Mediatorbasierte ad hoc Integration autonomer Web Services

Uwe Radetzki, Thomas Bode, Armin B. Cremers

Institut für Informatik III
Universität Bonn
Römerstr. 164
D-53117 Bonn
{ur,tb,abc}@iai.uni-bonn.de

Abstract: Ad hoc Integration autonomer Dienste ist ein wichtiger Bestandteil bei der dynamischen Fusion verteilt vorliegender Daten und Methoden. In diesem Papier stellen wir ein komponentenbasiertes Mediatormodell vor, mit dessen Hilfe wir hoffen, dieser Vision einen entscheidenden Schritt näher zukommen. Das Modell erlaubt die Beschreibung der benötigten Mediatoren und ermöglicht eine effiziente Identifikation und Komposition existierender Funktionalitäten. Die Darstellung der relevanten, semantischen Kontextinformation erfolgt dabei ontologiebasiert.

1 Einleitung

Die Fähigkeit die in konkreten Anwendungssituationen relevanten Informationen schnell und je nach Aufgabe flexibel kombinieren zu können, spielt in vielen Bereichen (z.B. im Katastrophenmanagement [Be03]) eine entscheidende Rolle. Allgemein gesehen werden Verfahren benötigt, die eine ad hoc Integration von heterogenen und autonomen Informationsdienste leisten bzw. geeignet unterstützen. Als Schlüsseltechnologie für die Komposition solcher Dienste werden heute insbesondere Service-Orientierte Architekturen (SOAs) auf der Basis von Web Services vorgeschlagen. In den Forschungsprojekten IRIS (Interoperability and Reusability of Internet Services) [RMC04] und COBIDS (COmponent-Based Integration of Data and Service) [Bo04] beschäftigen wir uns seit einigen Jahren mit der aufgabenorientierten Integration verteilter Informationsdienste auf der Basis von Web Services. Ein grundlegendes Konzept in beiden Ansätzen ist der Einsatz von aufgabenspezifischen Mediatoren zur Überbrückung der vorhandenen semantischen und strukturellen Heterogenitäten der beteiligten Dienste [RC04]. Damit bilden diese Mediatoren die zentrale „Plattform“ für die eigentliche Fusion und Verarbeitung der verteilt vorliegenden Daten und Dienste (*mediatorbasierte Informationsfusion*).

Natürlich sollten solche Mediatoren nicht ständig neu entwickelt werden. Darum erfolgt ihre Realisierung komponentenbasiert, so das eine spätere *Adaption* des Verhaltens bzw. eine Erweiterung und *Komposition* der Mediatoren ermöglicht wird [SGM02]. Darüber

hinaus streben wir die Schaffung eines Pools registrierter, aber im Internet verteilt vorgehaltener anpassbarer Mediatoren an (vgl. UDDI für Web Services). Als erster Schritt bei der Suche nach geeigneten Mediatoren in diesem Pool wird vom System zunächst aus den Schnittstellenbeschreibungen der an der Fusion beteiligten Dienste eine Beschreibung der benötigten Mediatorfunktionalität generiert (*Anfrageprofil*). Ein spezieller Suchprozess gleicht anschließend diese Beschreibung mit der Beschreibung der Mediatoren des Pools ab und schlägt möglichst geeignete Mediatoren vor. Hierbei wird auch eine mögliche Komposition unabhängiger Mediatoren berücksichtigt. Die Adaption der konkret benötigten Mediatorfunktionalität erfolgt dann durch den Benutzer in einem vom System unterstützten Prozess. Gerade die komponentenorientierte Realisierung der Mediatoren schafft hier die benötigte Flexibilität und unterstützt die Wiederverwendung einzelner Softwarebausteine in unterschiedlichen Kontexten. Die Funktionalität der neu geschaffenen oder auch nur gezielt angepassten Mediatoren wird selbstverständlich im Mediatorenpool registriert und steht für andere Nutzungskontexte zur Verfügung.

Verfahren zur automatischen Generierung von Mediatoren, z.B. inhaltsorientierte oder ontologiebasierte Ansätze [BL04], können leicht in den beschriebenen Ansatz einbezogen werden. Wir sind jedoch davon überzeugt, dass die automatische Generierung alleine die komplexen Probleme bei der Fusion sehr heterogener Dienste auf absehbare Zeit nicht lösen kann. Gerade die Kombination automatisch generierter und spezialisierter, domänenabhängiger Mediatorkomponenten erscheint hier ein viel versprechender Weg.

