Dynamic Updating of Object-Oriented Software Systems Based on Aspects

Author(s):
Previtali-Cech, Susanne

Publication Date:
2009

Permanent Link:

https://doi.org/10.3929/ethz-a-005938293 →

DISS. ETH NO. 18361

DYNAMIC UPDATING OF OBJECT-ORIENTED SOFTWARE SYSTEMS BASED ON ASPECTS

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Technical Sciences

presented by
SUSANNE PREVITALI-CECH
Dipl.-Ing., University Klagenfurt, Austria
born February 14, 1976
citizen of Austria

accepted on the recommendation of
Prof. Dr. Thomas R. Gross, examiner
Dr. Gavin Bierman, co-examiner
Prof. Dr. Mira Mezini, co-examiner

2009

Abstract

In this thesis, we propose a system that allows object-oriented applications to be updated at run-time. There are a number of domains where downtime of an application due to software maintenance is unacceptable. For server applications, for instance, it is essential that they are continuously responsive and also up-to-date, and thus, any software updates must be applied while the application is running. The dynamic updating system we propose leverages aspect technology to express a software update as a modular unit and thus allows introducing the update in an incremental fashion.

We consider three characteristics of aspect-oriented programming as appealing with regard to implementing a software updating system: An aspect system provides a mechanism to specify crosscutting behavior, a machinery to activate that behavior (i.e., either at compile-time or run-time), and the guarantee that this behavior is eventually (i.e., whenever the associated join point is reached) executed. Software updates follow a similar pattern: The update can be expressed as some changed or additional behavior that must be integrated into the existing application, whereby the actual integration can either happen at compile-time or run-time, but must take effect once the update has been applied.

Although many updates remain encapsulated within a single method body a class, the nature of software updates is clearly crosscutting. Classes depend on each other and if the specification of one class changes, all the clients of the class are consequently affected. Using the client/supplier relationship we can capture class dependences and understand the crosscutting nature of software changes.

The update model we introduce is based on the client/supplier relationship. Employing the client/supplier relationship as a model, we can identify the updates that are logically related and encapsulate these updates in a modular unit, the aspect. We have developed an updating system for Java programs that implements the update model and automatically analyzes two program versions and extracts the necessary updates. In addition, we have conducted an empirical study on eight real-world Java applications whose versions encompass several years of development. The empirical study validates the update model and the updating system. The study shows that for some versions, the necessary updates can be extracted automatically by the updating system. Many other version updates require only minimal modifications to the new program version to enable automatic update extraction.

Zusammenfassung

Diese Dissertation stellt ein System vor, das objektorientierter Software zur Laufzeit aktualisiert. In verschiedenen Bereichen ist eine Unterbrechung der Softwareausführung zur Wartung nicht tolerierbar. Da beispielsweise Serverapplikationen sowohl kontinuierlich verfügbar als auch auf dem letzten Stand sein müssen, muss die Software während der Ausführung aktualisiert werden. Das vorgeschlagene dynamische Aktualisierungssystem benutzt Aspekttechnologie, um ein Software-Update zu modularisieren und inkrementell zu verwenden.

Drei Merkmale der Aspektorientierung sind für die Implementierung eines Software-Aktualisierungssystems attraktiv: Ein Aspektsystem stellt einen Mechanismus zur Verfügung, um Klassen übergreifendens Verhalten zu beschreiben, eine Maschinerie, um dieses Verhalten zu aktivieren (zur Kompilier- oder Laufzeit), und die Garantie, dass dieses Verhalten, wenn ein entsprechender Programmpunkt erreicht wird, ausgeführt wird. Software-Updates folgen einem ähnlichen Muster: Ein Update kann als verändertes oder zusätzliches Verhalten ausgedrückt werden, das in eine existierende Applikation integriert werden muss. Die Integration des Updates kann zur Kompilier- oder Laufzeit stattfinden, muss jedoch wirksam werden, nachdem es eingebracht worden ist.

Obwohl viele Änderungen nur einzelne Methoden betreffen, sind Updates in der Regel Klassen übergreifend (“crosscutting”). Klassen hängen voneinander ab und wenn die Spezifikation einer Klasse ändert, sind alle Benutzer (die Klienten) der Klasse betroffen. Wir benutzen diese Beziehung zwischen Klienten- und Anbieterklasse, um Abhängigkeiten zu erkennen und die Klassen übergreifende Eigenschaft von Softwareänderungen zu verstehen.

Wie stellen ein Aktualisierungsmodell vor, das auf der Beziehung zwischen Klienten und Anbieter basiert. Mit diesem Modell können wir die logisch zusammenhängenden Updates identifizieren und diese in ein Modul, den Aspekt, kapseln. Wir haben ein Aktualisierungssystem für Java-Programme entwickelt, das dieses Modell implementiert und das automatisch zwei Programmversionen analysiert und die notwendigen Updates berechnet. Zusätzlich haben wir eine empirische Studie durchgeführt, die das Aktualisierungsmodell und das Aktualisierungssystem validiert. Die Studie analysiert acht Java-Programme, deren Versionen sich über einige Entwicklungsjahre strecken. Die Studie zeigt, dass Updates für einige Versionen voll automatisch und für die meisten Versionen nach geringen Anpassungen des Programmtextes berechnet werden können.