Plug-in Architecture and Design Guidelines for Customizable Enterprise Applications

Reinhard Wolfinger

Christian Doppler Laboratory
for Automated Software Engineering
University Linz, Austria
wolfinger@ase.jku.at

Categories and Subject Descriptors D.2.11 [**Software Engineering**]: Software Architectures – Patterns.

General Terms Design

Keywords Run-time adaptation; Plug-in architecture

1. Introduction

Today's enterprise software is often designed to have a component-based architecture and built with object-oriented application frameworks. Decomposition typically produces rather coarse-grained business logic and presentation components where features are usually deployed as a monolithic piece of software. Eventually all customers get the same application, while configuration or license codes determine whether particular features are enabled or not. This one-size-fits-all approach causes three major problems:

(a) Enterprise software is inherently complex and feature-rich, while individual users only need a small fraction of the features. Hence, if the user interface of an application is cramped with features for all business processes, users struggle to find features they really need for their tasks.

(b) Customer requirements for enterprise applications are characterized by a large variation and depend on industry or company size. It is impossible for an enterprise application of any size to fully meet customer requirements with a standard product. Even if an application covers all the major business-relevant scenarios, customers typically ask for more features addressing their special needs. Some of these features are highly industry-specific and thus outside the manufacturer's core competence. Thus the manufacturer may decide not to include them in the product. Still, customers need a way to add features that are important to them.

(c) Enterprise customers tend to be conservative about deploying patches. Often one business unit urges to deploy a certain patch, while another business unit is reluctant to do so yet. The coarse-grained deployment model assumes that patching means to replace large parts of the application. This rules out selective patch deployment scenarios.

From these problems we derive the following main goals of our research:

(a) Customizability and run-time adaptability. Complex enterprise applications should be made customizable to the needs of individual users by breaking it up into a thin core application that can be extended with features that are plugged into the core and integrate seamlessly with it. Using run-time system adaptation mechanisms the application cannot only be customized to the needs of individual users, but can also be reconfigured for a specific working context without restarting it.

(b) Extensibility through third parties and end users. End users and third parties should be able to safely contribute any functionality the manufacturer did not already provide in the base product. Integrating this functionality should be made as simple as dropping an executable into the application directory. The ultimate goal is a slim system with a potentially unlimited, but controlled extensibility.

2. State of the Art

Although component technologies such as COM, CORBA, .NET Assemblies or JavaBeans simplify software reuse and independent component deployment, they rely on programmatic effort for assembling components. Current architectures often require component assembly to happen at development time. This cannot adequately support enterprise applications that increasingly need to be extensible by third parties and reconfigurable at run time.

Recently, the concept of plug-in components has emerged as a promising way of building applications that are inherently extensible and customizable to the needs of specific users. Several plug-in approaches have already found their way into today's software development practice. For instance, OSGi defines a standard for deploying and managing coarse-grained components. OSGi defines several mechanisms that are relevant for plug-in frameworks such as lifecycle management of components, service locators, or hot updates. The configuration of a resulting application depends on the set of deployed components, which may vary per user and can even change at run time. Eclipse can be regarded as the most outstanding representative of plug-in systems today. Eclipse plug-ins are fine-grained components with a well-defined and published interface that can plug into so-called extension points of other components.

3. Research Issues

Despite the success of plug-in systems so far, many open research issues remain as current plug-in systems still suffer from several deficiencies:

(1) Run-time adaptation of systems, where users can decide which features should be made available, is not fully auto-

OOPSLA'08 October 19–23, 2008, Nashville, Tennessee, USA.

mated in current systems. *How can we support end users at run time to adapt an application to changing working scenarios?*

(2) In Eclipse as well as in OSGi, the integration of plug-ins may still require a significant programming effort. *How can we automate the integration of plug-ins?*

(3) Application design in general is a challenging task, but plug-in-based applications and run-time adaptation bring up additional design issues. *How can we provide guidance for software architects on designing customizable and extensible plug-in applications?*

(4) Contract specification is an issue which we consider to be rather immature. In Eclipse, for example, component contracts are described as extension points and extensions in dedicated XML files, an approach that we found to be rather tedious. *How can we make contract specification more effective and easier to maintain?*

Moreover, OSGi targets embedded systems and Eclipse targets tool environments. No plug-in approach is known which specifically targets enterprise applications. However, when applying the plug-in approach in the enterprise domain, new challenges and requirements emerge:

