Component Architecture for Building Web-based Synchronous Collaboration Systems

Lukasz Beca, Geoffrey C. Fox, Marek Podgorny
Northeast Parallel Architectures Center
111 College Place; Syracuse University
Syracuse, New York 13244
{beca, gcf, marek}@npac.syr.edu

Abstract

The component technology gains popularity as its benefits, the reusability and the simplicity of use become more and more evident. In this paper we demonstrate how component technology can be effectively applied to the process of building collaborative applications in the Web environment. In order to support development of collaborative tools the components must satisfy a set of requirements inherent for a synchronous collaboration environment. Distribution of arbitrary events and objects, access to the collaborative session state, and means of synchronizing operations on the shared resources must be provided. Tango Beans is a set of components based on the Tango Interactive framework that has been implemented to facilitate rapid development of collaborative applications. We explain how Tango Beans work and how they can be used to develop collaborative tools for synchronous distance learning and Web conferencing.

Keywords

Components, collaboration framework, synchronous collaboration, Java

1. Introduction

The construction of a collaboration system is a complex and difficult process. Collaborative systems operate in distributed, multi-user environments, and are, as a rule, more difficult to design and implement than ordinary, monolithic desktop applications. The simplest approach to building collaboration systems is to re-use already existing applications and make them collaborative by sharing their display among users [6]. Unfortunately, this solution has many limitations. The applications provide the same application interface to all the users and as a result they lack flexibility necessary to accommodate user's specific needs (e.g. application interface for the teacher should be different from student's application interface) so it is difficult to build well integrated collaborative system from such applications.

Very often domain specific collaborative tools must be used that cannot be implemented by simply sharing display of the application. For example, a chat or an application signalizing that student wants to ask a question during a distance leaning session do not have corresponding standalone applications. The complexity of developing such collaboration-aware applications can be greatly reduced if the component technology is applied. Components are pieces of software that implement single, well defined functionality. This functionality can be accessed via component interface. Each component model provides mechanisms for component interface description. Using those mechanisms the application builder environments can analyze the component and provide information about it to the developer. Currently, two important software component models exist: Java Beans developed by Sun [11] and COM developed by Microsoft [9]. Components developed using the same model can be connected with each other. As a result, they can be created by different software vendors and then assembled into working application using visual programming tools, supported by nearly all integrated development environments (IDE). There are already hundreds of components available on the software market. Currently, most of them are elements of graphical user interface but the number of components that implement other functionality grows. It would be very useful to have generic components that implement also collaboration functionality. Using such components it would be possible to develop collaboration tools easily by combining "off the shelf components" in an application builder environment. This paper discusses the requirements that should be satisfied by components so that they can be used in creation of sophisticated collaborative tools. The requirements result from the analysis of many collaborative applications we have developed for the Tango Interactive system [4]: whiteboards, shared educational applications, chats, multiuser games, and tools for shared multimedia presentations. We describe an example implementation of collaborative components developed to automate building collaborative tools for the Tango Interactive system and created according to the discussed requirements.

2. Requirements for collaborative components

From the perspective of component technology, applications are just sets of the interconnected components. User actions on the graphical user interface components are transformed into the application events. The application events are passed to other components and affect their state. For example, when the button ‘clear’ is pushed in a chat application, the text area component receives the event and clears the entered text. The easiest way to make an application collaborative is to propagate its events to applications operated by other session participants and to forward them to appropriate components. In this way the actions in one application will be mirrored in others. This is especially important for applications that maintain “What You See Is What I See” (WYSIWIS) property [10] (for example shared Whiteboard). Therefore a collaborative component must have the ability to capture and distribute arbitrary application events (not only graphical user interface events) among applications in collaborative session.

Also, a mechanism must be provided to select the events that should be distributed. Very often the application developer does not want to share all the application events. For example, it is not necessary to notify all the session participants that one of the participants chose painting tool in the Whiteboard application. Only content of the drawing area is of interest to all participants and only events that affect the state of the drawing area should be shared.

