JavaBeans as a Framework for Collaborative Software

James "Bo" Begole *Philip L. Isenhour* *Clifford A. Shaffer*
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
{begolej, isenhour, shaffer}@cs.vt.edu

ABSTRACT
We describe the use of a popular component framework, JavaBeans, to facilitate developing collaborative software. We describe the class of collaboration-unaware components that can be effectively shared under this framework, and an extension that supports development of efficient collaboration-aware components.

KEYWORDS: Computer-supported cooperative work, groupware, component frameworks, Java, JavaBeans.

INTRODUCTION
By facilitating assembly of re-usable components, development frameworks such as Microsoft's Object Linking and Embedding/Component Object Model, IBM and Apple's OpenDoc, and Sun's JavaBeans are intended to ease application construction. This paper describes leveraging a popular component architecture, JavaBeans, for a different purpose: to facilitate developing collaborative software. We felt that developers already familiar with a popular component framework would find using it easier than developing collaborative software *ad hoc,* or by learning and integrating a third-party groupware toolkit. Although several groupware toolkits exist, none are based on a well-known framework. We find that aspects of JavaBeans make it surprisingly simple to create collaborative software components.

Java simplifies distribution of software and largely eliminates platform compatibility issues. While these characteristics are useful for software in general, they are critical for collaborative software.

JAVABEANS AND BOUND PROPERTIES
In JavaBeans, a **property** is a "named attribute associated with a bean that can be read or written by calling appropriate methods on the bean [5]." Typically, a bean exposes its state through its public properties. A property is declared by matched `get` and `set` methods. For example, an object can expose a property named `background` by simply providing methods named `getBackground` and `setBackground`, with appropriately typed return values and parameters. Using the Java Reflection package, JavaBeans allows dynamic querying and setting of properties at run time.

The JavaBeans specification encourages developers to not only expose attributes as properties, but also to provide notification of property changes to registered listeners. When a property changes, a bean can generate a `PropertyChangeEvent` describing the change and deliver it to registered listeners. A property that fires such an event is called a **bound property,** because its value can be bound to a property of another object.

COLLABORATING BEANS
Our approach to collaboration is based on a replicated architecture, where each collaborator maintains a copy of the shared data. The JavaBeans mechanisms described previously give us the necessary tools to replicate component state at the granularity of properties. Because the framework can dynamically query and set properties, a developer no longer needs to encode state changes in a message protocol; the framework can extract the necessary information from a change notification and construct messages for distribution.

Our collaborative JavaBeans framework, realized in a system called **Sieve** [3] (Figure 1), listens for property changes from each bean component. When a `PropertyChangeEvent` is generated, **Sieve** packages it in a message that is sent to the corresponding replicas. These messages are sent in the same order to each replica using a communications package specifically designed for collaborative software, called Java Shared Data Toolkit (JSDT) [1]. When such a message is received, the local replica of the changed component is found, and the changed property is set to the new value.

Late-comers are brought up to date by replaying the record of property changes. Replay time is minimized by keeping only the most recent property changes. We prefer this to an alternative of sending a copy of a shared object using Java Object Serialization (JOS), because JOS requires specific development effort and snapshots would disrupt other collaborators' work.

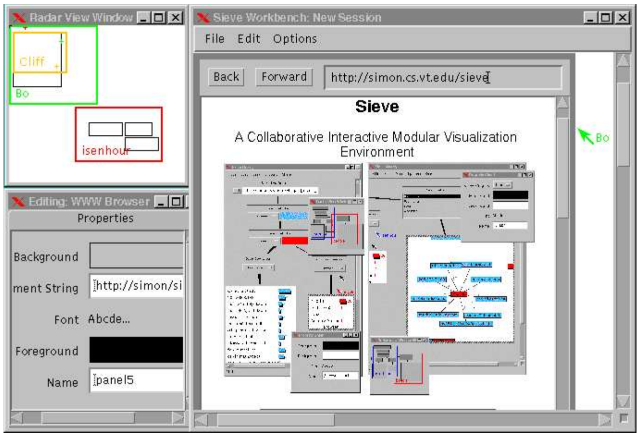

Figure 1: A HotJava Browser bean shared in the Sieve workspace window (on right). Properties may be edited on a bean directly or in the lower left window. Each collaborator's view is indicated by a uniquely colored rectangle in the upper left window (radar view).

