

HARALD SCHÖNING

# Registry/Repository: Vom XML-Datenbanksystem zum Herzstück der serviceorientierten Architektur

In einer SOA-Implementierung finden sich Services in verschiedenen Ebenen, wobei die höheren Ebenen die Dienste der niedrigeren in Anspruch nehmen. Zur Gewährleistung von Wiederverwendung, aber auch zur Erhaltung der Wartbarkeit ist ein Verzeichnis (Registry) erforderlich. Eine sinnvolle Ergänzung dazu ist ein angegliedertes Repository. Hat man diese Kombination im Einsatz, eröffnen sich weitere Nutzungsmöglichkeiten. Mit CentraSite wird ein Beispiel einer solchen Registry/Repository-Kombination vorgestellt, und weiter gehende Einsatzmöglichkeiten werden diskutiert.

## 1 Einleitung

Die serviceorientierte Architektur (SOA) [Erl 2005] wird heute vielfach als Grundprinzip einer modernen IT-Infrastruktur zur Unterstützung von Geschäftsprozessen innerhalb einer Unternehmung und zwischen Unternehmungen angesehen. Weder der Name noch das Prinzip der SOA sind wirklich neu: Der Begriff wurde 1996 von Gartner-Analysten geprägt [Schulte & Natis 1996] und definiert als »a style of multitier computing that helps organizations share logic and data among multiple applications and usage modes«.

Die Wiederverwendung, die hier als Wesensmerkmal einer SOA herausgestellt wird, ist auch der Beweggrund für das größer werdende Interesse an einer Umstellung betrieblicher Systeme auf eine SOA-konforme Realisierung: Man verspricht sich Einsparungen durch Wiederverwendung von Geschäftslogik, sei es innerhalb eines Betriebs oder in der Kooperation mehrerer Betriebe: »The main drivers for SOA-based architectures are to facilitate the manageable growth of large-scale enterprise systems, to facilitate Internet-scale provisioning and use of services and to reduce costs in organization to organization cooperation« [MacKenzie et al. 2006]. Ferner verspricht man sich durch eine leichtere Umkonfiguration oder den Austausch isolierter Einheiten eine größere Agilität und geringere Zeiten für die Anpassung der Geschäftsprozesse an neue Gegebenheiten.

Konsequenterweise erfordert eine SOA die lose Kopplung der beteiligten Komponenten, insbesondere deren Plattformunabhängigkeit und Austauschbarkeit.

Lediglich die Schnittstelle einer Komponente muss bekannt sein, um sie in einer SOA verwenden und ansprechen zu können, die Implementierung der Komponente kann auf einer vom Klienten völlig verschiedenen Plattform, in einer anderen Programmiersprache und auf einer entfernten Maschine erfolgen. Eine solche Komponente, die über eine bekannte Schnittstelle einen definierten Dienst anbietet, wird Service genannt.

Im Rahmen dieses Beitrags wird, nach einer allgemeinen Übersicht über Komponenten einer SOA, eine dieser Komponenten, die Registry, genauer betrachtet. Als Beispiel einer Kombination aus Registry und Repository wird das Produkt CentraSite der Software AG kurz vorgestellt, das auf dem XML-Datenbanksystem Tamino basiert. Dabei wird deutlich, dass die Realisierung einer solchen SOA-Komponente sehr von den Fähigkeiten eines XML-Datenbanksystems profitiert. Schließlich wird ein Ausblick gegeben, wohin sich die Funktionalität eines Registry/Repository entwickeln kann.

## 2 Komponenten einer serviceorientierten Architektur

Eine SOA setzt keine spezielle technische Infrastruktur voraus. Eine frühe Infrastruktur zur Realisierung einer SOA war beispielsweise CORBA [CORBA]. Natürlich bieten sich heute besonders Web-Services als Infrastruktur für die Implementierung einer SOA an, da inzwischen fast jede Plattform und jede Programmiersprache erlauben, Web-Services zu realisieren.