It must be possible to specify a target or multiple target components for distributed events. Almost always the component that is the source of the event is different from the target component. In complex scenarios one event may affect even several components. There must be a way to define recipient of each shared event so that if the event is distributed among applications in the same collaborative session, it will be automatically delivered to proper components.

Another useful feature is ability to transfer arbitrary data among applications. For example, a drawing created during collaborative session should be transferred to all latecomers. Also text content of a chat session must be somehow distributed among session participants. Therefore some generic mechanism for distributing various objects on demand must be provided.

The collaborative component must provide access to the session state. The session state may contain such information as a list of session participants, a description of the participants (e.g. location, picture, e-mail address), assigned roles (e.g. teacher, student), session resources, floor control status, etc. This information can be used to support collaboration awareness in the application or to assure synchronized access to the limited session resources.

The collaborative components should be as easy to use as ordinary application components. The complexity of the communication mechanisms necessary to provide collaborative functionality should be hidden from the application developer. The collaborative components must be able to communicate with other components to enable simple and quick creation of the new applications. The applications constructed using the components should be easy to test, possibly outside collaborative environment, so that the errors not related to the collaboration functionality can be quickly identified and removed.

The collaborative components should provide intuitive graphical interface when displayed in the application builder environment during application implementation. In this way they can be easily manipulated and composed with other components. However, when the application is being executed, the collaborative components should remain invisible so that they do not clutter the graphical user interface of the application.

The collaborative components used to create application running on the Web should support easy integration with this environment. Usually, if the application is running inside a Web browser, the communication with the applications running on other hosts requires implementation of the browser specific security policy. If the components can encapsulate implementation of such policy the application becomes more portable and generic.

3. Tango Beans

Tango Beans is a set of components that form an API for the Tango Interactive (TI) system. They provide high level interface to the collaboration services offered by the TI framework. Tango Beans consist of two components: *TangoBean* and *ObjectPipe*. Their main goal is to enable easy creation of collaborative applications for the TI environment using visual programming tools. An application constructed using Tango Beans can communicate with other applications started in the same collaborative session. It can also access session information. Tango Beans are based on Java Beans – software component model technology for Java. They define generic interfaces that enable communication with other components developed according to Java Beans specification. Tango Beans were designed and developed according to the requirements presented in the previous section.

3.1. Tango Interactive collaboration framework

Tango Interactive is a generic collaboration framework that can be used to construct various systems for synchronous collaboration. It is also a Web-based framework, which means that system based on TI can be accessed from and started in a Web browser, it can interact with Java applets and JavaScript applications, and it can extensively use Web resources. The TI framework is quite complex and it supports various modes of collaboration. We present here a simplified description essential for understanding further sections of this paper.

The main components of the framework are the Collaborative Server and the Session Manager. The Collaborative Server maintains state of the system, i.e. the information about logged in users, started applications, established sessions etc. It also distributes state changes and routes messages among Session Managers. The Session Manager is an application (implemented as Java applet) that users can use to log into the system, start or terminate collaborative applications, view collaborative sessions currently established by other users and join those sessions. All actions performed by Session Manager that affect the state of the system must be accepted by the Collaborative Server. The following concepts have special meaning in the TI framework:

Application is a collaborative tool used by the user to communicate with other users. Chat or Whiteboard are examples of TI application.

Session is a set of TI applications of the same type that exchange information between each other. Each instance of the application belongs to a different user. Exchange of information is achieved by sending and receiving messages e.g. chat session enables exchange of text information between chat applications that are started in the same session. The communication between applications in the same session is transparently supported by the TI framework. The sessions can be established using TI Session Manager.

Participant is a user that takes part in a session. Participant exchanges information with other participants using applications. Each participant can have only one application running in a session, but a user can participate in multiple sessions at the same time.