Die Sprachen WSDL (Schnittstellenbeschreibung) und UDDI (Registrierung) des Basis Stack der Web Services sind für eine mediatorbasierte Informationsfusion ungeeignet, da es ihnen an geeigneten Methoden zur Beschreibung expliziter Semantik [Pa02] und expliziten Anpassungsfähigkeiten mangelt. Daher ist ein alleiniges Abstützen auf diese Sprachen für den Umgang mit Mediatoren ungenügend. OWL-S hingegen erlaubt eine semantische Auszeichnung von Diensten, jedoch mit dem Fokus auf allgemeine Geschäftsdienste [Co02]. Diese Sprache ist für das von uns vorgeschlagene Mediatormodell nicht restriktiv genug. Auch hier lassen sich explizite Anpassungsfähigkeiten nicht beschreiben. Im Folgenden stellen wir unsere Mediatorbeschreibungssprache und das ihr zu Grunde liegende Mediatormodell vor.

2 Ein komponentenbasiertes Mediatormodell

Ein komponentenbasierter Ansatz ermöglicht hochgradig flexibel anpassbare und wiederverwendbare Software. Um diese Qualitäten auf im Internet verfügbare Mediatoren übertragen zu können, bedarf es einer Beschreibungssprache, die sowohl die Syntax als auch die Semantik eines Mediators offen legt, und auch komponierte Mediatoren darstellen kann. Wichtig ist die Unterscheidung zwischen den Anpassungsfähigkeiten sowie den Mediationsmöglichkeiten eines Mediators. Eine derartige Beschreibungssprache stellt die entwickelte *IRIS Mediator Profile Language* (IRIS-MPL[1]) dar. IRIS-MPL ist auf Basis der Ontology Web Language (OWL) definiert und gliedert sich nahtlos an

[1] IRIS-MPL findet sich unter *http://www.cs.uni-bonn.de/~ur/iris/MediatorProfile.owl*

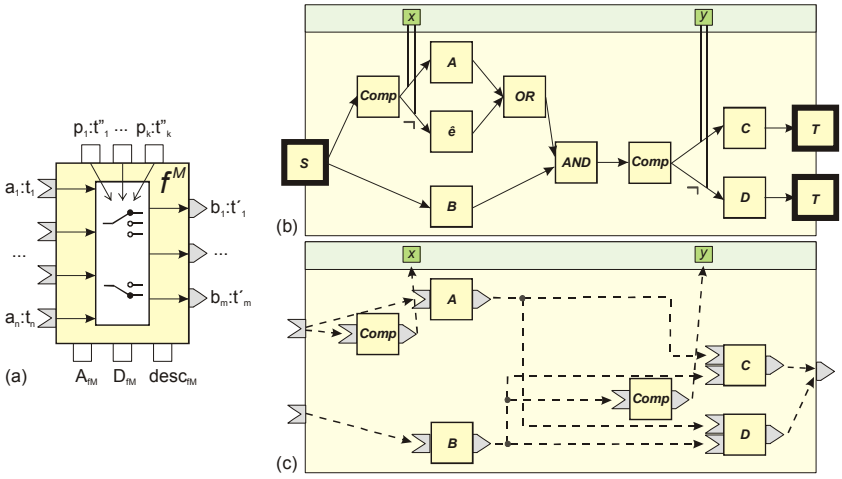

Abbildung 1: (a) Schematische Darstellung einer Functionality inklusive ihrer Anpassungsfähigkeiten; (b) Kontrollflussgraph und (c) Datenflussgraph eines Mediatorpatterns

Dienstbeschreibungssprachen wie OWL-S [Co02] und an Ontologien, wie sie z.B. im Kontext des Semantic Web entstanden sind bzw. zur Zeit entstehen. IRIS-MPL unterteilt sich in zwei Bereiche, die im Folgenden kurz skizziert werden.

