Component Reuse in Synchronous Collaborative Web 2.0 Applications

Tiago C. Gaspar\textsuperscript{1}, Antonio F. do Prado\textsuperscript{1}, and Cesar A. C. Teixeira\textsuperscript{1}

Universidade Federal de Sao Carlos, Sao Carlos - SP, Brazil

Abstract. The concepts and technologies that define Web 2.0 has revolutionized and extended collaboration assisted by computational systems. Recent developments in new technologies makes it possible to represent, in a Web environment, elements that are common to face-to face collaborations. The Web environment is formed by synchronous collaborative multimedia applications with rich interfaces using the Web as platform. The authors’ experience in projects on the synchronous collaborative application’s domain has allowed the identification of a complex and broad domain with many commonalities. This paper presents a software component reuse approach for synchronous collaborative Web 2.0 application’s development. It describes the construction of such components as well as an approach to reuse software by taking advantage of a shared architecture and components library to aid domain’s developers.

1 Introduction

Web 2.0 \cite{13} 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 stood up from others 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 doesn’t comply with all those characteristics simultaneously, but a subset of them.

Humans relations and knowledge production are changing rapidly. 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 collaboration practice has become popular, synchronous collaborations are not as popular as the asynchronous ones through blogs, social networks, wikis, and others. That is due to many factors, mainly because of technological barriers such as software and broadband network availability. Some of those factors are changing and it would not be a surprise if in a few years most of our meetings with bank managers, help desk support or guitar teachers would be online through Web applications.

The available bandwidth for Internet access supports rich media content production nowadays. Sites like YouTube, Flickr and Justin.tv are examples of growing rich media content available on the Web. Only in January 2009, 139 millions or 75\% of US’s Internet users had access to online multimedia content
Despite the large amount of rich media content being produced and consumed today, most of collaborations are still text based.

Synchronous collaborations supported by computational systems might evolve adopting richer media. The ideal scenario is to collaborate remotely the same way it’s done in face-to-face collaborations, so that geographic barriers would become less restrictive. Computer aided collaborations offer possibilities not found in regular collaborations like database information persistence, content search, content production support or remote collaboration.

Synchronous collaborations usually have some elements in common such as audio, video or text interaction. Those basic collaboration elements can be seen as synchronous interaction units that provide a way to exchange information. The software components that represent those synchronous interaction units allow software reuse among different synchronous collaborative applications as they share basic characteristics.

The Tidia-Ae (Information Technology for Development of Advanced Internet - Electronic Learning) project, which has the contribution of this work’s authors, 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 that project is dedicated to the research and development of multimedia synchronous collaborative applications with rich interfaces. Ae has been used as a learning platform in many universities and therefore that is a demand to expand the existing set of applications.

However, developing synchronous collaborative applications with rich interfaces can be costly and difficult. Compared to desktop applications, those applications must fulfill non-functional requirements related to security, fast responses, elaborated user interfaces, browser related issues, and so on. Technological barriers lead development teams to spend long hours addressing those non-functional requirements. Heterogeneity of browsers, platforms and programming languages are another problem concerning development teams’ productivity. That 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. [12] state that software reuse improves the overall quality of a system if quality components were used in that 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.

This paper proposes a component based development approach for the Web 2.0 synchronous collaborative applications domain. The purpose of this approach is to aid domain application developers to reuse software in a systematic manner. The reuse approach is divided into Domain Engineering, in which the domain components are developed and stored in a library, and Application Engineering, in which applications are assembled using the components library. All applications resulted from this approach share same architecture, although applications
might use different sets of synchronous interactive components according to the
it’s requirements.

2 Synchronous Collaborative Domain

Face-to-face collaborations such as business meetings or classroom lessons can
have counterparts in the computer mediated synchronous collaborative domain.
Interactive elements present in a business meeting, for instance, such as sight,
hearing, or a piece of paper can be represented in a computer mediated sce-
nario with video, audio, and whiteboard representation respectively. One could
use those 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 those 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.

Human senses play an important role in face-to-face collaborations. In a
classroom, not only the image or the voice of the teacher might have influence in
the learning process, but also the smells and the classroom environment itself.
It is difficult with today’s technology to offer an accurate real life experience in
terms of human senses stimuli. The digital representation of smell, for instance,
is not quite ready yet. Although several studies [14] [21] [4] and products are
available [19], those devices are far from effective.