Am Beispiel von Web-Services lassen sich auch Grundprinzipien einer SOA illustrieren: Der Dienstgeber (Service-Provider) weiß im Allgemeinen nicht, wer den Service benutzen wird. Die Schnittstelle eines Web-Service wird durch eine WSDL-Definition (Web Services Description Language) [Christensen et al. 2001] beschrieben, aus der die angebotenen Operationen und deren Parameter mit ihren Datentypen hervorgehen. Eine Beschreibung der Semantik kann – in Prosa – ebenfalls enthalten sein. Mechanismen für eine formalere, auf Techniken des Semantic Web beruhende Beschreibung werden noch diskutiert [OWL-S], werden aber sicherlich über kurz oder lang allgemein zum Einsatz kommen.

Eine WSDL-Beschreibung eines Service lässt jedoch keine Rückschlüsse auf dessen Realisierung zu. Selbst die Adresse, unter der der Service angesprochen wird, kann in einer WSDL fehlen.

Entweder als Erweiterung der WSDL oder separat kann der funktionalen Beschreibung noch eine nicht funktionale hinzugefügt werden, die beispielsweise Richtlinien (Policies) für die Benutzung des Service spezifiziert oder Qualitätssicherungen (etwa Verfügbarkeit) gibt.

Services können ganz verschiedene Ebenen innerhalb von Geschäftsprozessen realisieren: von der einfachen Infrastrukturdienstleistung (aktueller Börsenkurs eines bestimmten Wert-

papiers) bis hin zu komplexer Geschäftslogik (Bonitätsprüfung). Daher ist es nahe liegend und auch im Sinne einer SOA, wenn die Implementierung von Services selbst wieder Services benutzt, die zur Erfüllung des Gesamtzwecks geeignet miteinander verknüpft werden. Durch eine solche Orchestrierung werden aus feingranularen Services höherwertige geschaffen. Dabei entstehen durchaus auch Services, die menschliche Interaktion erfordern können oder komplexe Abläufe beinhalten, die durch ausgefeilte Logik in der Serviceimplementierung gesteuert werden, also *Workflow* oder Geschäftsprozesse im eigentlichen Sinn. Eine bekannte Sprache zur Formulierung von Geschäftsprozessen, die auch die Gegebenheiten einer SOA gut integriert, ist BPEL (Business Process Execution Language). Ein in BPEL beschriebener Prozess kann wiederum als (Web-)Service angeboten werden. Schließlich kann aus Services und grafischen Oberflächen eine sogenannte *Composite Application* zusammengestellt werden, für die es wenig oder keiner Programmierung bedarf.

Bei der Umstellung einer IT-Landschaft auf eine SOA wird im Allgemeinen kein revolutionärer Ansatz verfolgt, der einen vollständigen Austausch vorhandener Software voraussetzen würde, sondern ein integrativer Ansatz, der vorhandene (*Legacy*-)Systeme mit einbindet, indem diese sich ebenfalls als Service (*Web-Service*) positionieren. Entsprechende Adaptersoftware erlaubt ein nicht invasives Hinzufügen einer Serviceschnittstelle auch für Mainframe-basierte Software. Je nach Struktur und Engriffsmöglichkeiten in diese Software ist auch eine stärker integrierte Anbindung möglich. Eine solche Öffnung vorhandener Komponenten wird in [Channabasavaiah et al. 2003] sogar als Voraussetzung für eine SOA-Implementierung genannt.

Neben vorhandenen Anwendungen müssen auch vorhandene Daten in eine SOA eingebunden werden können. Hier kommen die Techniken der Informationsintegration zum Zug, die eine servicebasierte Schnittstelle zu vorhandenen Datenbanken und anderen Datenquellen realisieren, wobei sie die im Rahmen von SOA geforderte Abstraktion leisten und den Ursprung der von ihnen gelieferten Daten verbergen, insbesondere im Fall der Kombination aus mehreren Datenquellen.

Abbildung 1 zeigt eine beispielhafte Schichtenbildung in einer SOA.

Selbstverständlich funktionieren lose Kopplung und einfache Austauschbarkeit nur, wenn Schnittstellen und Kommunikationsprotokolle einen gemeinsamen Standard erfüllen. Dieser Aspekt ist es, der die Implementierung einer SOA auf der Basis von Web-Services attraktiv macht, da es für diese entsprechende weltweit akzeptierte und verbreitete Standards gibt. Zu nennen sind hier vor allem WSDL [Christensen et al. 2001] zur Schnittstellenbeschreibung und SOAP (Simple Object Access Protocol) [Gudgin et al. 2003] als Kommunikationsprotokoll. Wie fast alle im Rahmen von Web-Services relevanten Standards basieren WSDL und SOAP auf XML, wobei WSDL auch XML-Schema benutzt.