(5) In the enterprise domain customers expect high availability of systems. Integrating plug-ins that have been contributed by unknown third parties can represent an unpredictable risk for the stability of the system. Current plug-in systems do not offer a solution for this problem. *How can extensible applications take precautions against being taken down by a flawed or malicious extension?*

(6) An open plug-in system allows third parties to contribute functionality. The creator of a host application might want to restrict what a plug-in can do and what it cannot do. Current plug-in systems cannot restrict composition aspects, as would be required, for example, to control which contributor can contribute to which parts of the system. *How can extensible applications constrain the permissions of plug-ins?*

4. Research Approach

Our approach is to adopt the concept of plug-in component architectures in the domain of enterprise software in order to improve extensibility and customizability there. We will come up with a new architecture, new design approaches and accompanying design guidelines.

In a first project phase we have conducted an comprehensive review of the research literature as well as current industry standards for flexible software architecture. Our research issues address several weaknesses in state-of-the-art architecture. Furthermore we have analyzed the enterprise software of our industry partner BMD Systemhaus GmbH and have identified additional requirements for plug-in based appliations in the enterprise domain [2]. Together with our industrial partner we have developed several usage scenarios for run-time adaptation in the enterprise domain confirming the need of such an approach [3].

As a follow-up step we have been designing a plug-in platform featuring an ultra-thin core layer as well as a slot and extension mechanism that enables developers to build highly flexible applications. Our design and implementation address the research issues (1), (2), (4), and (5) from Section 3. The plug-in platform is based on the metaphor of slots and extensions for specifying interaction and integration of components. A slot specifies a contract for extending a piece of software (called the slot host). An extension is a plug-in component that fills a slot [1]. The core layer acts as a run-time environment that automatically composes applications from a set of available plug-ins. In contrast to other platforms, an application can be dynamically reconfigured by addition or removal of plug-ins without restarting the application [3]. Our platform allows developers to restrict the permissions of plug-ins, which attach to a specific slot, or to protect the host application against crashes of buggy or malicious plug-ins taking specific precautions for isolating the plug-in from the rest of the application [2].

In 2008 and 2009 we plan to address the remaining research issues (3) and (6) from Section 3 and to validate our approach:

Guidelines and Patterns. We currently conduct a case study where we reengineer our industry partner's Customer Relationship Management (CRM) application to fit our plug-in platform. During the case study we identified design issues specific for plug-in-based applications and run-time adaptation. For instance, software architects need to find the right level of granularity for plug-ins, to design stable slot interfaces before opening them to the public, or to consider that the application's user interface will be adaptable at run time. On those and other issues we want to provide guidance. We plan to identify recurring patterns and best practices that we plan to collect, categorize and document, e.g.:

- *Classification*. A plug-in based application is assembled from different types of plug-ins. Some of our design patterns will apply to all types, while others are specific to certain plug-in types. In order to organize our design patterns, we first categorize plug-ins.
- *Patterns.* In software engineering design patterns offer best practice solutions to common design problems. They establish a common vocabulary and ease communication between developers. In a similar way we want to make our design knowledge reusable for plug-in developers.
- *Guidelines*. We will provide guidance on how to proceed in application design, i.e., how to apply our patterns in plug-in design, and how to assemble the application by integrating plug-ins using our slot and extension mechanism.

Security constraints. We plan to refine the security mechanisms in our platform for compositional purposes. This will allow a slot host to control who is allowed to contribute based on the provider's identity.

Evaluation. We plan to evaluate the plug-in platform and design guidelines in a comparison study within the context of students from a software engineering course. Additionally we will continue the case study with our partner's CRM application in order to evaluate run-time adapation scenarios. We plan to present our evaluation strategy at the symposium's workshop and would appreciate feedback on how to best evaluate our plug-in platform and design guidelines.

References

[1] Wolfinger, R., Dhungana, D., Prähofer, H., and Mössenböck, H.: A Component Plug-in Architecture for the .NET Platform. 7th Joint Modular Languages Conference (JMLC 2006), Oxford, UK, September 13-15, 2006.

[2] Wolfinger, R., and Prähofer, H.: Integration Models in a .NET Plug-in Framework. SE 2007 - the Conference on Software Engineering, Hamburg, Germany, March 27-30, 2007.

[3] Wolfinger, R., Reiter, S., Dhungana, D., Grünbacher, P., and Prähofer, H.: Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques. 7th International Conference on Composition-Based Software Systems (ICCBSS 2008), Madrid, Spain, February 25-29, 2008 (Best Paper Award).