Referenz-Architektur und nichtfunktionale Anforderungen adaptiver Dialogkerne

Jürgen Rückert, Barbara Paech

Fakultät für Mathematik und Informatik
Im Neuenheimer Feld 326
69120 Heidelberg
{rueckert, paech}@informatik.uni-heidelberg.de

Abstract: Die Service-oriented Architecture (SOA) ermöglicht die Flexibilisierung von Anwendungen durch Einbindung von Diensten zur Laufzeit und durch Verwendung einer Vielzahl von Technologien zur Kommunikation mit den eingebundenen Diensten. Spätestens in Benutzungsschnittstellen (BSS) müssen EntwicklerInnen die fachliche und technologische Vielfalt von Diensten adressieren, da sich die BSS an die verwendeten Dienste anpassen muss. Die Komponente der BSS, die für die Anpassung an die verwendeten Dienste zuständig ist, wird im folgenden „adaptiver Dialogkern" genannt.
In unserem Beitrag formulieren wir funktionale und nichtfunktionale Anforderungen an einen adaptiven Dialogkern und stellen darauf basierend ein mögliches Architektur-Modell vor, welches die Konfiguration und Einbindung von Diensten zur Laufzeit ermöglicht. Für die Realisierung beurteilen wir den Einsatz einer Bibliothek mit technischen Adaptern und den Einsatz von Rahmenwerken zur „Dependency Injection" (DI).

1 Einleitung

Dienst-orientierte Systemlandschaften sind durch eine hohe Heterogenität gekennzeichnet. Einerseits können Dienste integriert werden, die fachlich unterschiedliche Funktionalitäten anbieten, andererseits sind diese Dienste mit unterschiedlichen Technologien anzusprechen. Den EndbenutzerInnen jedoch soll beim Arbeiten mit Benutzungsschnittstellen (BSS) die technologische Heterogenität verborgen bleiben [Ha05].
Eine Möglichkeit Dienste mit unterschiedlichen fachlichen und technologischen Schnittstellen in eine BSS einzubinden, ist die Verwendung eines Dialogkerns [Rü06], der sich an diese Dienste anpassen (adaptieren) kann. Die Adaption auf Basis einer Konfiguration hat einen großen Vorzug: AdministratorInnen und EndbenutzerInnen können neue Dienste in die BSS einbinden, ohne dass die BSS neu gebildet und installiert werden muss. Die in einer BSS unterstützten Nutzungsfälle, welche auf den konfigurierten Diensten basieren, können somit von außen gesteuert werden und erfordern nicht mehr einen vollständigen Entwicklungs-Durchlauf. Konfigurierte Dienste können entweder bereits beim Start der BSS bekannt sein, oder erst während Nutzung der BSS durch Suche in Registraturen ergänzt werden (SOA-Prinzip). Ein adaptiver Dialogkern ermöglicht die Einbindung von neuen Diensten zur Laufzeit.

In Kapitel 2 stellen wir die Referenz-Architektur des adaptiven Dialogkerns vor, und in Kapitel 3 untersuchen wir die Eignung von verfügbaren Open Source Rahmenwerken zur Implementierung dieser Referenz-Architektur. Die Eignung basiert auf einem Vergleich von gewünschten funktionalen und nichtfunktionalen Eigenschaften mit den von Rahmenwerken angebotenen Eigenschaften. Kapitel 4 fasst den Beitrag zusammen.

2 Konzeption des adaptiven Dialogkerns

Ein adaptiver Dialogkern einer BSS nach [Rü06] ist in der Lage lokale oder verteilte Anwendungskerne auf einfache Weise einzubinden und anzusprechen. Anwendungskerne unterscheiden sich fachlich und technisch. Zentrale Bedeutung hat dabei die Steuerung des adaptiven Dialogkerns durch eine Konfiguration, welches es erlaubt Anwendungskerne zur Laufzeit einzubinden, ohne den Quellcode des Dialogkerns ändern und den Betrieb der BSS unterbrechen zu müssen.

