were developed to allow simple and efficient message exchange.

Synchronous communication through Web Services require a different approach compared to traditional Web Service communication. The most common use of Web Services is the request/response communication paradigm. However, a message originated by a synchronous application might need to delivered without a previous request. When a synchronous text message is sent by an application, it needs to notify its destination, other applications, that there's a message to be consumed. The Publish-Subscribe [6], also known as Observer, design pattern is appropriate to provide that behavior.

Web Service Publish-Subscribe is supported by a set of OASIS specifications known as Web Service Notification (WSN) [14]. WSN is tightly integrated to Web Service Resource (WSR) [15], another OASIS standard. WSR allows statefull services supporting WSN to maintain states in order to notify subscribers about any changes in the subscribed subject.

Figure 3 shows a Publisher, a Subscriber, an application server with a notification broker and a messaging engine. In the figure, the Subscriber subscribe itself to a given topic, the Publisher notifies the application server about a modification in the topic and the NotificationConsumer receives a notification message about the change made by the Publisher.

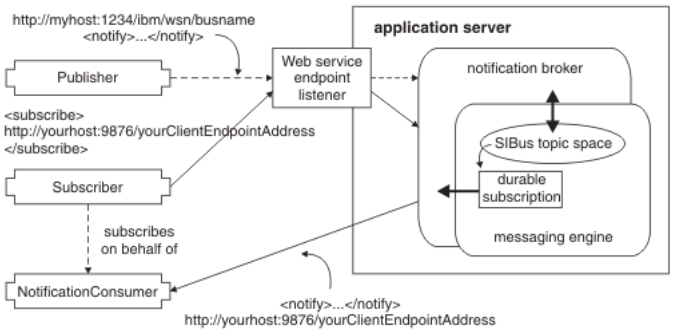


**Figure 3: Web Service Notification [9].**

Another approach is to query regularly a particular Web Service for any changes. That method is called polling and could possibly lead to a scalability issue. If there are a large number of Subscribers, the constant querying might cause heavy processing and a large network traffic. Unless applications present regular content changes, a large portion of the queries will be in vain, since is not possible to predict when an application has it's state changed.

SCS provides a semantic layer to synchronous collaborative applications using the underneath Publish-Subscribe pattern. Table 3 shows the most relevant services provided by

SCS.

**Table 3: Synchronous Communication Services**

| Synchronous Communication Services | |
|---|---|
| **Service** | **Parameters** |
| createTopic | |
| deleteTopic | topic_id |
| connectToTopic | user_id, topic_id, listener |
| disconnectFromTopic | user_id, topic_id |
| sendMessage | user_id, topic_id, message_id |
| getTopicUsers | topic_id |
| getTopicOnlineUsers | topic_id |
| getHistoryMessages | topic_id, start_date, end_date |

Notice that there's no receiveMessage service. The Subscriber, or application, must implement the NotificationConsumer interface to receive messages. So, it is necessary that the application publishes a Web Service to be receive notifications. This is not a problem to Web applications, most of the Web containers such as Tomcat, WebSphere or JBoss provide functionalities to expose Web Services.

### 3.1.4 Media Streaming Service (MSS)

A media streaming service is necessary in order to support high quality audio and video streaming. Although SOAP is a very flexible way to provide communication, it's not appropriate to live streaming audio and video. In fact, the HTTP protocol adds an overhead to multimedia streaming. Instead, the *RTMP* (Real Time Messaging Protocol) protocol[1][1] is used by the Media Streaming Service to support audio and video live streaming. RTMP servers can also persist media streamings.

### 3.1.5 GUI components

Develop RIAs has proved to be difficult and time consuming. The first Tidia-Ae applications pointed that roughly 60% of all development effort was devoted to that topic. To tackle this problem, GUI AJAX enabled window components were developed.