### 3 Die Registry

Wie bereits erwähnt, ist eines der Hauptziele von SOA, wiederverwendbare Komponenten zu schaffen und auch zu benutzen. Wiederverwendung von Services setzt jedoch voraus, dass die zur Verfügung stehenden Services bekannt sind bzw. nachgeschlagen werden können. Der reine Name eines Service ist noch nicht ausreichend, um die Verwendbarkeit für einen bestimmten Zweck zu erschließen, vielmehr müssen auch die Funktion des Service und seine Schnittstellen offenliegen. Organisatorische Information, wie der Anbieter des Service oder die vom Service zugesicherte Verfügbarkeit, Sicherheit usw., spielen ebenfalls eine Rolle bei der Beurteilung der Frage, inwieweit ein bestimmter Service in einer SOA einen bestimmten Platz einnehmen kann. Da man in einer ausgereiften SOA mit einer großen Zahl an Services rechnen muss, ist ein Verzeichnis der Services unerlässlich. Ein solches Verzeichnis wird vielfach als *Registry* bezeichnet.

#### 3.1 Standards: UDDI und ebXML

Mit UDDI (Universal Description, Discovery and Integration) gibt es schon lange einen XML-basierten Standard, der ganz im Sinn von SOA einen auf Web-Services basierenden Zugriff auf eine Registry definiert, aber auch ein Informationsmodell für diese vorgibt. Diesem Informationsmodell merkt man die ursprüngliche Positionierung von UDDI als weltweites Serviceverzeichnis

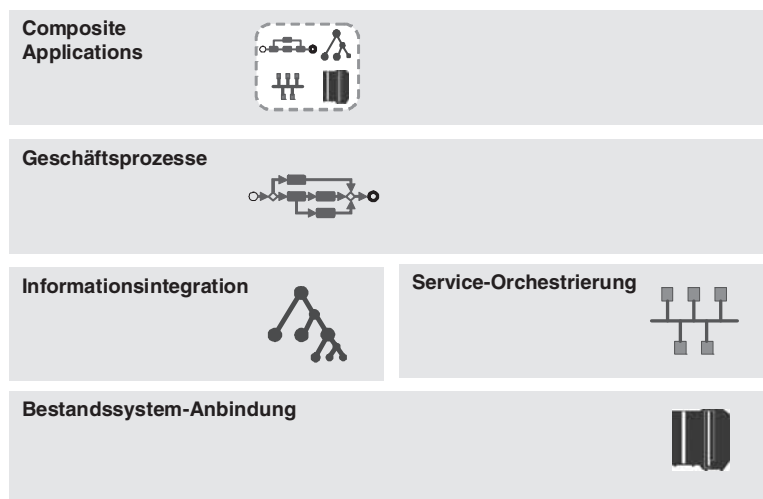


Abb. 1: Schichten in einer SOA

immer noch an: Jeder registrierte *Service* (der kein Web-Service sein muss) gehört zu genau einer *Business Entity* (die als der Dienstanbieter verstanden wird). Zu einem Service können beschreibende Texte hinzukommen. Ein Service kann mehrere *Binding Templates* haben, die verschiedenen Implementierungen des Service entsprechen. Diese *Binding Templates* können technische Information enthalten, wie z.B. eine Referenz auf die WSDL-Beschreibung der konkreten Implementierung, aber auch einen Verweis auf den Endpunkt, unter dem der Service in dieser Implementierung erreichbar ist. Weitere technische Information kann in Form von *tModel*-Referenzen angehängt werden. Ein *tModel* kann ganz unterschiedlichen Zwecken dienen – letztlich stellt es einen generischen Mechanismus dar, mit dem in UDDI alles modelliert werden kann, das kein Service, *Binding Template* oder *Business Entity* ist. Eine häufige Verwendung von *tModels* ist die einer Kategorisierung, die es ermöglicht, ein UDDI-Objekt beliebig zu klassifizieren. So kann es eine Kennzeichnung nach geografischen Kriterien geben, nach Branchen, nach Status usw. Diese Kategorisierung stellt eine wesentliche Basis für die Durchsuchbarkeit einer UDDI-basierten Registry dar. Leider sieht UDDI nur sehr indirekte Methoden vor, um die erlaubten/möglichen Werte einer solchen Kategorisierung festzulegen.

