

Aspect-oriented programming for COLLAGEN-based applications

Brian D. Ferris, Nihar M. Namjoshi, and Robert St. Amant

Department of Computer Science

North Carolina State University

EGRC-CSC Box 7534

Raleigh, NC 27695-7534

bdferris@eos.ncsu.edu, nmnamjos@eos.ncsu.edu, stamant@csc.ncsu.edu

ABSTRACT

Collagen provides collaborative, mixed-initiative interaction support for Java applications that assist users in complex problem-solving tasks. Our interest is in the integration of Collagen with existing Java applications, such that the power of Collagen becomes available without extensive reworking of existing code. We describe how Aspect-Oriented Programming techniques can address this issue at the program design level.

KEYWORDS

Collagen, Aspect-Oriented Programming

INTRODUCTION

Concepts and techniques for mixed-initiative interaction have only recently entered the mainstream of research on human-computer interaction. Despite their promise, we are aware of no mixed-initiative systems in wide use today. Such systems are harder to build than conventional systems, because their effectiveness relies on inference about difficult-to-specify properties of interaction, such as implicit task models, user goals, and user preferences.

Collagen is a Java toolkit that alleviates many of the difficulties in developing mixed-initiative interface agents [5]. Collagen supports collaborative interaction in several ways; it provides algorithms for discourse interpretation, plan recognition, and plan generation. Around a dozen Collagen applications have been described in the literature, several in domains of significant complexity, including travel planning, tutoring, and graphical interface design [4].

Our recent work has focused on integrating Collagen with an interactive statistical system that we call the Data Assistant [6]. The Data Assistant supports basic probability and statistical inference techniques including estimation, hypothesis testing, and simple regression analysis, available through a conventional graphical user interface. Although the Data Assistant was developed to be integrated with an

intelligent assistant, it was not designed with Collagen specifically in mind. Our main concern was the difficulty of modifying an existing, relatively complex application to accommodate the integration. Over the course of our development, we identified three levels for integration between Collagen and an application:

Low: A minimally integrated application notifies the Collagen agent of interface manipulation events and gives the agent the ability to perform manipulations itself. For example, when a user clicks a button, the agent is notified, and additionally, the agent may click the button on its own.

Medium: An additional level of integration is reached when the application makes internal data objects visible to the Collagen agent. The exposure of such data structures may range from offering machine representations of data already available to the user through the graphical interface to less public structures not visible to the user. By making various application data available in programmatically accessible ways, developers can avoid the complex task of pulling data directly from the graphical interface [2] and allow for refined interaction based on the internal state of the application.

High: A yet higher level of integration allows the Collagen agent to perform manipulation actions using direct programmatic interfaces of the application. Such capabilities allow the agent to reason based on perceived events and application state, as exposed in our previous levels of integration, and to initiate relevant calls to application interfaces with collected data. Such performances would bypass the direct manipulation interface exposed in our minimal definition of integration, allowing the agent considerable flexibility in interaction.

Not surprisingly, the necessary modifications to add each level of Collagen integration become non-trivial for a large application with a large number of interface manipulation widgets and diverse internal data structures. Additionally, these modifications often intrude on established boundaries of data flow and protection. Finally, as Collagen is further integrated into an application, it can become exceedingly difficult to run the application outside of a Collagen environment.

*LEAVE BLANK THE LAST 2.5 cm (1") OF THE LEFT
COLUMN ON THE FIRST PAGE FOR THE
COPYRIGHT NOTICE.*

We have attempted to address these issues in our work with the Data Assistant. To give the flavor of user interaction, when the user groups together statistical results (in an analog of a Data Mountain), the Collagen agent prompts the user to run correlation tests on the variables involved, to identify potential new relationships. In scenarios like these, the agent needs to know about potentially dozens of operators that may be relevant, as well as internal properties of the data that guide operator selection. To support integration with Collagen, without extensive additions to the Data Assistant, we have developed an approach based on aspect-oriented programming [3].

INTEGRATION WITH ASPECTJ

Aspect-oriented programming involves identifying functional concerns that span distinct application classes; common examples include logging and authorization mechanisms. Aspect-oriented programming allows for the consolidation of these typically scattered concerns into single modules, the identification of points where these concerns should interact with the application, and finally the automatic integration of the appropriate links and triggers to combine these crosscutting concerns with the rest of the application.

The task of identifying where aspect-oriented concerns will interface into an application is achieved with pointcuts, a grammar for matching method prototypes. For example, the following pointcut matches all method calls in the Example package with no arguments and a void return value:

```
pointcut examplePointcut() : call( void Example.*() );
```

Pointcuts are then linked to advices, bodies of code that get executed when a pointcut is triggered in an application. Continuing our example, we define the following advice:

```
before() : examplePointcut() {
    System.out.println( "Before call to Example.* method." );
}
```

This advice will execute the advice's body before any examplePointcut is triggered. There are a variety of ways to schedule advices with respect to their pointcuts, with options for before, after, around, and others which we will not discuss here. Collections of pointcuts and advices are bundled together into aspects, which are roughly analogous to classes in object-oriented programming. Aspects are then compiled alongside existing source code, and finally woven into a complete binary, with aspect advices being inserted into the application source during preprocessing.

The ability to use aspect-oriented methodologies to cross-cut concerns is directly applicable to the task of Collagen integration. In our Data Assistant application, we have a diverse collection of events, objects, and interfaces that need a common interface to the Collagen application adapter. While traditional object-oriented techniques would involve adding functionality to each distinct family of classes of event mechanisms and data entities, aspect-

oriented techniques allow Collagen adapter code to be centralized across all application concerns.