2.1 Mediatorprofil eines komponentenbasierten Mediators

Das *Mediatorprofil* beschreibt im wesentlichen syntaktische und semantische Fähigkeiten eines Mediators. In unserem Modell unterscheiden wir strikt die Mediationsmöglichkeiten, im Folgenden *Functionalities*, von den Anpassungsfähigkeiten, im Folgenden *Properties*. Die Properties erlauben das Verhalten einer Functionality entsprechend den Anforderungen einer konkreten Fusionssituation modifizieren zu können (z.B. Spezifikation des gewünschten „Ausgabeformates"). Functionalities beschreiben hingegen die tatsächlichen Mediationsmöglichkeiten, die bei einer Dienstfusion automatisch angesprochen werden können. Unter einem komponentenbasierten Mediator verstehen wir dann eine Sammlung von anpassbaren Functionalities und deren Properties.

Jede Functionalitiy besitzt eine Menge von *Eingabe-* und *Ausgabeports* für den Datenaustausch. Jeder Port lässt sich sowohl syntaktisch als auch semantisch auszeichnen. Dies geschieht zum einen durch Verweise auf ein Typsystem (z.B. XML Schema Datatypes) zum anderen durch Verweise auf eine gemeinsam genutzte Ontologie (*Shared Ontology*), die die relevanten Konzepte aus den beteiligten Anwendungsdomänen beschreibt. Weiterhin wird jede Functionality inhaltlich ausgezeichnet (z.B. durch eine textuelle Beschreibung $desc_{fM}$ oder eine Anwendungsdomäne D_{fM}). Abbildung 1(a) illustriert schematisch das Modell einer einzelnen Functionality f^M inklusive ihrer Properties ($p_1,...,p_k$) sowie der Eingabeports ($a_1,...,a_n$) und der Ausgabeports ($b_1,...,b_m$). Leider müssen wir uns aus Platzgründen in diesem Paper auf abstrakte Beispiele beschränken, da die Darstellung realer Mediationssituationen sowie der hierzu benötigten Mediatoren

und ihrer Semantiken zu umfangreich wäre. Dies gilt auch für den Aspekt der Properties bei den im Folgenden eingeführten Mediatorpattern.

2.2 Komponierte Mediatoren: Mediatorpattern

Der zweite Teil der IRIS-MPL erlaubt die Spezifikation eines *Mediatorpattern*, d.h. die Definition einer neuen Functionality auf Basis einer Komposition mehrerer vorhandener Functionalities. Hierbei müssen prinzipiell zwei Nutzungskontexte unterschieden werden. Der erste Nutzungskontext ergibt sich daraus, dass innerhalb einer gegebenen Dienstfusion eine konkrete Functionality benötigt wird, die aber nur durch eine Verschaltung vorhandener Functionalities realisiert werden kann (*ad hoc Komposition*). Hierbei sind sowohl Eingabe- als auch Ausgabeports durch das Anfrageprofil eindeutig spezifiziert (vgl. Abschnitt 1). Der zweite Nutzungskontext resultiert aus der Anforderung, dass ein Mediatorentwickler eine neue (komplexe) Functionality dadurch erstellen möchte, indem er auf bereits vorhandene Functionalities zurückgreift (*benutzerdefinierte Komposition*). Soll dies ohne explizites Neuprogrammieren möglich sein, erfordert das komplexere Kompositionsmöglichkeiten, da hier explizit Fallunterscheidungen hinsichtlich Daten und Properties ausgedrückt werden müssen.

Generell ist unser Verständnis eines Mediatorpatterns durch die Definition von Kompositionsgraphen geprägt. Abbildung 1(b) illustriert den *Kontrollflussgraphen* eines abstrakten Mediatorpatterns. Dieser Graph beschreibt die Aktivierungsreihenfolge der komponierten Functionalities. Neben den Knoten, die konkrete Functionalities repräsentieren (*A, B, C, D*) existieren weitere Knoten, die den Kontrollfluss modulieren. Hierzu zählen Knoten, die eingehende Aktivierungen synchronisieren (*AND, OR*), aber auch Knoten, so genannte *Komparatoren* (*Comp*), die Vergleiche für bedingte Kanten (s.u.) auswerten. Ein *OR*-Knoten wartet, bis ihn eine eingehende Kante aktiviert und gibt dann die Aktivierung an nachfolgende Knoten weiter. Später eintreffende Aktivierungen werden verworfen. Ein *AND*-Knoten wartet solange, bis alle eingehenden Kanten ihn aktiviert haben, bevor er die Aktivierung weiterreicht. Die Aktivierung des gesamten Patterns geschieht über einen eindeutigen *Source-Knoten* (*S*) und endet an einem *Target-Knoten* (*T*). Mit der Einführung eines „leeren“ Knotens (*ê*) soll dem *Death-Path Problem* entgegengetreten werden, d.h. dem Problem nicht weiter ausführbarer Aktivierungspfade. Die Kanten des Graphen können einfach oder bedingt sein. Eine bedingte Kante verweist auf eine boolsche Variable (*x, y*), welche angibt, ob diese Kante die Aktivierung an den nachfolgenden Knoten weiterleitet oder nicht.