Despite the lack of better support for human senses stimuli, collaboration
supported by computational systems may offer possibilities not found in face-to-
face collaborations. Functionalities like database information persistence, con-
tent search, content production support and remote collaboration are a few ex-
amples. In order to make those functionalities accessible for most users, the Web
environment sounds appropriate since it is becoming an ubiquitous environment.

Web synchronous collaborative applications are characterized by intense de-
mand for efficient communication services, user management services and com-
plex rich web interfaces. Those characteristics can be very expensive to imple-
ment for each application. In order to lower costs and development complexity,
services and components an an approach to use them are proposed throughout
this paper. Some of the components support a set of synchronous interaction
units such as text, audio, video, whiteboard and desktop sharing. Others are
meant to support more background tasks such as communication, user manage-
ment, session management and persistence.

Applications developed with the proposed approach are OS independent and
don’t require specific software installation. To take advantage of synchronous
interaction offered by those 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 and PDAs that have browser and Flash
plugin are also able to support rich interfaces and synchronous interactivity.
3 Reuse Approach

The proposed reuse approach was motivated by the development of synchronous collaborative Tidia-Ae applications. The first application developed was an instant messenger tool with text, audio and video support. As others Tidia-Ae synchronous collaborative applications were designed and developed, it became clear that they all shared some set of functionalities. Some of those commonalities were identified during the analysis phase, such as a communication service for message exchange, others, like some UI components, were identified as late as the final stages of development. After the identification of commonalities, the common parts were refactored, documented, packaged into software components and stored in a library. As other applications were been developed, the existent components were improved and others created. After three years process, it finally reached a more stable state as described next.

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.

Domain Engineering identifies common aspects of a domain and makes them available for posterior reuse by domain applications. The view is broader, centered not in one particular application but in a set of them. In that activity, it’s important to foresee requirements and tendencies for future applications. All components resulted from the Domain Engineering are stored in a components library for opportunistic reuse, whether by applications or in the development of new components.

Differently from Domain Engineering, that aims a broader universe, Application Engineering addresses application development reusing an architecture and components from the Domain Engineering. The applications use the components library to compose part of its functionality.

![Fig. 1. Reuse process.](image-url)
### 3.1 Domain Engineering

Domain Engineering is characterized by the 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 take place: Analysis, Design, Implementation and Tests.

The resulting components from the Domain Engineering are divided in 3 tiers: View, Logic and Persistence. In the View area, the rich interface components can be reused by JSF (JAVA Server Faces) tags. In the Logic area, the components are services that aids communication, authentication and collaborative session management. In the Persistence area, the components abstract complexities in creation, retrieval, update and deletion operations of persistent information in databases. Figure 2 shows an overview of the components library.

Dependencies among components or applications are managed by the Inversion of Control (IoC) design pattern [7]. An XML configuration file keeps all dependencies. The components’ implementations are gathered in that file. If any of those components’ concrete implementation needs to be replaced, the changes are restricted to the configuration file only, as long as the declared interfaces remain unchanged.

Listing 1 shows an example of persistence and communication services injection in the logic bean of the Instant Messenger application.

```xml
<!-- IoC configuration file-->
<bean id="br.fapesp.tidia.ae.im.logic.IMLogic" class="br.fapesp.tidia.ae.im.logic.
impl.IMLogicImpl" init-method="init" singleton="false">
  <property name="dao" ref="br.fapesp.tidia.ae.im.dao.IMDaoMySQL"/>
  <property name="commController" ref="br.fapesp.tidia.ae.scs.logic.LogicImpl"/>
</bean>

Listing 1.1. An application IoC configuration file
```

To illustrate how IoC works, suppose, in Listing 1, that the persistence tier (represented by the dao property) is MySQL database dependent. If MySQL needs to be changed to PostgreSQL, for instance, then, considering that the PostgreSQL DAO class implements the DAO interface, the only modification required is to update the dao property value. IoC brings adaptability to components and applications, lowering coupling and providing better maintainability.

In the next sections, the most relevant components developed by the Domain Engineering are described, starting with UI components, followed by service components.

**UI components.** One the most time consuming tasks of the synchronous collaborative web applications domain is to develop Web rich interfaces. Roughly 60% all development effort in the first developed applications was devoted to that topic. In order to lower that development effort, UI components were developed.
Most of those components are encapsulated by floating windows with resizing, drag, pop-up, maximize, minimize, close functionalities and AJAX communication support.