To show how straight forward those components can be used by applications, Listing 1 shows a JSF file that displays a Web page with a synchronous text interaction window. That simple application allow users, represented by the *logicBean*'s *participants* property, to exchange text messages.

```
<%@ taglib uri="http://br.fapesp.tidia.ae.ccsw2_0/jsf/
      rwic" prefix="rwic" %>
<f:view><h:form>
    <rwic:textWindow participants=#{logicBean.
          participants} style="...">
</h:form></f:view>
```

**Listing 1: Simple synchronous text interaction application.**

The developed GUI components were grouped in to two taglibs: RWIC (Rich Window Interface Components) and RISCOM (Rich Interactive Synchronous Components) [7]. RWIC components were designed to provide basic functionalities, such as audio, video and text support. RISCOM offers a set of more meaningful components like an Instant Messenger, Chat, List of Participants and Whiteboard.

[1]The *Flash* media streaming server used in this work is the open source project *Red5*. MSS is also compatible to *Flash Media Server*.

**RWIC - Rich Window Interface Components**

RWIC provides support to synchronous updates through AJAX[2], window actions (minimize, maximize, resize, popup, drag, and others), layout management and integration with messaging services SCS (Synchronous Communication Service) and MSS (Media Streaming Service).

Figure 4 shows an UML class diagram of the RWIC's tags internal objects. Abstract classes like *GenericCommWindow* and *AudioVideoWindow* encapsulates rich interfaces and communication complexities. *TextWindow* is a JSF tag that can be used for text exchange. *PublishVideoWindow* is a JSF tag that can be used for audio and video capture and streaming. *PlayVideoWindow* is also a JSF tag that displays a multimedia stream.

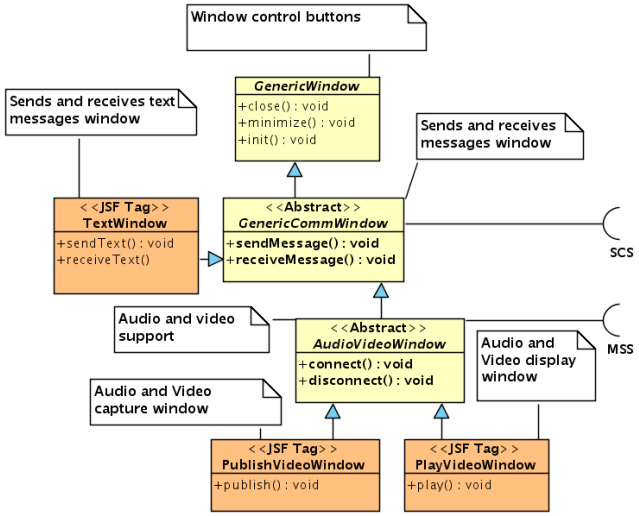


**Figure 4: RWIC class model.**

**RISCom - Rich Interactive Synchronous Components**

GUI components reuse is also available through a more functional set of components. The tag library *RISCom* provides components that can be combined in several ways. Some of these components are:

- *Instant Messenger*: GUI component for synchronous instant n-to-n participants interaction that supports audio, video, text, emoticons and file exchange.
- *List of Participants*: GUI component that exhibits present participants in a collaboration. It allows contact list selection actions to be customized in order to

[2]Although AJAX is by definition asynchronous, the user has the impression of synchronous updates due to the server's short answer periods (Reverse AJAX [3]).

activate other applications such as *Instant Messenger* or *Whiteboard*.

- *Mosaic*: GUI component that exhibits multiple video panels of participants. This component is actually a layout manager that has many instances of *PlayVideoWindow*.
- *Chat*: GUI component to text communication that offers content moderation functionalities.
- *Whiteboard*: GUI component of a synchronous shared whiteboard. Free hand annotations, geometric figure drawings and slides changes made on the whiteboard are synchronously updated and can be persisted for later reference [10].

## 3.2 Application Engineering

Application Engineering provides developers with an approach to build synchronous collaborative RIAs aided by the Web Services and GUI components described in previous sections.

Developing synchronous collaborative RIAs differs from developing desktop applications due to the previously mentioned non-functional requirements such as security, rapid responses, browser related issues, elaborated GUIs and so on. In order to address these non-functional requirements, the GUI component library was developed. RWIC and RISCom provides functionalities that can be useful to most of the synchronous collaborative RIAs. Text, audio and video interactions are most basic forms of information exchange in a synchronous computer mediated collaboration.

Another aspect of this domain is rapid prototyping. Clients often require small deliverables, maybe due to risks related to porting tasks that are usually done by desktop applications, or not even computer mediated, to Web applications.