Allerdings lässt UDDI durch sein simplistisches Informationsmodell sehr viele Freiheiten für die Abbildung von Information. So kann die Referenz auf eine WSDL-Beschreibung für einen Service an unterschiedlichen Stellen eingefügt werden. Auch die Frage, wie ein Service klassifiziert wird, ist vollständig dem eintragenden Benutzer überlassen.

Offensichtlich ist die Suche nach einem passenden Service in einer solch permissiven Umgebung nicht immer einfach. Um die Offenheit des UDDI-Standards sinnvoll so zu begrenzen, dass einerseits eine gezielte Suche möglich ist und andererseits gewährleistet ist, dass wenigstens ein Mindestmaß an technischer Information, die in einer WSDL-Beschreibung vorhanden ist, auch über UDDI zugänglich gemacht wird, gibt es Empfehlungen [Colgrave & Januszewski 2004], wie eine WSDL in UDDI zu registrieren ist. Diese sehen z.B. vor, dass für jeden *Port Type* und jedes *Binding* aus der WSDL ein eigenes *tModel* erzeugt wird. Sie geben auch eine Kategorisierung vor, die z.B. das vom Web-Service unterstützte Protokoll beschreibt.

Prinzipiell können beliebige Objekte mittels *tModel* modelliert werden, wie z.B. BPEL-Prozesse [von Riegen & Trickovic 2004]. Eine solche Modellierung ist jedoch ohne Kenntnis der Abbildungsvorschrift schwer zu durchschauen und erzeugt meist ein unübersichtliches Geflecht von *tModels*, die aufeinander aufbauen (*tModels* werden zur Klassifikation von *tModels* benutzt).

UDDI hat keine eigene Anfragesprache, sondern vordefinierte Anfrageoperationen, eine für jeden Objekttyp: *find\_binding*, *find\_business*, *find\_service* usw. Was über diese nicht zugreifbar ist (wie z.B. die Beschreibungen von Services), steht zur Suche nicht zur Verfügung. Auch weiter gehende Operationen wie z.B. Aggregation (Zählen) gibt es nicht. UDDI hat den Vorteil, ein etablierter Standard zu sein, der von vielen Produkten unterstützt wird, aber auch den Nachteil, bei großen Datenmengen aus den

genannten Gründen schnell an die Grenzen der Benutzbarkeit zu stoßen.

Ein weiterer Standard, der eine Registry vorsieht, ist ebXML. Mit dem ebXML Registry Information Model (RIM) wird eine wesentlich ausgefeiltere Registry-Struktur definiert, die jedoch mit *Business*, *Service* und *Service Binding* direkte Korrespondenzen zu der *Business Entity*, dem Service und dem *Binding Template* aus UDDI hat. Darüber hinaus bietet ebXML RIM die Möglichkeit, externe und interne Klassifizierungsschemata zu definieren. Interne Klassifizierungsschemata entsprechen Taxonomien, die in ebXML ausmodelliert sind, während für externe Klassifizierungsschemata die erlaubten Werte nicht in ebXML definiert sind. Neben Klassifizierung erlaubt ebXML die Verbindung (*association*) zwischen zwei Objekten, die benutzt werden kann, um Abhängigkeit und Ähnliches zwischen Objekten zu modellieren. ebXML definiert seine Objekte objektorientiert als Spezialisierung der Klassen *RegistryObject* und *RegistryEntry*. Es ist erlaubt, Objekte vom Typ *RegistryEntry* hinzuzufügen, womit ebXML einen offenen Erweiterungsmechanismus bietet.

Gewissermaßen eine Synthese aus UDDI und ebXML stellt JAXR [JAXR] dar, eine Java-basierte Registry-Schnittstelle, die im Wesentlichen das RIM von ebXML realisiert, aber auch die Beziehung zu UDDI klar definiert und dessen Informationsgehalt abdeckt.