2.1 Referenz-Architektur von Benutzungsschnittstellen

Benutzungsschnittstellen nach [Si04] bestehen aus einem GUI-Frontend (beispielsweise einem Web Browser), einer GUI-Engine (beispielsweise auf Basis der JavaServlet-Technologie), und einem Anwendungskern. Wird die BSS in einem Dienst-orientierten Umfeld eingesetzt, ist es allerdings möglich, dass mehrere Anwendungskerne eingebunden werden müssen. Dieser Fall ist in Abbildung 1 dargestellt. Das GUI-Frontend spricht mit Hilfe der Dialogpräsentation den Dialogkern an, um die gewünschte Funktionalität F1i bis F4i („i“ für „intern“) den EndbenutzerInnen über Nutzungsfälle zugänglich zu machen. Von zentraler Bedeutung ist der Dialogkern, der die verbindende Komponente zu den Anwendungskernen darstellt. Exemplarisch sind vier Anwendungskerne dargestellt, drei davon im Internet verteilt, die die verschiedenen Funktionalitäten F1e bis F4e („e“ für „extern“) bereitstellen und sich technologisch unterscheiden. In Abbildung 1 zeigen gestrichelte Linien lose Kopplung, die Indizes „i“ und „e“ verdeutlichen, dass sich die Datenstrukturen der Anwendungskerne von den internen Datenstrukturen der GUI-Engine (und den sichtbaren Feldern in den Dialogen) unterscheiden können.

2.2 Anforderungen an den adaptiven Dialogkern

Die funktionalen Anforderungen, die der adaptive Dialogkern erfüllen muss, damit die BSS neue Nutzungsfälle dynamisch unterstützten kann, lauten: **FR1:** Laufzeit-Erfassung von System-Funktionen und deren Qualität, die die BSS wünscht, und von Anwendungskernen, die diese System-Funktionen in Form von Operationen „implementieren“. **FR2:** Laufzeit-Erfassung von Bausteinen, die die Kommunikation zwischen BSS und Anwendungskernen ermöglichen (technische Adapter) und von Parametern zur Kommunikation (beispielsweise die Adresse eines Namensdienstes oder der Namen eines verteilten Objekts).

FR3: Laufzeit-Erfassung der Datenstrukturen der Operationen der Anwendungskerne und Abbildung auf die Datenstrukturen, die in den Dialog-Präsentationen benötigt werden.

Zudem muss der Dialogkern nicht-funktionale Anforderungen erfüllen: **NFR1:** Der Dialogkern muss die Möglichkeit zur **Erweiterung** durch Konfiguration (XML) bieten. Neue technische Adapter und deren Konfiguration müssen damit zur Laufzeit eingebunden werden können und neue Operationen und deren Datenstrukturen der Dialogpräsentation zugänglich gemacht werden können, ohne den Betrieb der BSS zu unterbrechen. **NFR2:** Der Dialogkern soll die **Testbarkeit** von Anwendungen dadurch unterstützen, dass verteilte Operationsaufrufe, die im produktiven Betrieb benötigt werden, ohne Änderung des Quellcodes auf lokale Operationsaufrufe, die während der Entwicklung der BSS benötigt werden, umgeschaltet werden können. **NFR3:** Der Dialogkern und die technischen Adapter müssen in der Lage sein die **Sicherheitsanforderungen** der BSS zu erfüllen. Es kann beispielsweise notwendig sein, sicherheitsrelevante Daten, wie Zahlungsdaten, nur verschlüsselt an einen Anwendungskern zu senden. **NFR4:** Der Dialogkern muss Aufrufe zu verteilten Anwendungskernen **performant** in Bezug auf Speicherverbrauch und Dauer durchführen.

2.3 Referenz-Architektur des adaptiven Dialogkerns

In der von uns vorgeschlagenen Referenz-Architektur für die innere Struktur des Dialogkerns, die in Abbildung 1 dargestellt ist, werden die funktionalen Anforderungen aus Kapitel 2.2 durch vier Arten von Komponenten beantwortet.

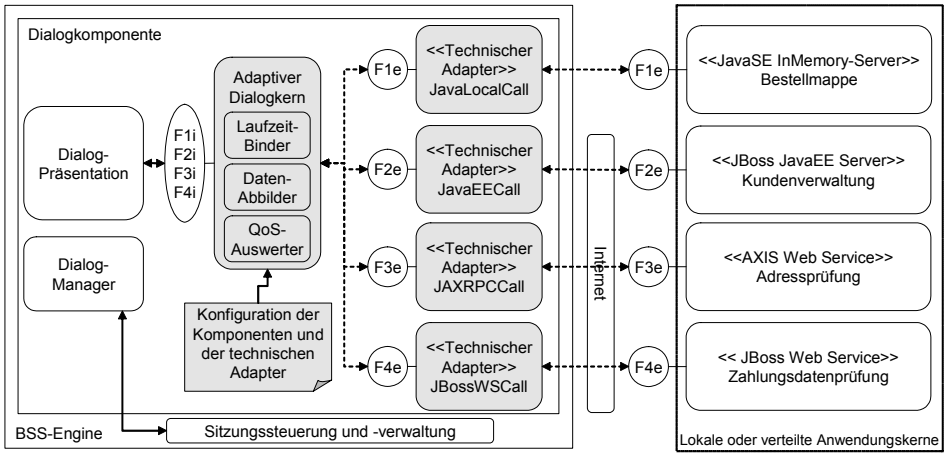

Abbildung 1: Einbau des adaptiven Dialogkerns in die Architektur von Benutzungsschnittstellen unter Verwendung exemplarischer Anwendungskerne und technischen Adaptern