The TI framework provides programming interfaces for various types of applications: Java applets, Java applications, JavaScript applications, C++ applications and LISP applications. By choosing appropriate set of applications it is possible to construct collaborative system customized for specific application domains. Currently, the systems based on the TI framework are used for general collaboration and distance learning.

3.2. Tango Bean

When present in the application, *TangoBean* turns a single-user application into a collaborative tool. The main function of *TangoBean* is to integrate the application with the TI system and to enable control over the application from the Session Manager. This ensures that the application can be started and terminated by the TI Session Manager. Moreover, the application will be able to receive events from TI system and send its own events to the system. Finally, the application will be able to access information about current state of the session and receive notifications about changes in this state.

In order to provide described functionality *TangoBean* communicates with TI framework and with components of the application. With support of the *ObjectPipe* components, *TangoBean* intercepts events that should be sent to other applications and forwards them to TI framework - see Figure 1. It also accepts events from TI and sends them to appropriate application components using *ObjectPipe* components.

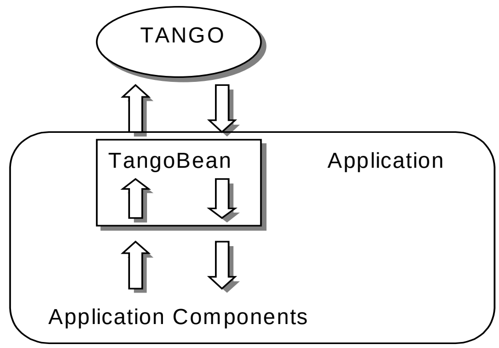

Figure 1: Event flow when the application with a *TangoBean* is placed in the TI environment

TangoBean may run in one of three different modes: collaborative mode, testing mode, and design mode. In collaborative mode all events received by TangoBean from the application components are forwarded to the TI system and the events received from the TI system are distributed to appropriate application components as in Figure 1. *TangoBean* enters collaborative mode when the application that contains *TangoBean* runs in the TI environment.

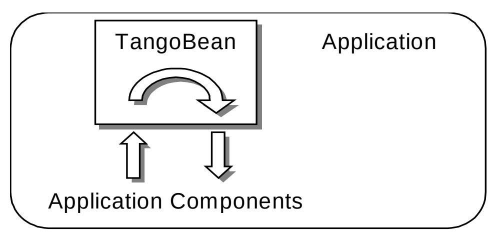

Figure 2: Event flow when the application with *TangoBean* is in testing mode

If *TangoBean* is unable to detect the TI system, it enters testing mode illustrated in Figure 2. In this mode, all messages sent to *TangoBean* from the application components are looped back and returned to the application. This mode enables application testing outside the TI environment e.g. in an applet viewer or in a Web browser. In the collaborative mode and the testing mode *TangoBean* is invisible.

The third *TangoBean* mode of operation is design mode. *TangoBean* enters this mode when the application is being constructed in an integrated visual development environment. In this state the component presents graphical interface so that it can be manipulated using various application builder tools. *TangoBean* switches to appropriate mode automatically depending on the detected runtime environment. The three different modes of operation are necessary to support all stages of application development process.

3.3. Object Pipe

ObjectPipe is a component that supports communication between components placed in different applications. The *ObjectPipe* components accept data of any type. Their primary task is to capture and distribute application events to other application instances in the session. They can be also used to distribute objects of arbitrary type among applications.

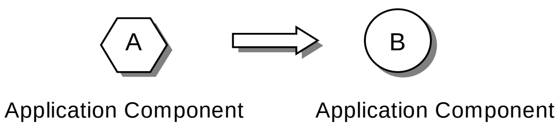

Figure 3: Usual event flow in application

In a typical application, a component sends events to another component in order to initiate some action. This situation is illustrated in Figure 3. In order to make the application collaborative we must be able to intercept some or all of the events generated in one instance of the application and send them to other instances. *ObjectPipe* grabs events generated by the application components and distributes them to the target components in the applications that belong to other session participants. For example, event generated by the click on the button can be sent to all or to selected applications in the session. In other words, *ObjectPipe* intercepts the button event, distributes it, and notifies interested components when such event arrives. This situation is illustrated in Figure 4. The general structure of the application remains unchanged, only the *ObjectPipe* component is placed between components A and B. Note that components A and B are placed in the application instances started by different users. Some applications need several *ObjectPipe* components to establish distinct channels of communication between different application components.

