Komponierung, Konfiguration und Adaptierung von heterogenen Software-Komponenten

Uwe Zdun
Abteilung für Wirtschaftsinformatik
Wien, Österrreich
zdun@acm.org

Zusammenfassung

In diesem Papier erläutern wir einige Probleme der Komponierung, Konfiguration und Adaptierung von heterogenen Software-Komponenten. Einige bekannte Pattern können verwendet werden, um diese Problem zu lösen. Aus diesen Lösungen entstand die Idee, die Pattern als systematisches Konzept für den Sprachentwurf zu verwenden – welches dann in der Sprache Frag umgesetzt wurde.

1 Einleitung

Komponenten sollen wiederverwendbare, in sich geschlossene Bausteine eines Software-Systems sein. Verschiedene Komponentenansätze haben auch einen starken Fokus auf Aspekten der Komponierung, Konfiguration und Adaptierung von Software-Komponenten, aber diese Aspekte sind noch nicht wohldefiniert. In diesem Papier soll eine Sprache vorgestellt werden, die sich in ihrem Sprachentwurf auf diese Aspekte konzentriert und somit einen einheitlichen Ansatz der Komponierung, Konfiguration und Adaptierung von Software-Komponenten darstellt.

Es gibt viele Komponentenbegriffe. Die Definition von Szyperski [5] soll hier als ein Startpunkt dienen: "*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*" Diese Definition umfasst Server-Komponenten-Ansätze (wie EJB, CCM und COM+), Java Beans, Komponenten-Frameworks in Skriptsprachen (wie Tcl, Python, Perl und Visual Basic), Active X controls, C Bibliotheken mit klar abgegrenzten Schnittstellen und viele andere Arten von Komponenten. Die Definition kann somit recht breit interpretiert werden, was insbesondere aus einer (praktischen) Reengineering-Perspektive wichtig ist, denn in der Praxis sind Komponenten oft nicht idealtypische Komponenten, sondern sind groß, haben keine erzwungenen Komponentengrenzen und haben keine definierten Schnittstellen.

Alle genannten Komponentenmodelle verfügen nichtsdestotrotz über Ansätze oder Best Practices zur Komponierung, Konfiguration und Adaptierung von Software-Komponenten. Hier stellen sich – insbesondere aus einer Maintenance und Reengineering Perspektive – eine Reihe von offenen Problemen:

- Viele Komponentenansätze sind nur für eine Sprache oder Plattform entworfen worden und nicht für heterogene Situationen (wie in vielen Alt-Systemen). Insbesondere sind Ansätze zur tieferen Sprachintegration, die bspw. Destruction Order, Inheritance Hierarchies, Garbage Collection und ähnliche Spracheigenschaften automatisiert beachten, kaum umgesetzt.

- Einige Komponentenansätze erlauben nicht das dynamische Deployen und Un-Deployen von Komponenten. Ferner fehlen gerade in Alt-Systemen oft Komponenten-Deployment-Abstraktionen, wie Package Konzepte. Diese nachträglich einzubringen bedeutet einen erheblichen Aufwand.

- Komponenten, insbesondere von Dritten, müssen oft angepasst werden, bspw. bei Schnittstellenänderungen und Versionsänderungen. Viele Komponentenansätze unterstützten die Adaptierung von Komponenten gar nicht oder nicht zur Laufzeit.

- Um auf Komponenten zugreifen zu können, ist es notwendig ihre Schnittstelle zu kennen; zum Teil werden diese zur Laufzeit benötigt. Es ist auch oft hilfreich, wenn man diese Schnittstellen erzwingen kann, wie im Facade Pattern [2].

- Um Re-Kompilierungen zu vermeiden, sollten Komponenten nicht nur durch Übergabeparameter konfiguriert werden, sondern es sollte auch weitere deklarative und verhaltensbeschreibende Konfigurationsmöglichkeiten geben.

Für jedes dieser Probleme existieren geeignete Lösungen. Eine Vielzahl dieser Lösungen ist bereits in Form von Software-Patterns dokumentiert. Eine EuroPLoP 2003 Focus Group hat die wichtigsten Pattern in diesem Bereich untersucht und kategorisiert [10]. Die Idee hinter der Sprache Frag ist, dem Entwickler eine Sprache an die Hand zu geben, welche die Lösungen in diesen Pattern in einer wiederverwendbaren Art und Weise als Sprachelemente implementiert. Die Sprache kann ferner in einer einfachen Art und Weise mit existierenden Umgebungen (bspw. anderen Sprachen, Plattformen, Komponentenansätzen, Rahmenwerken, Alt-Systemen, etc.) kombiniert werden.

2 Sprachunterstütze Patterns für Komponierung, Konfiguration und Adaptierung von Software-Komponenten in Frag

Frag ist eine objekt-orientierte Erweiterung der Sprache Tcl, die speziell für Einbettung in andere Systeme und deren Integration entworfen wurde. In diesem Abschnitt soll erklärt werden, wie Frag Patterns für die Komponierung, Konfiguration und Adaptierung von Software-Komponenten unterstützt und wie die obengenannten Probleme aufgelöst werden. Für eine detaillierte Sprachbeschreibung sei auf `frag.sourceforge.net` verwiesen.