### 3.2 Anforderungen an eine Registry

Als Verzeichnisdienst muss eine Registry natürlich gewährleisten, dass ein gesuchter Service leicht gefunden werden kann. Je spezifischer das Informationsmodell der Registry ist, desto besser kann die Suche unterstützt werden. Die Verwendung lediglich zur Suche vergibt jedoch Möglichkeiten, die die in der Registry abgelegten Informationen eröffnen. Durch eine Verknüpfung der vorhandenen Informationen können weitere Anwendungsfelder eröffnet werden:

- *Assoziationen*: So können in der Registry auch Abhängigkeiten von Services untereinander gespeichert sein, die nicht nur zur Suche eines Service, sondern auch für dessen Benutzung und Weiterentwicklung relevant sind. Services, die von anderen Services benutzt werden, sollten nicht abgeschaltet oder ohne Rücksprache modifiziert werden. Die Information über eine solche Benutzungsrelation ist jedoch nicht per se in den Beschreibungsdaten der Services enthalten. Der Besitzer/Erzeuger eines Service kennt diese hingegen (meistens, dazu unten mehr) und kann sie in der Registry explizit modellieren.
- *Governance*: Auch Governance-Information kann in der Registry verwaltet werden, wie z.B. die Verantwortung für einen Service (z.B. die zuständige Person), der Status eines Service (*Testbetrieb*, *produktiver Betrieb*, *Abschaltung vorgesehen* etc.). Damit muss die Registry aber nicht mehr nur für eine programmatische Abfrage offen sein, sondern auch eine grafische Benutzeroberfläche haben, die auf die Bedürfnisse der verschiedenen Benutzergruppen (Administratoren, Entwickler etc.) angepasst ist.
- *Sicherheitsmodell*: UDDI erlaubt das Verändern einer Information nur für deren Besitzer (im Normalfall der Erzeuger), das Lesen hingegen für jeden. Durch die skizzierten Anwen-

dungsmöglichkeiten wird jedoch ein ausgefeilteres Sicherheitsmodell erforderlich.

- **Offenes Datenmodell:** Wie beschrieben, sollte eine Registry die Benutzungsbeziehungen zwischen Services abbilden können. Oft sind aber Services nicht die wesentlichen Komponenten für einen Administrator. Betrachtet man einen mit BPEL beschriebenen Geschäftsprozess, so wird dieser zwar oft als Service zur Verfügung gestellt, von den Betreibern aber vor allem als Geschäftsprozess gesehen. Daher ist es hilfreich, zwischen Geschäftsprozess und Service differenzieren zu können, ohne dass die Benutzungsrelationen verloren werden. Der Geschäftsprozess wird zum eigenen Objekt, das von einem Service benutzt wird (dem Service, der die Schnittstelle zum BPEL-Prozess darstellt) und das wiederum andere Services benutzt. Eine Registry muss daher vorsehen, das Datenmodell um neue Typen von Objekten erweitern zu können.

Vordefinierte Anfragemuster, wie UDDI sie kennt, können neue Objekttypen nicht berücksichtigen und schließen auch eine flexible Benutzung der in einer solchen Registry entstehenden, u.U. recht komplexen Objektnetze aus. Daher muss eine Registry freie Anfragen in einer Anfragesprache unterstützen. Da sowohl UDDI als auch JAXR auf XML basieren, ist *XQuery* eine nahe liegende Wahl für die Anfragesprache. Dies gilt umso mehr, wenn die Anfrage WSDL-Dokumente und andere XML-Dokumente im SOA-Umfeld mit einbeziehen kann, was dann der Fall ist, wenn diese in einem mit der Registry verbundenen Repository gespeichert werden. Trotzdem sollte das Registry auch eine UDDI-Schnittstelle anbieten, da dieser Standard von vielen Werkzeugen unterstützt wird.

#### 4 Das Repository

Zu einem Service gehört meistens ein WSDL-Dokument, das ihn beschreibt. Dieses wiederum verwendet XML-Schemadefinitionen, die in eigenen Dateien vorliegen können. Für Geschäftsprozesse kann eine BPEL-basierte Beschreibung vorhanden sein. Zu Services kann es Entwurfsdokumente, Benutzerdokumentation etc. geben.

