Using a Lightweight Workflow Engine in a Plugin-Based Product Line Architecture

Humberto Cervantes and Sonia Charleston-Villalobos

Universidad Autonoma Metropolitana-Iztapalapa (UAM-I),
San Rafael Atlixco Nº 186, Col. Vicentina, C.P. 09340, Iztapalapa. D.F., Mexico
{hcm, schv}@xanum.uam.mx

Abstract. This paper presents a software product line architecture where applications are assembled by installing a set of plugins on a common software base. In this architecture, the software base embeds a lightweight workflow engine that guides the main flow of control and data of the application. This architecture eliminates the problem of scattered flow of data and control and facilitates plugin substitution. This architecture is currently being used to build a biomedical engineering research application on top of the Eclipse platform.

1 Introduction

In recent years, computer users have witnessed the emergence of a wave of successful applications whose functionalities can be extended via the addition of *plugins*. Plugins are binary extension units for an application whose architecture allows functionalities to be introduced by end-users at well-defined places once the application has been installed. Plugins are software entities that are closely related to components. Component-based development, however, does not usually consider that components can be added to applications after the applications have been installed. Components are rather used to facilitate the construction of the applications themselves (extensible or not) [9].

Building an application as a plugin-based system makes sense both from a technical and an economical point of view. Technically it makes sense because the approach promotes a high level of modularity and decoupling between the *base* (or main) application and the plugins which can be developed independently, helping reduce delivery periods. Economically, the approach also makes sense, since a developer can concentrate on the development of the base application or the development of its extensions. Furthermore, since development lifecycles are independent, plugins can be deployed separately from the base application. Plugin deployment activities, which are usually performed by end users, include install, update and removal of the plugins.

A field where the plugin-based approach can be particularly useful is the construction of software product lines. Software product lines represent sets of applications (typically from a common application domain) that share features and are developed by reusing certain elements, such as an architectural foundation [1]. If this architectural foundation is built following a plugin-based approach, the construction of the different applications can be achieved by installing different sets of plugins on

I. Gorton et al. (Eds.): CBSE 2006, LNCS 4063, pp. 198 – 205, 2006.

top of the foundation. One difficulty with this approach resides, however, in the fact that applications must typically execute workflows associated with their particular domain. In the case where an application is built as a set of plugins installed on a common foundation, the control and data flow of the application is usually scattered among the foundation and the set of plugins that compose the application. This situation limits the possibility of plugin update or substitution, since it is difficult for replacement plugins to guarantee that they implement the correct part in the control and data flow.

This paper proposes a solution to this problem that introduces concepts from workflow-based applications into a plugin-based architecture. Workflow-based applications support the definition and execution of business processes. In such applications, the definition and execution of the appropriate control and data flow, and the invocation of the application logic blocks are externalized. As a consequence, changes to the process can be done without impacting the application logic blocks which become independent from the main data and control flow and as a consequence can be replaced more easily [5]. The work presented in this paper is currently being applied in the construction of a biomedical engineering research application on top of the Eclipse platform.

The remainder of the paper is structured as follows: section 2 discusses plugin-based application development using Eclipse and discusses the scattered flow of control problem, section 3 describes the proposed architecture, section 4 presents current and future work respectively, section 5 presents related work and finally, section 6 concludes the paper.

2 Plugin-Based Application Development

This section discusses plugin-based application development and the problem of scattered flow of control. Due to space restrictions, discussion around plugins focuses mainly on the Eclipse platform, which is used to implement the ideas described in this paper.

2.1 Plugin-Based Applications and Eclipse

Today, a wide variety of plugin-based applications exist; these applications include web browsers, image editing tools and integrated development environments (IDEs). All of these tools are characterized by the fact that they are built as standalone applications that provide an initial degree of functionality when installed. This functionality is available even when no plugins are present; for example, when a web browser is installed, it allows pages to be read but multimedia content cannot usually be displayed.

Among plugin-based applications, the Eclipse platform [4] has some particular characteristics. This platform was originally conceived as a foundation to facilitate the construction of IDEs. As such it provided, in addition to standard elements such as a text editor, development facilities that included team development support. A specific development environment could be built by adding a set of tools, for example a compiler for a particular language, to the base platform. The different tools were

delivered as plugins. Today, the Eclipse platform has evolved from being an IDE-oriented foundation to become a generic plugin-based application foundation. Development specific elements have been moved out of the base platform which is now called the Rich Client Platform [6]. The Rich Client Platform (RCP) provides the minimal functionality required to allow all-purpose plugin-based client-side applications to be built on top of it.