Aspect-oriented programming is especially appropriate for Collagen application integration because of the AspectJ toolkit (<http://eclipse.org/aspectj/>), which provides fully-integrated aspect-oriented support for Java. AspectJ provides an aspect-aware Java compiler that parses aspect pointcuts and advices, in the end producing byte-code that can be run under a standard Java virtual machine and, more importantly, with Collagen.

EVENT NOTIFICATION

We begin with direct-manipulation interface event notification. In a typical Java Swing application, the application is notified of manipulation events, such as a button-press, by registering an ActionListener object with each manipulation component. Each distinct component requires a distinct ActionListener implementation, each with an appropriate actionPerformed() method. We can easily receive notification every time an ActionListener(). actionPerformed() method is called using an AspectJ crosscut.

```
before( ActionEvent e ) :
    execution( void *.*actionPerformed( ActionEvent ) ) && args( e )
{
    AbstractButton button = (AbstractButton) e.getSource();
    adapter.perform(
        new PushButtonManipulation( getCurrentActor(),
            AbstractButtonTerm.make(button) ) );
}
```

This single pointcut allows us to capture every single call to every single implementation of the ActionListener.ActionPerformed() method within our application, notifying the application adapter and subsequently Collagen of all manipulation performances with no modification to the application source code. Additional pointcuts can easily be defined to capture other semantic events, such as combo box selection, list scrolling, and even mouse movements. No object-oriented technique could so easily provide similar functionality with the same level of intrusiveness.

DATA EXPOSURE

Now consider a level of Collagen integration that requires access to internal application data structures. We can utilize the static crosscutting feature of AspectJ to provide our ApplicationAdapter privileged access to otherwise private internal data members. Consider an Application class with two private members, viewActor and modelActor, respectively. We can provide public accessors to these members using a static crosscut.

```
public privileged aspect ApplicationCrosscuts {
    public ModelActor Application.getModelActor() {
        return this.modelActor;
    }
}
```

```

public ViewActor Application.getViewActor() {
    return this.viewActor;
}
}

```

The privileged aspect adds two accessors to previously hidden members of the Application class, which our application adapter could use to pass on relevant data to Collagen. Additional static crosscuts for exposing internal application data can all be consolidated in a single aspect module, instead of spreading such modifications across the various classes of the application. Removing the additional accessors then becomes as easy as removing one module.

INTERFACE EXPOSURE

The previous examples of event notification and data exposure can be combined into a single methodology that lets us achieve our deepest level of integration: exposing Collagen to internal application interfaces. Consider the following interface definition:

```

public interface Analysis {
    public ResultSet performAnalysis( DataSet dataSet );
}

```

We can model this interface with the following Collagen manipulation definition:

```

manipulation PerformAnalysisManipulation {
    gloss { "perform an analysis on the {#dataSet} data set" }
    parameter DataSetTerm dataSet;
    result ResultSetTerm result;
}

```

We can then capture performances of this manipulation for Collagen notification with the following pointcut:

```

after( DataSet dataSet ) returning( ResultSet result ) :
    call( ResultSet Analysis.performAnalysis( DataSet ) )
    && args( dataSet )
{
    adapter.perform( new PerformAnalysisManipulation(
        getCurrentActor(),
        DataSetTerm.make( dataSet ),
        ResultSetTerm.make( resultSet ) ) );
}

```

This pointcut exposes Collagen to both internal application events and internal data structures. Collagen can additionally request subsequent PerformAnalysis-Manipulation performances, with the application adapter calling the appropriate internal interfaces on exposed implementors of the Analysis interface.

DISCUSSION

Our work is similar in spirit to the VAMPIRE system [1], in its emphasis on integrating IUI techniques into a conventional development process familiar to designers and

software engineers. Aspect-oriented programming techniques with AspectJ provide a powerful method, well-understood within software engineering, of integrating a complex application into the Collagen framework. A few lines of code can capture event notification for hundreds of widget manipulations, all without modification of existing application source. Additionally, more complex application data and interface exposure is possible with appropriate pointcuts to the Collagen application adapter. Because pointcuts can be selectively applied as recipes require, aspect-oriented techniques find a good balance between flexibility and performance, versus agent-application frameworks that attempt to gather all interaction events at the expense of application performance [2].

There are drawbacks to aspect-oriented techniques that apply to all uses, not just Collagen integration. For complex pointcuts and static crosscuts, it can become difficult to track which aspects are modifying which application code. Consistent coding conventions and aspect browsing tools can simplify the task.

Further work in this area will focus on automatic generation of Collagen manipulation classes based on pointcuts of application interfaces, along with exploring the possibilities for enhanced agent-user collaboration that such deep Collagen integration allows.

ACKNOWLEDGMENTS

Thanks to Mitsubishi Electric Research Laboratories for Collagen, and to Chuck Rich and Candy Sidner for their time and effort in support of this project. This research was supported by the National Science Foundation under award 9988507. The information in this paper does not necessarily reflect the position or policies of the U.S. government, and no official endorsement should be inferred.

REFERENCES

1. J. Eisenstein and C. Rich. Agents and GUIs from task models. In *Proceedings of IUI'02*, pages 47–54, 2002.
2. H. Lieberman. Integrating user interface agents with conventional applications. In *Proceedings of IUI'98*, pages 39–46, 1998.
3. G. Kiczales et al. Aspect-oriented programming. In *Proceedings of ECOOP'97*. Pages 220–242, 1997.
4. C. Rich, N. Lesh, J. Rickel, and A. Garland. A plug-in architecture for generating collaborative agent responses. In *Proceedings of AAMAS*, pages 782–789 2002.
5. C. Rich and C. L. Sidner. COLLAGEN: a collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3/4), 1998.
6. R. St. Amant, M. D. Dinardo, and N. Buckner. Balancing efficiency and interpretability in an interactive statistical assistant. In *Proceedings of IUI'03*, pages 181–188, 2003.