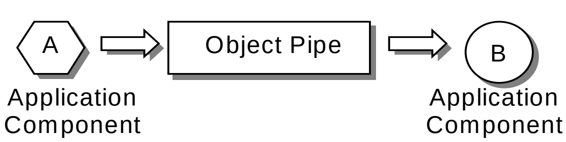

Figure 4: Event flow between components with *ObjectPipe* introduced

ObjectPipe must be connected to an instance of *TangoBean* to send and receive messages from the TI system. The *ObjectPipe* component is visible only during the application design time.

3.4. Context initialization

The Tango Beans components and the application components must be appropriately connected to enable distribution of desired events and objects. All connections must be defined during application development in visual application builder environment. At this stage, the developer makes selection of events and objects that will be transmitted using *ObjectPipe* components and defines target components for those events and objects.

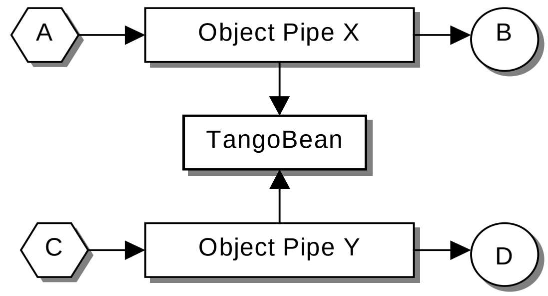

Figure 5: Connections between Tango and application components

The direction of the connections is significant. It determines the way the events and object are passed between the components. The application component that is a source of the events (to be shared) should be connected to *ObjectPipe* and the *ObjectPipe* component should be connected to the application component that is supposed to accept events (target component). If *ObjectPipe* is connected to more than one component, all of those components will receive the copy of the event.

Each *ObjectPipe* component must be connected to *TangoBean*. When the collaborative application starts, the *ObjectPipe* components register themselves in *TangoBean*, which assigns them unique identifiers in the scope of the application. From that moment *TangoBean* can pass the events from the *ObjectPipe* components to the TI system and route events arrived from the TI system to the appropriate *ObjectPipe* components.

Figure 5 presents all necessary connections between the application components A, B, C, and D, *ObjectPipe*s X and Y, and *TangoBean*. The established connections enable distribution of events generated by component A to component B and events generated by component C to component D. The connections can be created using tools provided by the application builders. When all the connections are completed, the corresponding code is automatically generated.

3.5. Event flow

All applications that run in the same TI session are generated using the same code therefore the *ObjectPipe* components used in different instances of the same application have the same identifiers if they appear in the same context. For example, let's consider the collaborative session with two participants – Mary and John. They use an application that distributes the event generated by clicking the 'start' button. The *ObjectPipe* component that receives events from the 'start' button, will have the same identifier in Mary's and John's instances of the application.

All *ObjectPipe* components with the same identifier can be referred to as peer components. Each *ObjectPipe* component in any instance of the application can communicate with the peer *ObjectPipe* components in other instances of the application. All events that are fired in the application and are intercepted by *ObjectPipe* will be delivered to all or selected peer pipes.