Alle diese Dokumente, die sehr eng mit den Objekten verbunden sind, die in der Registry beschrieben sind, sollten »nahe

bei« der Registry gespeichert werden und von dieser ausgehend auffindbar sein, um die relevante Information nicht zu sehr zu zersplittern. Daher wird einer Registry oft ein Repository-Teil angegliedert, der das Abspeichern beliebiger Daten erlaubt. JAXR sieht das ausdrücklich vor. Hier bietet sich als Schnittstelle WebDAV [Goland et al. 1999] an, das als http-Erweiterung gut in eine Intra- und Internetlandschaft passt, aber auch (über eine WebDAV-fähige Benutzerschnittstelle wie z.B. den Windows Explorer) eine vertraute Dateisystemsicht auf die Daten erlaubt.

#### 5 CentraSite

Zur Illustration einer konkreten Realisierung eines solchen Registry/Repository-Systems wird das System CentraSite [CentraSite] kurz vorgestellt, das in gemeinsamer Entwicklungsarbeit von Fujitsu und Software AG auf der Basis des XML-Datenbanksystems Tamino [Schöning 2003] entwickelt wurde. Abbildung 2 zeigt eine Übersicht über CentraSite. CentraSite dient zwei wesentlichen Zielen:

- Zum einen stellt es das Herzstück der *crossvision suite* der Software AG dar, die Werkzeuge für die verschiedenen Aufgaben in einer SOA enthält (Information Integration, Business Process Modelling, Workflow, Enterprise Service Bus, Legacy Integration, ...). In dieser Nutzung werden in CentraSite nicht nur die jeweils erzeugten oder benutzten Web-Services registriert, sondern auch die jeweiligen Objekte der einzelnen Werkzeuge (Business Process, BPEL-Sequenz etc.) und Benutzungsbeziehungen zwischen diesen und den Services. So wird Beherrschbarkeit durch einen vollständigen Überblick über die SOA-Landschaft ermöglicht. Selbstverständlich wird CentraSite auch bei der Suche nach verwendbaren Services z.B. bei der Erstellung eines XPDL-basierten Geschäftsprozesses genutzt.
- Zum anderen wird CentraSite als Repository im SOA-Entwicklungsprozess eingesetzt. Hierzu gehören die Erzwingung von Designregeln durch CentraSite, die Verwaltung von Policies und die Abbildung der verschiedenen Entwurfsstadien bis hin zur Produktion und der Übergänge zwischen diesen.

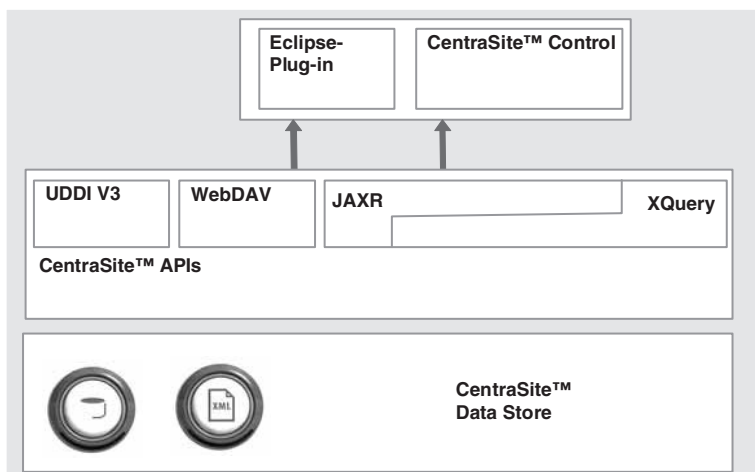


Abb. 2: Überblick über CentraSite

## 5.1 Schnittstellen

CentraSite benutzt eine gemeinsame Datenbank für den Registry- und den Repository-Anteil. Als Schnittstelle auf den Repository-Anteil wird WebDAV angeboten. Auf der Registry-Seite wird ein JAXR-konformes Datenmodell durch eine Menge von XML-Schemadefinitionen realisiert. In diesem Datenmodell sind die durch JAXR vorgegebenen Objekttypen (*Service*, *Organisation*, *User*, ...) vordefiniert. Besonders hervorzuheben sind hier:

- *Classification Scheme* und *Classification* – Ein Classification Scheme entspricht einer Taxonomie. Es kann extern sein, d.h., die zulässigen Werte werden durch externe, der Registry nicht bekannte Regeln festgelegt, oder intern, d.h., in der Registry sind die erlaubten Werte bekannt (JAXR nennt diese *Concept*). Werte können hierarchisch angeordnet sein, sodass etwa ein Classification Scheme *Welt* die Werte Afrika, Asien, Europa etc. hat, wobei z.B. Europa als untergeordnete Werte *Deutschland*, *Schweiz*, *Österreich* etc. haben kann. Diese Werte können Objekten durch eine (interne) Klassifikation zugeordnet werden. Eine externe Klassifikation ordnet einem Objekt ein Paar aus einem literalen Wert und einem externen Classification Scheme zu. Mit einer solchen Klassifikation können z.B. Services mit ihren Anwendungsbereichen (Finanzbuchhaltung, Logistik etc.) ausgezeichnet werden. Somit sind Services über die Klassifikation auffindbar (z.B. in der grafischen Benutzeroberfläche durch Navigation über die Taxonomie).
- *Association* – Dieser Objekttyp ermöglicht die Erzeugung von Beziehungen zwischen Objekten. Die erlaubten Assoziations-typen sind durch ein ebenfalls vordefiniertes Classification Scheme festgelegt. Dieses ist jedoch erweiterbar, sodass neue Assoziations-typen eingeführt werden können. Associations erlauben die Modellierung von Benutzungsbeziehungen und

ermöglichen so eine Navigation im Abhängigkeitsgraphen (s. Abb. 3). Unbenutzte Services oder auch zentrale Services lassen sich so identifizieren:

- *External Link* – Dieser Objekttyp verweist mittels URI auf ein Objekt außerhalb der Registry, also z.B. auf ein Dokument im Repository.
- *Audit log* – Jede Änderungsoperation an den Daten wird in einem Objekt dieses Typs protokolliert.

Da *Classification* und *Association* vollwertige Objekte sind, können sie wiederum zugeordnete *Associations* und *Classifications* haben. Für jedes Objekt kann die XML-Repräsentation abgefragt werden; diese ist auch das Exportformat.

Als weitere Schnittstelle wird XQuery unterstützt, das auf den XML-Repräsentationen der JAXR-Objekte arbeitet, aber auch auf XML-Objekte im Repository Zugriff hat, sodass eine Anfrage Registry und Repository kombinieren kann.

## 5.2 UDDI

Wie bereits ausgeführt, sollte eine Registry auf jeden Fall UDDI unterstützen, und zwar in der Version 2.0, weil viele Werkzeuge heute noch keine neuere Version akzeptieren, und in der neueren Version 3.0. Während die UDDI-v3.0-Spezifikation [Clement et al. 2004] vorgibt, wie eine Registry vorgehen muss, um gleichzeitig UDDI 2.0 und UDDI 3.0 zu unterstützen, ist diese Frage für eine gleichzeitige Unterstützung von JAXR und UDDI nicht vollständig in einer Spezifikation geklärt. CentraSite nimmt hier eine Abbildung vor, die die folgenden Eigenschaften erfüllt:

- Daten, die über UDDI v2 oder UDDI v3 gespeichert werden, sind natürlich über diese Protokolle auch wieder zugänglich, sind aber auch über JAXR erreichbar und interpretierbar und stellen dort vollwertige Objekte dar.