Eclipse's plugin model allows plugins to interact directly with other plugins and not only with the base application. As a result, plugins can be composed in a similar way to components. Eclipse's plugins provide *extensions* when they contain new functionalities to be introduced into the base application or into another plugin. Plugins can also declare *extension points* which are locations where other plugins can introduce their own extensions and enrich the original plugin's functionality. Furthermore, extension points are always optional, meaning that any provider of an extension point must be capable of functioning even if no provider of extensions for that extension point are present. Finally, the vision behind the Eclipse's architecture is that everything is built out of plugins, including the base RCP itself, which currently consists of around 11 different plugins whose presence is mandatory. The plugins that form the RCP manage plugin deployment activities and declare a series of extension points which allow other plugins to introduce new functionalities such as toolbars, menu entries and views. Today, the RCP is gaining much momentum as the Eclipse IDE itself facilitates enormously the task of constructing plugin-based applications on top of the RCP through its Plugin Development Environment (PDE).

2.2 The Structure of an Application Developed on Top of Eclipse

In a certain way the Eclipse's plugin approach is a hybrid between "traditional" plugin approaches and component-based development. In Eclipse, a customized (or extended) base platform is assembled by selecting a set of plugins that will provide functionalities that are added to the basic ones already provided by the RCP. Once assembled, this extended base platform is delivered to end users as a standalone application. The plugins that form this application may not be removed afterwards. However, after this application is installed, end users can continue extending the functionality of the application by installing additional plugins into it. The architecture of a typical application built on top of the RCP is depicted in figure 1. This figure also shows that plugins can interact directly among each other without extending the base platform.

2.3 The Problem of Scattered Flow of Control

Eclipse plugins typically provide user interface elements that allow the user to interact with the application. These user interface elements, such as buttons on a toolbar, for example, are associated with handlers that can invoke methods declared on an interface provided by a different plugin's extension. In such a situation the logic that guides the flow of control and data of the application ends up being scattered among the plugin set and the base platform. This situation can limit the substitutability of the plugins used to build the application. For instance, when a plugin is substituted by another plugin, either by a newer version or by a different plugin that provides the

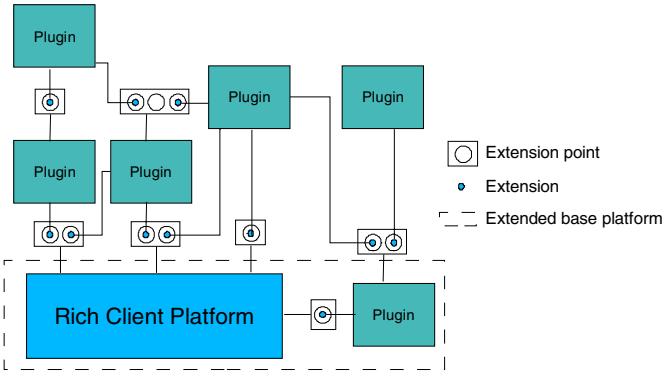

Fig. 1. Architecture of an application built on top of the Eclipse Rich Client Platform

same extension, the replacing plugin must correctly implement its predecessor's part of the flow of control and data that is necessary for the application to function correctly.

In today's applications these problems are addressed by limiting substitutability and by constraining the places where plugins can extend the base application. This solution is, however, undesirable for the context of plugin-based product line architectures, where it is necessary to allow a large number of plugins to be substituted by completely different sets.

3 A Workflow-Based Product Line Architecture

This section introduces the concepts workflow-based applications and discusses how these concepts are used to solve the problem exposed in the previous section.

3.1 Workflow-Based Applications

Workflow-based applications support the definition and execution of business processes (composed of activities associated to a particular domain). In such applications, the definition and execution of the appropriate control and data flow, and the invocation of the application logic blocks are externalized. This allows the workflow to be changed without impacting the application logic blocks [5]. In workflow-based applications, logic blocks become flow-independent in the sense that they do not contain application logic associated with the execution of the business process. This facilitates the replacement of logic blocks.

3.2 Using Workflows in a Plugin-Based Product Line Architecture

To limit the problem of scattered flow of data and control and to facilitate plugin substitutability and reusability, the concepts of workflow-based applications can be introduced in a plugin-based product line architecture. The fundamental idea is to limit the degree of direct interaction among plugins and to introduce a third party (a mediator) responsible for controlling the application's main flow of control and data.