Conflict Resolution

It is possible for two or more collaborators to generate potentially conflicting property changes. To resolve this, we use the strategy described by Karsenty and Beaudouin-Lafon [4], where potentially conflicting operations mask, commute or are order-specific. Typically, property changes are **maskable.** That is, when two operations occur in sequence, the result is the same as when only the last change is applied. Examples of typical maskable properties are color, length, position, and size. Maskable properties are handled by ensuring that only the last property change is applied to each replica.

In some cases, property changes are not maskable. They may be **commutative,** where the result of applying a series of operations is the same regardless of the order, or **order-specific,** where the final state depends on the order. For both cases, the component developer must be aware that the component will be shared. Instead of firing a simple `PropertyChangeEvent`, the component must generate a specific type of event that is recognized by the **Sieve** environment, and must handle potentially conflicting events. This can be trivially implemented by not applying the change locally until the corresponding message returns. More efficient algorithms are known, such as the distributed operational transformation algorithm (dOPT) of Ellis and Gibbs [2].

Collaboration Unaware and Aware Components

Many useful collaboration-unaware components may be developed quickly and shared with our approach. For example, we implemented a wrapper of the HotJava Bean component that consists of less than 100 lines of code. Thus we created a collaborative version of a powerful WWW browser with little work.

For some components, however, property changes may not provide sufficiently fine granularity. Consider a component that implements a simple text editor. One approach to sharing it would be to expose the entire text content as a property. Any keystroke that changed the content would then cause a property change, and the entire content would be propagated to all replicas. This approach might be sufficient for small chunks of text, but would not be desirable once the content grew to more than a few words. A better solution is to send only a light-weight description of each change. JavaBeans encourages components to generate an event as notification of any such change. This event can be distributed using the same mechanism as a `PropertyChangeEvent`, and if it contains sufficient information, a collaboration-aware wrapper can merge the change into each replica's content. Component-specific collaboration support code must be written to handle the event, but the component itself remains collaboration-unaware.

CONCLUSIONS

We explored JavaBeans as a framework for developing collaborative software. We found that when a component's state can be efficiently defined by a finite set of bound properties, it can be shared with no extra development effort. A collaborative version of the HotJava browser illustrated that many useful components may be shared with little implementation effort. For those cases where collaboration-aware components are desired, mechanisms can be developed explicitly to create efficient collaborative components.

Sieve is available at http://simon.cs.vt.edu/sieve.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the Department of Education under grant P116B60201 and the National Science Foundation under grants DGE-9553458 and REC-9554206.

REFERENCES

1. R. Burridge. Java ™ Shared Data Toolkit User Guide. Sun Microsystems, Inc., Mountain View, CA, Apr. 1998.

2. C. Ellis and S. Gibbs. Concurrency Control in Group Systems. *SIGMOD Record*, 18(2):399–407, June 1989.

3. P. Isenhour, J. B. Begole, W. S. Heagy, and C. A. Shaffer. Sieve: A Java-Based Collaborative Visualization Environment. In *IEEE Visualization '97 Late Breaking Hot Topics Proceeding*, pages 13–16, Oct.22–24 1997.

4. A. Karsenty and M. Beaudouin-Lafon. An Algorithm for Distributed Groupware Applications. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 195–202, Pittsburgh, PA, May 1993. IEEE Computer Society Press.

5. JavaBeans ™ 1.01, last access: Apr. 15, 1998. <http://-splash.javasoft.com/beans/docs/beans.101.pdf>.