Zentraler Bestandteil bildet der „**technische Adapter**“, von dem für jede einzubindende Technologie einer zur Verfügung stehen muss. Dieser besteht aus ausführbarem Code und einer Konfiguration. Die Konfiguration spezialisiert den Adapter für einen bestimmten Anwendungskern. Sie beinhaltet technische Parameter, wie Verbindungseinstellungen und Bibliotheken zur Kommunikation (über Socket, RMI, CORBA, SOAP etc.) mit den Anwendungskernen [Rü06]. Die Bibliotheken können verteilt vorliegen und müssen über das Internet geladen werden können.
Der adaptive Dialogkern kommuniziert mit Hilfe der Komponente „**Laufzeit-Binder**“ mit den Adapter-Komponenten, um Referenzen auf die Anwendungskerne zu erhalten und die Operationen der Anwendungskerne aufzurufen. Eine Konfiguration des Laufzeit-Binders ist nicht notwendig.
Der adaptive Dialogkern übernimmt mit Hilfe der Komponente „**Daten-Abbilder**“ die Abbildung der externen Datenstrukturen der Anwendungskerne auf die internen Datenstrukturen, die in den Oberflächen der BSS benötigt werden, und umgekehrt. Die Konfiguration enthält die Operationen der Anwendungskerne und des Dialogkerns, sowie alle Regeln zur Abbildung der Datenstrukturen.
Ergänzend besteht der Dialogkern aus einer Komponente „**QoS-Auswerter**“, die die vom Benutzer gewünschte Qualität, beispielsweise Sicherheit (Verschlüsselung von Zahlungsdaten und Signaturen von Bestellungen), an einen Dienst zur Suche nach einem geeigneten Anwendungskern übergibt. Im Fall von Web Services kann ein solcher Dienst von einer UDDI-Registratur angeboten werden [RP06]. Im weniger dynamischen Fall wird ein QoS-Auswerter benötigt, wenn ein Anwendungskern ausfällt und ein alternativer Anwendungskern angesprochen werden muss oder die Daten zwischenzeitlich lokal gespeichert werden sollen. Eine Konfiguration enthält die Namen der Registratoren und Regeln für die Auswahl gewünschter Anwendungskerne. Die Regeln können aus den Anforderungen der BSS zur Entwicklungszeit formuliert werden.

Nicht jede Komponente muss alle Anforderungen aus Kapitel 2.2 umsetzen:

Technische Adapter: FR2, NFR1, NFR2, NFR3
Laufzeit-Binder: FR1, NFR4
Daten-Abbilder: FR3, NFR1, NFR3, NFR4
QoS-Auswerter: FR1, NFR1, NFR2

2.4 Exemplarische Anwendung

Ein Unternehmen verwende intern eine BSS um die Bestellungen zu bearbeiten, die KundInnen im eShop des Unternehmens aufgegeben haben. Die BSS (Abbildung 1) nutzt dazu zwei Server, die durch Konfiguration bekannt sind: Eine „Kundenverwaltung“, die die Arbeit mit Kunden-, Produkt- und Bestelldaten erlaubt (F2e) und auf einem zentralen Rechner im Unternehmen installiert wurde. Und eine „Bestellmappe“, die die Bestellungen einer KundIn während der gesamten Bearbeitungsdauer im Speicher der BSS hält (F1e, ähnlich einem „Warenkorb“ im eShop), nachdem sie beim Start der BSS von einem zentralen Rechner im Unternehmen in die BSS eingeladen wurde.

Die BSS nutzt zusätzlich zwei externe Web Services von Partnerunternehmen, die ebenfalls durch Konfiguration bekannt sind: Einen Dienst zur „Adressprüfung“ (F3E) und einen Dienst zur „Zahlungsdatenprüfung“ (F4e). Die Server und die Services besitzen nicht nur unterschiedliche Funktionalitäten, sondern auch unterschiedliche Technologien zur Kommunikation zur Laufzeit (F2e, F3e, F4e) bzw. zur Einbindung (F1e) zur Laufzeit.