If all connections among the application components are established correctly, the events can be routed through the resulting communication framework. Figure 6 presents event flow between two applications in the same session (assumed connections between components as in Figure 5). The event can be an object of arbitrary type that is sent between application components as a result of the action executed by one of the components. An event created by component A is routed to component B as follows:

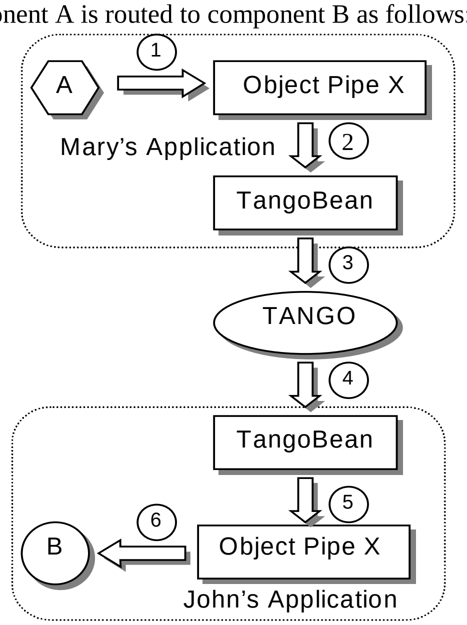

Figure 6: Event flow between applications in the same session

1. Component A sends event to *ObjectPipe* X: Application component A in Mary's application is connected to *ObjectPipe* X and it can fire events that will be intercepted by the pipe.

2. *ObjectPipe* X sends event to *TangoBean*: *ObjectPipe* X in Mary's application encodes event received from the component A. In process of encoding *ObjectPipe* X adds its own pipe identifier to the event data. It passes obtained event to *TangoBean*. *ObjectPipe* X does not forward the event to the component B present in Mary's application.

3. *TangoBean* sends event to TI system: *TangoBean* sends the event to the TI system using low level TI API. TI system distributes the event to all participants in session except for the user that generated the event.

4. *TangoBean* accepts event from TI system: *TangoBean* embedded in the John's application accepts the event from the TI system.

5. *TangoBean* sends event to *ObjectPipe* X: *TangoBean* in John's application finds the *ObjectPipe* registered with the same identifier as the identifier extracted from the event. It is a peer of the *ObjectPipe* X in Mary's application. The event is routed to this *ObjectPipe.*

6. *ObjectPipe* X sends event to Component B: *ObjectPipe* X in John's application sends the event to the component B.

The events can be sent to all applications in the session or to the selected group of the applications. Appropriate method of *ObjectPipe* should be used to select the set of the event recipients.

3.6. Access to the session state information

Tango Beans enable access to the session state information. Application components can retrieve useful data such as a list of session participants, the name of the session master or the floor control status, by interaction with *TangoBean* component. Session information can be accessed in two ways: (1) Application component can ask directly *TangoBean* about data of interest by calling appropriate methods. (2) It can also register itself in *TangoBean* using listener interface and receive asynchronous notifications about the session state changes. The application component does not have to receive all session events. It may choose to receive only selected notifications.

3.7. Development process

Tango Beans must be installed in an IDE before they can be used. When this step is completed, Tango Beans are displayed together with other available components and can be used to build the application. This can be done by visual manipulation of the components. When the application is created it can be executed and tested directly in the IDE, Applet Viewer or Web browser. At this stage application errors can be discovered and corrected. *TangoBean* and *ObjectPipe* objects will enter testing mode automatically. In the next step a distribution package for application is produced using standard tools available in integrated development environments. The package should be signed digitally if the application is supposed to work in a Web browser. Finally, the package should be registered in the TI Session Manager so that it can be used in the TI environment. When registered, the application can be started, terminated, and managed from the TI Session Manager.

4. Related work

So far collaborative components are not widely used for building collaborative tools. A similar work was presented by IBM researchers [3], who created a component suite for synchronous collaboration based on the Live system. This set of components supports broad collaboration functionality. In addition, a number of GUI components were designed that provide commonly used functionality in collaborative applications.

There are many frameworks available for constructing synchronous collaboration environments. Habanero developed at NCSA [7] and Java Shared Data Toolkit developed by Sun [5] are examples of the most sophisticated systems for Java platform. However, they do not provide component based APIs.