As shown in Figure 1, applications are developed by the same Domain Engineering disciplines: Analysis, Design, Implementation and Tests. The specific activities that takes place in each of theses disciplines, regarding Application Engineering, are:

- Analysis: the application's requirements are identified, as well as a set of existent Web Services and GUI components candidates that might fulfill part of the requirements.
- Design: the set of candidate Web Services and GUI components are refined and attached to a defined architecture. The remaining requirements are designed guided by the application's specifics requirements (not fulfilled by domain's artifacts), architecture, selected Web Services and GUI components.
- Implementation: specific application's functionalities are implemented, GUI Components and Web Services are connected to the application's code.
- Tests: after unit testing application's specific functionalities, integration, system, and acceptance tests takes place with the selected Web Services and GUI Components already in place.

After all those disciplines, a small deliverable should be available for appraisal. If the deliverable is not approved by the client, than a new cycle must begin with the feedbacks from the previous increment. After the client's approval, a new set of requirements will be addressed in the next increment. The process goes on until the application is considered complete and accepted by the client.

Figure 5 shows an application architecture. It's instantiated to an example, that will be detailed later, but it's important to emphasize that all applications in the proposed approach follow the same architecture. The internal application structure is divided in to three tiers: *View*, *Logic* and *Persistence*. Each tier defines set of responsibilities and depends only of the direct lower tier. Messages are exchanged without jumps between tiers.

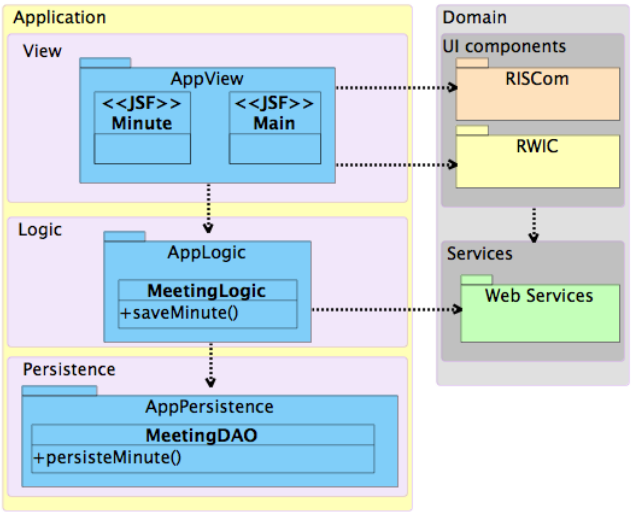


**Figure 5: Application standard architecture.**

*Meeting*, a proof of concept application, whose architecture is shown in Figure 5, was designed to support online meetings so that participants can see, hear, text each other, scribble on the whiteboard and at the end of the meeting, compose a minute with the discussed points. This example was selected emphasize how the proposed architecture supports reuse of domain artifacts and application's specific requirements. Notice that most of the requirements of *Meeting* are covered by the existent GUI components and Web Services except for the minute requirement. The proposed reuse approach allows developers to focus mostly on the application specific requirements.

*Main* JSF page gathers all GUI components reuse while *Minute* JSF page is responsible for the minute GUI. *MeetingLogic* and *MeetingDAO* are devoted to the minute's functionalities only, accessing Session Management Services to access users informations. Notice that application's developers don't need to acknowledge reused internal GUI components complexities related to audio, video, text or whiteboard. Internally, these GUI components uses the previously described Web Services to provide its functionalities.

Figure 6 shows how the Meeting application calls a Session Management Web Service to access participants information

for a given session.