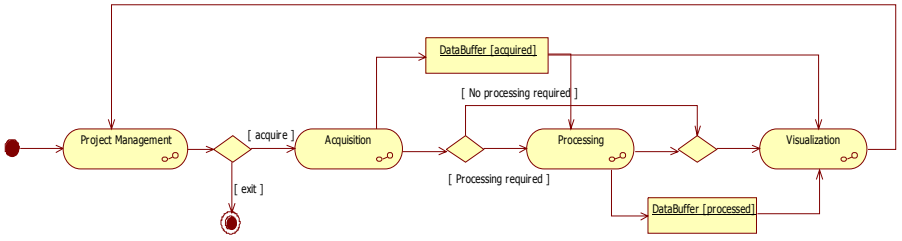

Fig. 2. Typical workflow found in DSP applications for biomedical research

In the context of this work, this idea has been used to create a plugin-based product line architecture oriented towards the construction of digital signal processing (DSP) applications for biomedical engineering research. In this field, these applications are generally guided by the workflow depicted in figure 2. This main workflow contains four different activities: project management, data acquisition, processing and visualization. The activity diagram shows that data that is acquired can be visualized immediately or be processed by applying different algorithms before visualization. Each of the activities of the main workflow is itself guided by a specific sub-workflow. Figure 3 shows the sub-workflow associated to the acquisition activity in the main workflow. This sub-workflow allows users to test the acquisition device before performing the actual data acquisition and storing the corresponding data.

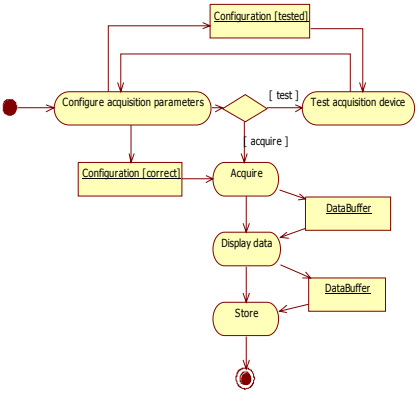

Fig. 3. Detail of the sub-workflow associated to the acquisition activity

To achieve a workflow based approach in plugin-based product line architecture, the different workflow definitions are contained and executed by the product-line architecture. Figure 4 illustrates how this is achieved: the product-line architecture, which is the common element to specific applications, is constructed as an extended Eclipse base platform that contains a lightweight workflow engine. Plugins are associated with particular activities, such as data acquisition or visualization. Every plugin's extension interface provides methods that fall into three different categories. The first category of methods allow the workflow engine to control the execution of

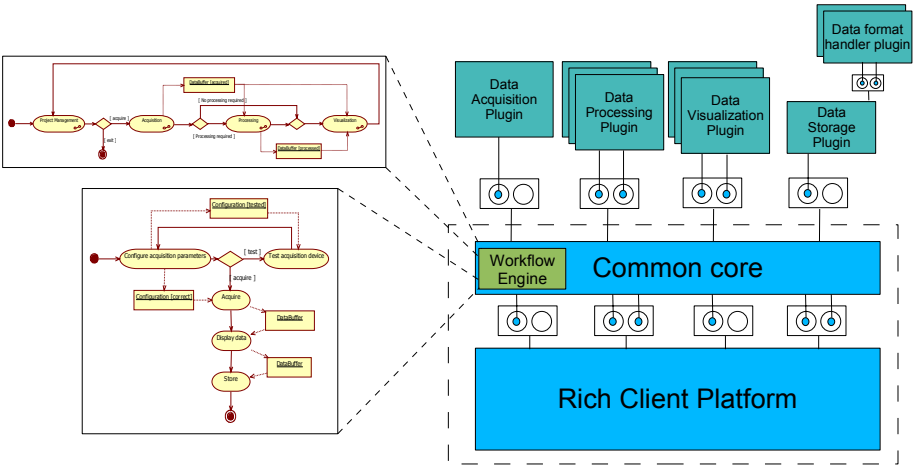

Fig. 4. Architecture with the lightweight workflow engine

an activity. The second category allows the workflow engine to register itself as an observer of activity events. The events produced by the plugins include activity termination and cancellation. The third category of methods allow the workflow engine to perform data transfer. As a result, workflows are executed as the engine initiates activities and receives notifications from the plugins.

Finally, even tough the approach described here requires that plugins associated with different activities to not interact directly, it does not limit plugins associated to a particular activity from providing extension points that allow other plugins to extend their functionality. An example of this occurs in the plugin associated with the data storage activity in figure 4, this plugin can be extended by specialized plugins that allow the data to be stored in different formats. The fact that the data storage plugin is extended by other plugins is, however, irrelevant to the application's workflow.

4 Current Results and Future Work

The work presented in this paper is part of an ongoing project that is realized at the Universidad Autónoma Metropolitana Iztapalapa in Mexico City. This project received an Eclipse innovation grant from IBM in 2005. The goal of this project is to build an application to acquire data obtained from respiratory sounds. This application will be deployed in a hospital environment where it will allow both physicians to better diagnose certain respiratory conditions and biomedical researchers to gather data on the field. This project is realized following the Unified Process development-methodology and is currently in the construction phase. A screen capture of an executable prototype for the project is depicted in figure 5. This prototype is built on top of the base architecture described in the previous section.