In Abbildung 1(c) wird der *Datenflussgraph* desselben Patterns dargestellt. Die an den Eingabeports des Pattern eingehenden Daten werden an die entsprechenden Functionalities und Komparatoren weitergeleitet. Anschließend werden sukzessiv Ausgabeports mit Eingabeports der im Kontrollfluss nachfolgenden Functionalities verbunden. Abschließend werden der Semantik der Komposition entsprechend die resultierenden Daten auf die Ausgabeports des Patterns übergeben. Hierbei ist zu beachten, dass die Signatur des resultierenden Patterns unabhängig davon ist, welcher Target-Knoten erreicht wird und wie die entsprechenden Daten an den Ausgabeports angelegt werden. Häufig wird man

hier den Fall finden, dass das Resultat ein XML-Dokument ist, welches durch einem einfachen String kodiert wird.

Jedes Mediatorpattern definiert eine neue Functionality, die ebenfalls durch ein Mediatorprofil beschrieben und im Pool aller Functionalities registriert wird. Aus diesem Grund lassen sich Mediatorpattern auch innerhalb eines anderen Patterns verwenden.

3 Fazit

In diesem Papier haben wir ein komponentenbasiertes Mediatormodell vorgestellt, mit dessen Hilfe wir der Vision einer ad hoc Integration autonomer Dienste einen entscheidenden Schritt näher gekommen sind. Dieses Modell profitiert von der auf OWL basierenden Beschreibungssprache IRIS-MPL, sowie den Anpassungsfähigkeiten der Komponententechnologie. IRIS-MPL erlaubt die Interna eines Mediators beschreiben und zur Laufzeit explorieren zu können. Ohne dies wäre eine Identifizierung adäquater Mediatoren und deren Anpassung, Ladung und Aktivierung nicht möglich. Zudem erlaubt es die Erstellung neuer Funktionalitäten durch Komposition von bereits vorhandenen Funktionalitäten. Die Identifizierung von Mediatoren lässt sich durch ontologische Mittel unterstützen (siehe hierzu [RC04]).

Literaturverzeichnis

[Be03] Bernard, L. et. al.: Ontologies for Intelligent Search and Semantic Translation in Spatial Data Infrastructures. Photogrammetrie - Fernerkundung - Geoinformation (PFG), 2003(6); S. 451-462.

[Bo04] Bode, T., Radetzki, U., Shumilov, S., Cremers, A.B.: The development of COBIDS: A Component-Based Framework for Sharing Standardized and Non-Standardized Geo-Services. 18th Int. Conf. Informatics for Environmental Protection (EnviroInfo), Geneva, Switzerland, 2004, accepted paper.

[BL04] Bowers, S., Ludäscher, B.: An ontology-driven framework for data transformation in scientific workflows. Int. Workshop on Data Integration in the Life Sciences (DILS'04), 2004.

[Co02] The DAML Services Coalition: DAML-S: Web Service Description for the Semantic Web, Proc. of Int. Semantic Web Conf. (ISWC), 2002.

[Pa02] Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matching of Web Service Capabilities. Proc. of Int. Semantic Web Conf. (ISWC), 2002.

[RC04] Radetzki, U., Cremers, A.B.: IRIS: A Framework for Mediator-Based Composition of Service-Oriented Software. Proc. of 2nd Int. Conf. of Web Services (ICWS), San Diego, USA, 2004, in press.

[RMC04] Radetzki, U., Mancke, S., Cremers, A.B.: IRIS: A Framework Supporting Composition and Device-Specific Access of Software Services. 18th Int. Conf. Informatics for Environmental Protection (EnviroInfo), Geneva, Switzerland, 2004, accepted paper.

[Sy02] Sycara, K., Widoff, S., Klusch, M., Lu, J.: LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and MulitAgent Systems, Kluwer Academic Publisher, (5), 2002, S. 173-203.

[SGM02] Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. Component Software Series. Addison-Wesley, 2 edition, 2002.