To show how those UI components can be used by applications, Listing 2 shows a JSF file that displays a Web page with a synchronous text interaction window. That simple application allows users, represented by the `logicBean`’s property `participants`, to exchange synchronous text messages.

```jsp
<%@ taglib uri="http://br.fapesp.tidia.ae.ccsw2000/jsf/rwic" prefix="rwic" %>
<f:view><h:form>
    <rwic:textWindow participants=#{logicBean.participants} style="..."/>
</h:form></f:view>
```

Listing 1.2. Simple synchronous text interaction application.

The developed UI components were grouped in two JSF tag libraries: RWIC (Rich Window Interface Components) and RISCom (Rich Interactive Synchronous Components). RWIC components were designed to provide some basics functio-
nalities, like audio, video and text interactions. RISCom offers a set of more meaningful components like an instant messenger, a chat, a list of participants, a whiteboard and desktop sharing.

**RWIC.** RWIC provides support to synchronous updates through AJAX\(^1\), 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), both described in later sections.

Figure 3 shows a class diagram of the RWIC’s internal objects. Abstract classes like GenericCommWindow and AudioVideoWindow encapsulate 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.

**RISCom.** UI components reuse is also available through a more meaningful set of components. The JSF tag library RISCom provides components that can be combined in several ways. Some of those components are:

- **Instant Messenger:** UI component for synchronous n-to-n participants interaction that supports audio, video, text, emoticons and file exchange.
- **List of Participants:** UI component that exhibits online and offline participants in a collaboration. It allows participant selection actions to be customized in order to activate other applications such as Instant Messenger or Whiteboard.

\(^1\) Although AJAX updates are asynchronous, the user has the impression of synchronous updates due to the server’s short answer periods (Reverse AJAX [3]).
Communication Services. Synchronous communication services is perhaps one of the most fundamental commonalities of this domain. Every synchronous application needs to exchange messages between two or more participants. In order to provide that functionality, a communication service was developed to allow simple and efficient message exchange.

Synchronous Communication Service (SCS) allows message exchange following the Publish-Subscribe design pattern [8]. SCS supports text, files, notifications or custom application messages. The service also supports off line messages that are delivered as soon as the participants become online. SCS provides a domain semantic to synchronous collaboration message exchange and depends of an external message service such as JMS (Java Message Server)[11].

Figure 4 shows the internal class diagram of the service. In that figure, the SCSClientReceiver class acts as mediator to an external messaging service, receiving messages and forwarding it. Clients must specialize that class to receive messages. SCSLogic offers, mainly, methods to establish connections and to send messages. The SCSProviderAdapter class adheres to the Adapter design pattern [8] in order to lower coupling between the external messaging service2. SCSClientReceiver as well as SCSLogic are injected in the applications by the IoC container and, therefore, the external messaging service replacement won’t require any code changes.

The Media Streaming Service (MSS) offers audio and video live streaming using the RTMP3 (Real Time Messaging Protocol) protocol [1] and it also can persist media streamings. A communication bridge between MSS and SCS was developed to allow notification events such as video and recording events (start and finish). Some application instance might need to be notified when an audio/video publication has been started by another application instance, so it is possible, for instance, to open PlayVideoWindow and display the participant’s streaming media.

Other Services Other services present in the components library are:
Fig. 4. Synchronous Communication Service (SCS) class model.

- The Session Service offers session management such as email notification, timeout notification, parameters configurations, schedule conflicts, participants informations and others.
- The Authentication Service offers session authentication, user management, role management and participant detailed information.
- The Persistence Service (GenericDAO) supports data persistence management. It provides functionalities to data creation, retrieval, update and deletion.

### 3.2 Application Engineering

The proposed Application Engineering provides developers with an approach to build synchronous collaborative Web applications aided by the services and UI components described in previous sections.

Developing synchronous collaborative Web applications differs from conventional development such as desktop applications due to the previously mentioned non-functional requirements such as security, rapid responses, browser related issues, elaborated UIs and so on. In order to address those non-functional requirements, the UI components and services were developed. RWIC and RISCom provides functionalities that can be useful to most of those applications. Text, audio and video interactions are most basic forms of information exchange in a synchronous computer mediated collaboration.