Sprachintegration Viele Komponentenansätze sind nur für eine Sprache ausgelegt. Frag implementiert das Split Object Pattern [9] und ein extrem flexibles Objekt-Konzept. Dadurch kann Frag automatisiert an andere Sprachen, wie C, C++, Java und Tcl, angepasst und mit ihnen weitgehend integriert werden. Wie in [9] beschrieben, umfasst diese Integration beispielsweise Objekt-Identitäten, Klassenhierarchien und andere Objekt-Beziehungen. Ein wichtiger Aspekt für die Wartung und Konfiguration von Systemen ist: Frag kann vollständig ist diese anderen Host-Sprachen als Glue-Sprache eingebettet werden. Insbesondere läuft Frag auch in einer Java Virtual Machine über den Jacl [1] Interpreter.

Deployment von Komponenten Einige Server Component Patterns [6] beschreiben Aspekte des Deployments, insbes. Component Installation, Component Package und Assembly Package. Diese Pattern werden durch den Tcl `package` Mechanismus unterstützt. Somit steht eine geeignete Deployment-Abstraktion für Host-Sprache und Glue-Sprache Komponenten zur Verfügung. Das Pattern Component Configurator [4] erlaubt zur Laufzeit Komponenten zu deployen und wieder aus dem System zu entfernen. Frag unterstützt dieses Pattern durch ein dynamisches Objekt-System und Introspection Options [7]. Dies bedeutet, dass Komponenten dynamisch zur Laufzeit als Packages geladen werden können und die Konfiguration dieser Komponenten nicht vor der Laufzeit bekannt sein muß.

Komponenten-Adaptierung Frag unterstützt die Pattern Interceptor [4] und Message Interceptor [7] durch ein Mixin-Konzept in der Sprache. Jede Komponente lässt sich dynamisch mit Vorher- und Nachher-Verhalten erweitern. Mit Interzeptoren kann man flexibel auf Schnittstellenänderungen, Versionsänderungen, und ähnliche Situationen reagieren. Frag kann als Indirection Layer [7] für Komponenten genutzt werden. D.h. Nachrichten im System können abfangen und manipuliert werden. Beispielsweise zeigen wir in [9], wie man Komponentenaufrufe für ein dynamisches Feature-Tracing umleiten kann. Diese Sprachmittel können auch zur Adaptierung beim dynamischen Komponenten-Deployment eingesetzt werden.

Schnittstellenbeschreibungen Frag unterstützt die Pattern Component Wrapper [3] und Wrapper Facade [4], welche das Einbinden von externen Komponenten und Libraries beschreiben. Über Component Wrappers und Schnittstellen-Klassen (wie Facades [2]) kann das Pattern Explicit Export/Import [3] unterstützt werden. Durch das Pattern Introspection Options [7], das in Frag für jedes Sprachelement umgesetzt ist, können diese Information zur Laufzeit abgefragt werden.

Komponentenkonfiguration Frag ist als Tcl-Erweiterung automatisch eine Command Language [8]. Command Languages werden oft für dynamische Konfigurierungen mit Verhalten benutzt. D.h. Frag ist entworfen, um dynamisch neues Verhalten in Komponenten einzubringen. Überdies können in Frag aber auch andere Metadaten als Annotations [6] und Metadata Tags [7] verwendet werden, wie bspw. XML-Formate. Diese dienen hauptsächlich der Konfiguration der Komponenten mit deklarativen Parametern und der Angabe von Metadaten über die Komponenten zur Selbstdokumentation.

3 Zusammenfassung

In diesem Papier haben wir den Pattern-basierten Entwurf der Sprache Frag beschrieben. Es wurden nicht alle der Pattern beschrieben, die in Frag zur Komponierung, Konfiguration und Adaptierung von Software-Komponenten eingesetzt werden, sondern nur die wesentlichen. Aber man sieht, dass auf diese Weise erhebliche Probleme in diesem Bereich in einer systematischen Art und Weise angegangen und sprachlich gelöst werden können. Anwendungsfälle der Sprache werden bspw. in [9] beschrieben.

Literatur

[1] M. DeJong and S. Redman. Tcl Java Integration. http://www.tcl.tk/software/java/, 2003.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[3] M. Goedicke and U. Zdun. Piecemeal legacy migrating with an architectural pattern language: A case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 14(1):1–30, 2002.

[4] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Patterns for Concurrent and Distributed Objects*. Pattern-Oriented Software Architecture. J. Wiley and Sons Ltd., 2000.

[5] C. Szyperski. *Component Software – Beyond Object-Oriented Programming*. ACM Press Books. Addison-Wesley, 1997.

[6] M. Völter, A. Schmid, and E. Wolff. *Server Component Patterns – Component Infrastructures illustrated with EJB*. J. Wiley and Sons Ltd., 2002.

[7] U. Zdun. Patterns of tracing software structures and dependencies. In *Proceedings of EuroPlop 2003*, Irsee, Germany, June 2003.

[8] U. Zdun. Some patterns of language and component integration. draft; submitted to EuroPLoP 2004, 2004.

[9] U. Zdun. Using split objects for maintenance and reengineering tasks. In *8th European Conference on Software Maintenance and Reengineering (CSMR'01)*, Tampere, Finland, Mar 2004.

[10] U. Zdun and M. Voelter. Europlop 2003 focus group on component composition. In *Proceedings of EuroPlop 2003*, Irsee, Germany, June 2003.