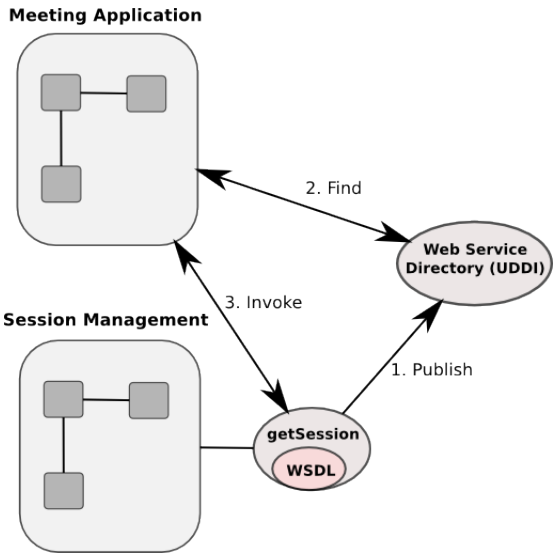


**Figure 6: Authentication/Authorization Web Service request.**

Listing 2 shows the use of the GUI components by the Meeting application in the *Main* JSF page[3].

```
<%@ taglib uri="http://br.fapesp.tidia.ae.ccsw2_0/jsf/
    rwic" prefix="rwic" %>
<%@ taglib uri="http://br.fapesp.tidia.ae.ccsw2_0/jsf/
    riscom" prefix="riscom" %>
<f:view><h:form>
    <h:commandLink value="sair" action="finish" style="
        ...">
    <riscom:whiteboard participants=#{MeetingLogic.
        participants} editAll="true" style="...">
    <rwic:textWindow participants=#{MeetingLogic.
        participants} moderate="false" style="...">
    <rwic:publishAV serverUrl=#{MeetingLogic.
        MSSServerUrl} stream=#{MeetingLogic.userId}
        moderate="false" debug="true" style="...">
    <riscom:mosaicAV serverUrl=#{MeetingLogic.
        MSSServerUrl} streams=#{MeetingLogic.
        participantsId} moderate="false" debug="true"
        style="...">
</h:form></f:view>
```

**Listing 2: *Main* JSF page code.**

Figure 7 shows a screenshot of the *Main* JSF page.

Most of the proposed services described in this work provides semantic layer to other services and frameworks that are commonly used in software development, particularly in JAVA. Some of these services used in developed applications, including Meeting, are: JSF and ZK framework for GUI, Apache Axis for SOAP communication, Hibernate as ORM framework, MySQL as DBMS, Red5 for audio and video streaming service, Tomcat as application container and Spring as IoC manager.

[3]The code listed has been simplified to emphasize the use of RWIC and RISCom tags.

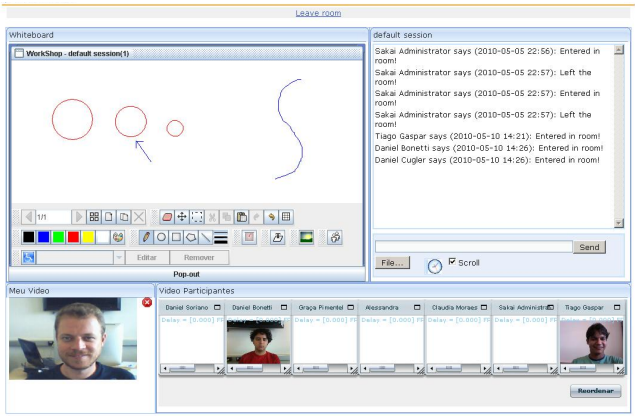


**Figure 7: Meeting application's screenshot.**

# 4. RELATED WORK

Web 2.0 synchronous collaborative works are still hard to find. However, many synchronous collaborative development approaches were proposed in the past, particularly in the groupware toolkits category. Some of them are Groupkit [18], COAST [19], Clockworks [8] and Redezvous [17]. Although these toolkits are elaborated (some of them proposing languages for domain specification) they lack more practical approach in technological terms. They tend to propose custom services and components that does not take advantage of the scalability and robustness that frameworks and services already available can offer. Another issue is that they are mostly desktop applications.

PowerMeeting [21] is a recent work and does proposes a Web environment for rich user interface synchronous collaboration using GWT and AJAX. It supports whiteboard, brain storming and chat tools. PowerMeeting also provides a plugin oriented framework in order to allow developers to build plugin applications. However, despite the reuse concern like it's framework and plugin facilities, it doesn't offer a defined reuse process. It's structure doesn't support much interoperability leading all developed applications to reside inside it's web portal.

# 5. CONCLUSIONS AND LEARNED LESSONS

This work presents an approach, motivated by real needs, to reuse software in the development of synchronous collaborative RIAs. The experience in the Tidia-Ae project was fundamental to the proposal conception, providing a realistic view of the domain and associated risks in the development of applications.