Developing synchronous collaborative Web applications with rich interfaces has proven to be a difficult task. Therefore, evolutive prototyping with small increments might be a good practice. In each increment, a small deliverable is made to the client. That practice allows developers to focus in small defined tasks and also allows clients to be a part of the development process making the requirements more clear.
As shown in Figure 1, applications are developed by the same Domain Engineering disciplines: Analysis, Design, Implementation and Tests. The specific activities that take place in each of these disciplines, regarding Application Engineering, are:

- **Analysis:** application’s requirements are identified, as well as a set of existent services and UI components candidate that might fulfill part of it’s requirements.
- **Design:** the set of candidate services and UI 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).
- **Implementation:** specific application’s functionalities are implemented, services and UI Components are connected to the application’s code.
- **Tests:** after unit testing application’s specific functionalities, integration, system, and acceptance tests take place with the selected services and UI Components already in place.

After 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 by 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 into three tiers, analogously to the domain components division: **View**, **Logic** and **Persistence**. Each tier defines sets of responsibilities and depends only of the direct lower tier. Messages are exchanged in a top-down fashion without jumps between tiers. The dependency among tiers are managed by an IoC container. The design pattern IoC lowers coupling, allowing, for instance, a whole tier switch by changing the XML configuration file.

**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 in a 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 and application specific requirements. Notice that, except for the minute part, most of the requirements of **Meeting** are covered by the existent components. Through reuse, it is possible to focus mostly on the application’s specific requirements.

**Main JSF page**, as seen in Figure 5, gathers all UI components reuse while **Minute JSF page** is responsible for the minute UI. **MeetingLogic** and **Meeting-DAO** are devoted to the minute’s functionalities only, reusing existent services to access user informations and persistence functionalities respectively. Notice that application’s developers don’t need to know the details of the reused UI components and it’s complexities related to audio, video, text and whiteboard.
Listing 3 shows the reuse of the UI components by the Meeting application in the Main JSF page\(^4\).

\[
\begin{verbatim}
%@ taglib uri="http://br.fapesp.tidia.ae.ccsw20/jsf/rwic" prefix="rwic" %>
%@ taglib uri="http://br.fapesp.tidia.ae.ccsw20/jsf/riscom" prefix="riscom" %>
<f:view><h:form>
  <h:commandLink value="Leave room" action="finish" style="...">
  <riscom:whiteboard participants=#\{MeetingLogic.participants\} editAll="true" style="...">
  <riscom:chat participants=#\{MeetingLogic.participants\} moderate="false" style="...">
  <rwic:publishAV serverUrl=#\{MeetingLogic.MSSServerUrl\} stream=#\{MeetingLogic.userId\} moderate="false" debug="true" style="...">
  <riscom:mosaicAV serverUrl=#\{MeetingLogic.MSSServerUrl\} streams=#\{MeetingLogic.participantsId\} moderate="false" debug="true" style="...">
</h:form></f:view>
\end{verbatim}
\]

**Listing 1.3.** Main JSF page code.

Figure 6 shows a screenshot of the Main JSF page. A whiteboard component is shown in the top left of the Figure and a Chat component is shown in the top right. On the bottom, a PublishVideoWindow component and a Mosaic component are exhibited.

\(^4\) The code listed has been simplified to emphasize the use of RWIC and RISCom tags.
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 those services used in developed applications, including Meeting, are: JSF and ZK framework for UI, Hibernate as ORM framework, MySQL as DBMS, JMS messaging service, Red5 for audio and video streaming service, Tomcat as application container, and Spring as IoC manager.

4 Related Work

Web 2.0 synchronous collaborative works with software reuse concerns are still rare in literature. However, many synchronous collaborative development approaches were proposed in the past, particularly in the groupware toolkits category. Some of them are Groupkit [16], COAST [17], Clockworks [9] and Rezervous [15]. Although those 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 don’t 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 [20] 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 flexibility 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 Web 2.0 applications. 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 such applications.

Some Tidia-Ae applications were developed using the described 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 results in the application development process, an experienced Java developer, with the development environment set, took approximately 4 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 87% of code reuse.

It is possible, in a well delimited scope and with a defined process, to attain interesting benefits using component based development. The construction of a components library was not an easy task, it demanded technical as well as domain vision, planning and knowledge. As result, the development of applications became faster and simpler.

One interesting aspect learned from this domain is that UI development is responsible for over 60% of development efforts. This is due to many factors such as HTML lack of support for richer set of UI components, browsers rendering incompatibilities and poor AJAX debugging support. Because of such difficulties, large part of our effort was focused in developing UI web components that can be easily used. The UI tag libraries proposed in this work allows developers to use those components simply by adding tags to the source code as showed in Listings 2 and 3.

6 Acknowledgements

We thank FAPESP TIDIA-Ae, process 2005-2010/60653-1, that provided support to the development of this work.

References