3 Realisierung des adaptiven Dialogkerns

Zur Realisierung des adaptiven Dialogkerns müssen die vier Komponenten aus Kapitel 2.3 entwickelt werden. Der Bedarf an technischen Adaptern jedoch ist erst bei Kenntnis der einzubindenden Anwendungskerne bekannt. Ebenso entwickelt werden muss ein Mechanismus zur Konfiguration der Komponenten. Dieses Kapitel stellt Open Source Rahmenwerke in Java vor, die versprechen, eine Eigenentwicklung zu ersparen und vergleicht diese gegen die in Kapitel 2.2 aufgestellten Anforderungen.

3.1 Eignung vorgefertigter technischer Adapter

Crispy [Crispy] ist ein Rahmenwerk zum Aufruf von verteilten Komponenten, die in unterschiedlichen Technologien implementiert sind. Crispy bietet dazu eine Klasse „ServiceManager“, die über Attribute gesteuert wird und verteilte Komponenten aufruft. CRIPSY unterstützt mehrere Technologien (JAX-RPC, XML-RPC, EJB) und verwendet dazu vorgefertigte Klassen wie beispielsweise den „JaxRpcExecutor“. Teilweise (REST, HTTP) werden eigene Implementierungen verwendet und keine externen Bibliotheken. Tabelle 1 zeigt einen Vergleich von Crispy und unserer Architektur in Bezug auf die in Kapitel 2.2 genannten funktionalen Eigenschaften. Die später vorgestellten DI-Rahmenwerke unterstützen nur die Container- und Konfigurations-Funktionalität des Dialogkerns und werden deshalb nicht aufgeführt.

Tabelle 1 Vergleich funktionaler Eigenschaften des adaptiven Dialogkerns und Crispy

	Adaptiver Dialogkern	**Crispy**
FR1	Laufzeit-Binder: QoS-Auswerter:	ServiceManager-Klasse und deren Properties. Nicht unterstützt.
FR2	Technische Adapter:	Technologiespezifische Klasse (z.B. JaxRpcExecutor).
FR3	Daten-Abbilder:	Nicht unterstützt.

Der Laufzeit-Binder und die technischen Adapter aus Abbildung 1 können vollständig durch den ServiceManager und die technologiespezifischen Klassen von Crispy (für lokale Java-Operationsaufrufe, JAX-RPC, XML-RPC, RMI, EJB, JBoss Remoting, REST, HTTP, CORBA) realisiert werden. Die Komponenten QoS-Auswerter und Daten-Abbilder können allerdings nicht mit Hilfe von Crispy realisiert werden, da Crispy diese Funktionalität nicht adressiert, diese sind selbst zu entwickeln.

3.2 Eignung vorgefertigter Komponenten-Container

Die Kollaboration einer Client-Komponente mit einer Server-Komponente ist nicht auf das Verhältnis von Dialogkern und Anwendungskern beschränkt. Jede kollaborierende Komponente muss wissen, mit welcher Komponente zu kommunizieren ist, wo diese sich befindet und wie mit dieser kommuniziert werden kann. Sind diese Parameter in der Client-Komponente festgelegt, reduziert sich deren Qualität in Bezug auf Austauschbarkeit, Wiederverwendbarkeit und Erweiterbarkeit. Das „Dependency Injection (DI) Pattern“ (aka „Inversion of Control“), vermindert die Abhängigkeit einer Client-Komponente von einer bestimmten Server-Komponente. Die Client-Komponente erhält von einer Such-Komponente (Assembler) eine Referenz auf die gewünschte Server-Komponente. Die Client-Komponente enthält nur noch die Abhängigkeit von der Such-Komponente und der gewünschten fachlichen Schnittstelle der Server-Komponente. Drei Mechanismen zur DI werden unterschieden [Fo04]:

1. Interface Injection: Client-Komponenten implementieren eine Schnittstelle, über die ein Assembler Server-Implementierungen liefert (injection).

2. Setter-Injection: Client-Komponenten bieten JavaBeans-Eigenschaften (set-Methoden), über die sie Server-Implementierungen erhalten.

3. Constructor Injection: Client-Komponenten bieten einen Konstruktor, über den sie Server-Implementierungen erhalten.

Derzeit existieren mehrere DI-Rahmenwerke: SpringCore [SpringCore], PicoContainer [PicoContainer] und Apache HiveMind [HiveMind].