Some applications were developed using the described Web Services, GUI components library and architecture such as a remote teaching application (REFACE), a video conference application with teleprompters (Tete-a-Tete) and a thesis defense application (Viva). In order to give the reader an idea of the approach's results in the application development process, an experienced Java developer, with the development environment set, took approximately 5 hours to develop the case study Meeting, described in section 3.2. The number of code lines written exclusively for the Meeting application compared to the total code lines used by the

Meeting application (including components) showed 83% of code reuse.

Adopting a SOA provided the applications constructed from the proposed reuse approach with loose coupling, separation of concerns and a well defined architecture. Such characteristics adds value to applications and increases interoperability between platforms.

# 6. FUTURE WORK

Due to increasing popularity of smartphones and new generation tablets, in a near future most of the collaborations could happen through such devices. This work might evolve the proposed approach to improve support for mobile devices. Specific mobile GUI components could be developed to allow data communication between the described components. The proposed WebServices could evolve and include REST support to some of its services which could simplify development.

# 7. ACKNOWLEDGEMENTS

FAPESP provided support to the development of this work through the project TIDIA-Ae, process 2005/60653-1.

# 8. REFERENCES

[1] Adobe. Real-time messaging protocol. http://www.adobe.com/devnet/rtmp/, May 2009.

[2] B. Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):14–24, 1986.

[3] E. Bozdag, A. Mesbah, and A. van Deursen. A comparison of push and pull techniques for ajax. *ArXiv e-prints*, June 2007.

[4] ComScore. Youtube.com accounted for 1 out of every 3 u.s. online videos viewed in january. http://www.comscore.com/Press_Events/-Press_Releases/2008/03/YouTube_Usage, Jul 2009.

[5] Fapesp. Fundação de amparo à pesquisa do estado de são paulo. http://www.fapesp.br/, January 2010.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addision-Wesley, 1995.

[7] T. C. Gaspar, A. F. Prado, and C. A. Teixeira. Linha de produtos de software para colaboração síncrona na web 2.0. In *Anais do Simpósio Brasileiro de Sistemas Multimídia e Web (Webmedia)*, 2009.

[8] T. Graham, C. Morton, and T. Urnes. Clockworks: Visual programming of component-based software architectures. *Journal of Visual Languages and Computing*, 7(2):175–196, 1996.

[9] IBM. Web service notification overview. http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1-/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/-ref/rjwsn_ex_sub.html, 2008.

[10] C. Jardim, A. Martelini Jr, J. Freire, E. Silva, S. Lara, F. Santos, T. Kudo, R. Fortes, and M. Pimentel. Whiteboard: uma ferramenta de apoio ao ensino e aprendizado com uso de anotação eletrônica. *XVI Simpósio Brasileiro de Informática na Educação (SBIE)*, 2005.

[11] N. Josuttis. *Soa in practice*. O'Reilly, 2007.

[12] H. Mili, F. Mili, and A. Mili. Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering*, pages 528–562, 1995.

[13] OASIS. Specification tc.: Uddi version 3.0.2 specification. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm, 2004.

[14] OASIS. Web service notification standard. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn, 2006.

[15] OASIS. Web service resource standard. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=-wsrf#technical, 2006.

[16] T. OReilly. What is web 2.0: Design patterns and business models for the next generation of software. http://oreilly.com/web2/archive/what-is-web-20.html, September 2005.

[17] J. Patterson, R. Hill, S. Rohall, and S. Meeks. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, page 328. ACM, 1990.

[18] M. Roseman and S. Greenberg. Building real-time groupware with groupkit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(1):66–106, 1996.

[19] C. Schuckmann, L. Kirchner, J. Sch "ummer, and J. Haake. Designing object-oriented synchronous groupware with coast. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 30–38. ACM New York, NY, USA, 1996.

[20] Tidia-Ae. Tecnologia da informação para o desenvolvimento da internet avançada - aprendizado eletrônico. http://tidia-ae.incubadora.fapesp.br/portal, May 2009.

[21] W. Wang. Powermeeting: gwt-based synchronous groupware. In *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, pages 251–252. ACM New York, NY, USA, 2008.