Significant work was also done in area of automatic sharing of application events. For example *collawt* package [1] enables distribution of events by replacement of all the components of Java GUI by their collaborative versions. However, this approach does not provide flexibility of selecting events to be distributed.

5. Implementation status

Early prototype of Tango Beans was presented at NCSA Alliance'98 [2] where the development of simple collaborative application was demonstrated as a part of the Tango system tutorial. Currently, Tango Beans API with full functionality is a part of release of Tango Interactive 2.0 system [8].

6. Conclusions

Component technology offers significant advantages in construction of tools for synchronous collaboration. It hides complexity of collaboration functionality implementation from the developer and offers reuse of existing components to create richer and customized collaborative tools. Using this approach, sophisticated collaborative applications can be relatively easily created. The software industry increasingly adopts component standards and we can expect more similar work in near future with extended functionality.

The solution presented in this paper – Tango Beans – reflects specific needs of collaborative tools created for the Web. Tango Beans offer significant benefits for developer of collaborative systems. They enable quick development of sophisticated collaborative tools by using visual programming methods. In implementation process large number of components available on the market can be used. Developer can choose the events and the data to be shared so that even complex collaborative behavior can be implemented. Testing outside collaborative environment is supported for easy application debugging.

Tango Beans have several limitations. They can be used only to develop Java applets and applications. For other languages, such as C++, other set of collaborative components must be implemented. However, since our target is Web environment, this limitation is not critical. Also, the current Tango Beans architecture requires all collaborative applications in the same session to be generated using the same code. This is necessary condition for proper automatic registration of the *ObjectPipe* components in the *TangoBean* component. This constraint will be removed in future versions of Tango Beans.

7. Acknowledgements

The work presented in this paper was funded by the DoD High Performance Computing Modernization Program CEWES Major Shared Resource Center through Programming Environment and Training, contract number: DAHC 94-96-C-0002. Tango Interactive system was designed and developed by Tango Interactive group at NPAC at Syracuse University. Many thanks to Grzegorz Lewandowski for his helpful comments.

8. References

[1] Abdel-Wahab, H., Kvande, B., Nanjangud, S., "Using Java for Multimedia Collaborative Applications", *Proceedings of PROMS'96: Third International Workshop On Protocols for Multimedia Systems, 1996*, Madrid, October 1996, pp. 49-62.
[2] Alliance'98, http://alliance.ncsa.uiuc.edu/alliance98/.
[3] Banavar, G., Doddapaneni, S., Miller, K., Mukherjee, B., "Rapidly Building Synchronous Collaborative Applications by Direct Manipulation", *Proceedings of the ACM 1998 conference on Computer supported cooperative work*, 1998, pp. 139-148.
[4] Beca, L., Cheng, G., Fox, G. C., Jurga, T., Olszewski, K., Podgorny, M., Sokolowski, P., and Walczak, K., "Java enabling collaborative education, health care, and computing", *Concurrency: Practice and Experience*, Vol. 9(6), June 1997, pp. 521-533.
[5] Burridge, R., Java Shared Data Toolkit User Guide, Version 1.4, Sun Microsystems, http://www.sun.com/software/jsdt/techinfo/index.html, 1998.
[6] Lauwers, J.C. and Lantz, K.A., "Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems", *Proceedings of CHI'90*, ACM, 1990, pp.303-311.
[7] NCSA, Habanero Project, http://www.ncsa.uiuc.edu/SDG/Software/Habanero/.
[8] Northeast Parallel Architectures Center at Syracuse University, Tango Interactive, http://www.npac.syr.edu/tango
[9] Rogerson, D., Inside COM, Microsoft Press, Redmont, Washington, 1997.
[10] Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tatar, D., "WYSIWIS Revised: Early Experiences with Multiuser Interfaces", ACM Transactions on Office Information Systems 5(2), 1987, pp. 147-167.
[11] Sun Microsystems, The JavaBeans 1.01 Specification, http://java.sun.com/beans/.