Im Falle unseres adaptiven Dialogkerns können DI-Rahmenwerke verwendet werden, um den Dialogkern von Implementierungen der technischen Adapter unabhängig zu machen. Dies ist in Abbildung 2 dargestellt. Der Quellcode des Dialogkerns enthält nur Aufrufe der gemeinsamen Schnittstelle der technischen Adapter. Der Assembler liest die Konfiguration der technischen Adapter ein, konstruiert die technischen Adapter und gibt Referenzen an den Dialogkern, unter Anwendung eines Injection-Typs. DI-Rahmenwerke stellen keine technischen Adapter bereit, da DI-Rahmenwerke Domänen-unabhängig sind. Der Quellcode des Dialogkerns enthält nur die minimal notwendigen Abhängigkeiten und kann auf flexible Art und Weise neue technische Adapter erhalten.

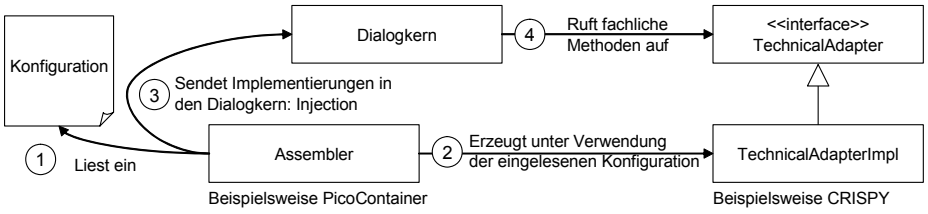

Abbildung 2 Anwendung der Dependency Injection auf die Beziehung von Dialogkern und dynamisch einzubindenen technischen Adaptern mit Hilfe eines DI-Rahmenwerks

Der **SpringCore** [SpringCore], als Kern des Spring-Rahmenwerks, bietet eine leichtgewichtige Basisinfrastruktur zur Unterstützung von JavaBeans und der DI durch die Klasse BeanFactory. Die BeanFactory kann vom Dialogkern genutzt werden, um den Lebenszyklus der technischen Adapter zu verwalten. Die technischen Adapter müssen dazu aber durch eine JavaBean gekapselt werden [Jo05].
Der **PicoContainer** [PicoContainer] kann für den adaptiven Dialogkern verwendet werden um den Lebenszyklus der Adapter-Komponenten zu verwalten und Adapter-Komponenten dem Dialogkern zu senden. Allerdings werden in unserem Fall nicht die Vorteile des PicoContainer ausgenutzt, die in einer starken Vernetzung der verwalteten Objekte liegen, da die Adapter-Komponenten keine starken Abhängigkeiten voneinander besitzen.
Der **HiveMind** [HiveMind] Mikro-Kernel verwaltet Dienste, die in einer Java Virtual Machine leben. Dienste sind Fassaden, die in Java-Klassen („Plain Old Java Objects“) implementiert sind. Die Dienste werden von HiveMind instanziiert und konfiguriert, beispielsweise zu Beginn der Anwendung. HiveMind läßt die Dienste miteinander kollaborieren mittels DI. Mehrere Dienste und deren Konfigurationen werden in Modulen (Java-Archiven) gebündelt, die in "Module Deployment Descriptors“ beschrieben werden. Der Mikro-Kernel kann als Dialogkern dienen, die Dienste kapseln die technischen Adapter.

Tabelle 2 Vergleich nichtfunktionaler Eigenschaften von DI-Rahmenwerken

	SpringCore	**Pico**	**HiveMind**
(**NFR1**) (**NFR3**) Verwaltet „Plain Old Java Objects“ ohne spezifische Erweiterungen:	Ja. Der Lebenszyklus ist wahlweise kontrollierbar mit Hilfe der Schnittstellenklasse Startable.	Ja.	Ja.
(**NFR4**) Unterstützt Caching:	Ja. Ein deklaratives Caching mit EHCache, JBoss Cache, Java Caching System, OSCache, und Tangosol Coherence ist möglich.	Ja. Eigene Implementierung des Cachings.	Ja, eigene Implementierung des Cachings.
(**NFR1**) (**NFR2**) Erlaubt Konfiguration in Datei:	Ja, mit Hilfe der Klasse XmlBeanFactory.	Ja, mit Hilfe eines [NanoContainer].	Ja.