Future work includes the construction of a different but related application on top of the same product-line architecture with a set of different plugins. Other areas of interest include the use of a language to describe the workflows (currently this is done

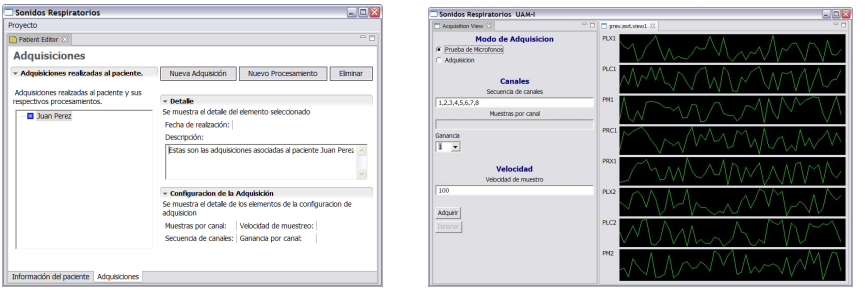

Fig. 5. Screenshots of the application

programmatically). Furthermore, workflow description could itself be delivered via plugins to facilitate extension. A final area of interest to the authors is the possibility of supporting dynamic deployment activities in the application, this would include the install, update and removal of the plugins during execution.

5 Related Work

The work presented in this paper is related to three main research areas which include component-based software development, product line architectures and workflow-based applications.

5.1 Component-Based Software Development

As mentioned before, components and plugins have similar features. Examples of existing component models are JavaBeans [8] and the CORBA Component Model [7]. Such models are successfully used today in the construction of applications both on the client and on the server side. However, component models are not really oriented towards supporting plugin-oriented features such as end-user managed evolution or the interaction with an extensible application core. It is possible to implement a plugin-based architecture over a standard component model, but this is not straightforward.

5.2 Software Product Line Architectures

Among the core assets of a software product line, the software architecture plays the most central role [1]. One of the most successful plugin-based product line architectures is the original Eclipse platform, which allowed different IDEs to be built. A framework that supports the construction of plugin-based product line architectures is described in [2]. The approach proposed, however, does not deal with the issue of scattered flow of control.

5.3 Workflow-Based Applications

The concepts of workflow-based applications are particularly popular in the area of service oriented architectures (SOA). The Business Process Execution Language

(BPEL) supports the execution of processes that interact with web services [3]. There are some similarities between SOA and plugin-based architectures: the plugin-based application core is similar to a service requester that searches the plugin registry to discover available services (provided by registered plugins). Once the core finds them, it interacts with these service providers. In service orientation, there are no guarantees that a service requester may find a particular service; this is the same for a plugin-based application since there is no guarantee that a particular plugin will be present. There are, however, some differences with SOA, since in SOA, service providers and requesters may arrive or depart constantly, and such a high level of dynamism is not currently present in plugin-based architectures.

6 Conclusions

This paper has presented a plugin-based software product line architecture where the architecture embodies a lightweight workflow engine to facilitate plugin substitutability and specific product development. This architecture is currently being applied in the construction of a biomedical engineering research tool. It must be noted that the ideas presented in this paper are not specific to plugin-based applications; they can also be beneficial to the construction of standard component-based applications.

Acknowledgments. The authors wish to acknowledge IBM for its financial support, and the students who are participating in the project and who make it possible.

References

1. Bass, L. and Clements, P. and Kazman, R., "*Software Architecture in Practice (2d Edition),*" Addison Wesley, 2003
2. Caporuscio, M. and Muccini, H. and Pelliccione, P. and Di Nisio, E. "*Towards a Plugin-based Implementation of Product Line Architectures,*" unpublished paper found online at http://se2c.uni.lu/tiki/se2c-bib_abstract.php?id=1948. Visited 01/06
3. Curbera, F. and Al., "Business Process Execution Language (BPEL) for Web Services, Version 1.1," Online document at http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, last visited 01/06
4. The Eclipse Foundation, Official Eclipse Homepage at http://www.eclipse.org, last visited 01/2006.
5. Leymann, F. and Roller, D., "*Workflow-based applications,*" IBM Systems Journal, Vol. 36, No. 1, (pp. 102), 1997.
6. McAffer, J. and Lemieux, J-M., "*Eclipse Rich Client Platform*," Eclipse Series, Addison Wesley, 2006
7. Object Management Group (OMG), "*CORBA Component Model, Version 3.0,*" Online document available at http://www.omg.org/technology/documents/formal/components.htm, June 2002
8. Sun Microsystems, "*Java Beans Specification, Version 1.01*", Available online at http://java.sun.com/products/javabeans/, July 1997.
9. Clemens Szyperski, "*Component Software: beyond object-oriented programming,*" 2d Edition, Addison-Wesley Professional, 2002.