(**NFR1**) Verlangt festes Schema der Konfiguration:	Ja, vorgegebenes DTD.	Ja, vorgegebenes DTD.	Nein, definierbar spezifisch für Anwendung.
(**NFR1**) (**NFR2**) Erlaubt Generierung einer Dokumentation der Konfiguration:	Nein.	Nein.	Ja, mit HiveDoc.
(**NFR1**) Ist gut dokumentiert:	Sehr gut.	Gut.	Gut.

Wie in Tabelle 2 dargestellt, erscheint der SpringCore durch das Angebot von optionaler Lebenszyklus- und Caching-Funktionalität, als das flexibelste DI-Rahmenwerk, welches zudem sehr gut dokumentiert ist. Gleichzeitig bietet der Einsatz des SpringCore die Möglichkeit die ergänzenden Spring Integrations-Schichten „JEE“, „Web“, „ORM“, „DAO“ und „AOP“ nahtlos in eigenen Anwendungen einzusetzen.

4 Zusammenfassung

In diesem Beitrag haben wir die Einsatzmöglichkeit des adaptiven Dialogkerns vorgestellt, funktionale und nicht-funktionale Eigenschaften definiert, daraus eine Referenz-Architektur abgeleitet und abschließend überprüft, ob sich der Einsatz von Crispy und eines DI-Rahmenwerkes zur Implementierung des adaptiven Dialogkerns lohnt. Der Einsatz von Crispy ist sinnvoll, um den eigenen Implementierungsaufwand von technischen Adaptern zu reduzieren. Der Einsatz eines DI-Rahmenwerks ist sinnvoll um die Dauer der Realisierung zu verkürzen, die Qualität des Quellcodes zu erhöhen und gleichzeitig eine hohe Flexibilität durch Konfiguration des Dialogkerns zu erreichen. Wir haben uns für das DI-Rahmenwerk SpringCore entschieden, da es besonders flexibel ist. Auf Basis der hier vorgestellten Entscheidungen ist nun unser Ziel den vorgestellten adaptiven Dialogkern zu entwickeln.

Literaturverzeichnis

[Crispy] Crispy Communication per Remote Invocation for different kinds of Services via ProxYs. http://crispy.sourceforge.net/, Stand 01.2007.

[Fo04] Martin Fowler: Inversion of Control Containers and the Dependency Injection Pattern. http://martinfowler.com/articles/injection.html, 01.2004.

[Ha05] Ulrike Hammerschall: Verteilte System und Anwendungen, Kapitel 1.3.3 Transparenz. Pearson Studium, 1. Auflage, 2005.

[HiveDoc] Apache HiveDoc Tool, Dokumentation einer HiveMind Registratur. The Apache Software Foundation. http://hivemind.apache.org/hivedoc.html. Stand 01.2007.

[HiveMind] Apache HiveMind Project. The Apache Software Foundation. http://hivemind.apache.org/. Stand 01.2007.

[JEE] Java Platform, Enterprise Edition. Sun Microsystems. http://java.sun.com/javaee/. Stand 01.2007.

[Jo05] Rod Johnson: Introduction to the Spring Framework. http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework. 05.2006.

[JSE] Java Platform, Standard Edition. Sun Microsystems. http://java.sun.com/javase/. Stand 01.2007.

[NanoContainer] NanoContainer Project mit Produkt NanoWar, http://www.nanocontainer.org/, http://nanocontainer.org/dtd/nanocontainer-xml-1_0.dtd. Stand 01.2007.

[PicoContainer] PicoContainer Project mit Produkt IoC Component Container. PicoContainer Organisation http://picocontainer.codehaus.org/, 01.2007.

[RP06] Jürgen Rückert, Barbara Paech. Web Service Quality Descriptions for Web Service Consumers. Best Paper Award. Proceedings der CONQUEST 2006, dpunkt.verlag, Seite 203, Heidelberg.

[Rü06] Jürgen Rückert, Jerko Horvat, Dinh-Khoa Nguyen, Stefan Becker, Barbara Paech: Modell zur Adaption eines Dialogkerns. Modellierung 2006, Workshop Qualität von Modellen, Innsbruck, 03.2006. http://www-swe.informatik.uni-heidelberg.de/research/publications/Modellierung2006-AdaptionDialogkern.pdf

[Si04] Johannes Siedersleben: Moderne Software-Architektur. Umsichtig planen, robust bauen mit Quasar. DPunkt-Verlag, Heidelberg, 2004.

[SpringCore] Spring Framework, Java/JEE Anwendungsrahmenwerk. Interface21. http://www.springframework.org/. Stand 01.2007.