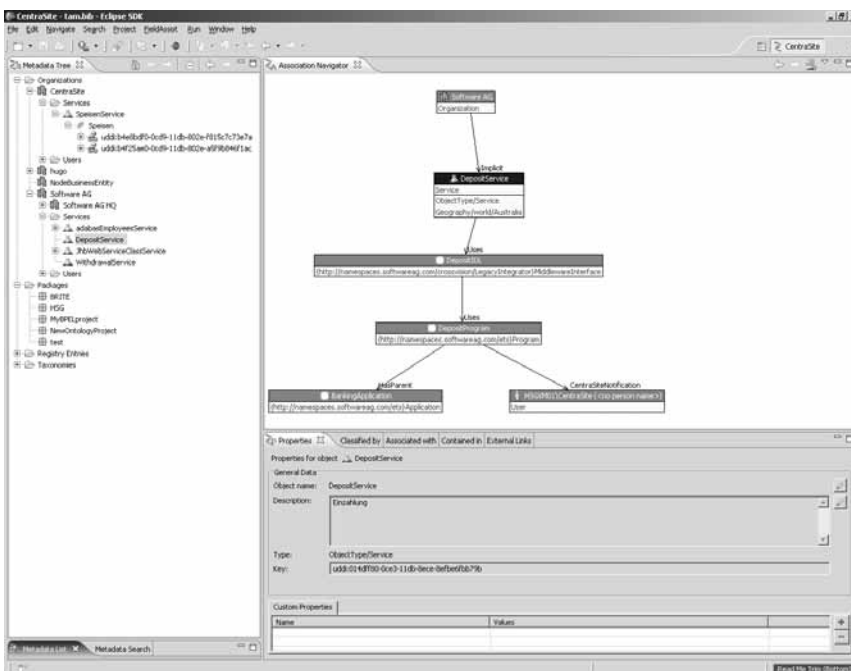


Abb. 3:  
Abhängigkeitsgraph  
im Eclipse-Plug-in

- Daten, die über JAXR gespeichert wurden, sind über UDDI v2 oder UDDI v3 zugänglich, wobei allerdings in manchen Fällen nicht alle Informationen verfügbar gemacht werden können, da es für diese in UDDI kein Äquivalent gibt. *Organisation*, *Service* und *Service Binding* in JAXR entsprechen dabei *Business Entity*, *Service* und *Binding Template* in UDDI. *Classification Schemes* und *Concepts* werden in UDDI auf *tModels* abgebildet.

Diese Abbildung besteht im Fall von *CentraSite* in der Transformation von einer XML-Repräsentation (der von JAXR) in eine andere (UDDI). Es verwundert daher nicht, dass an dieser Stelle XQuery zum Einsatz kommt, um die Transformation auszuführen.

Ein Problem ergibt sich hier aus dem Sicherheitsmodell von UDDI, das so in JAXR nicht existiert. *CentraSite* löst dies dadurch, dass das UDDI-Sicherheitsmodell erzwungen wird, solange keine expliziten anderen Sicherheitsdefinitionen vorgegeben werden.

### 5.3 Nutzung

Durch Unterstützung von UDDI kann *CentraSite* von Werkzeugen direkt angesprochen werden, die UDDI unterstützen, wie z.B. Microsoft InfoPath. Stärker integrierte Werkzeuge, wie z.B. der *crossvision Information Integrator* von Software AG, nutzen die JAXR-Schnittstelle, um komplexere Navigationen auf *CentraSite* ausführen zu können, oder XQuery, um mächtigere Suchen zu realisieren. Auch die GUIs (s. Abschnitt 5.6) nutzen XQuery in größerem Umfang.

### 5.4 Erweiterbarkeit des Datenmodells

Neue Objekttypen als Spezialisierung des JAXR-Typs *Registry-Entry* können über einen Wizard der grafischen Oberfläche definiert werden. Jeder Typ hat einen Namen und einen Namensraum, optional eine Beschreibung und ein zugeordnetes Icon.

Objekte eines Typs haben – durch die Vererbung vorgegeben – eine Reihe von Eigenschaften, z.B. einen Namen und eine Beschreibung. Ferner erlaubt JAXR sogenannte *Slots* für Objekte, die objektspezifisch weitere Eigenschaften aufnehmen können. Dieses Modell erweitert *CentraSite* dahingehend, dass auf Typebene weitere Eigenschaften definiert werden können, die alle Objekte dieses Typs dann besitzen. Somit gibt es für ein Objekt in *CentraSite* potenziell drei Sorten von Eigenschaften:

- Solche Eigenschaften, die von JAXR vorgegeben sind (z.B. *name*, *description*).
- Objektypspezifische Eigenschaften, also solche, die bei der Typdefinition in *CentraSite* spezifiziert wurden. Neben dem Datentyp (einer Auswahl aus den XML-Schema-Datentypen) kann angegeben werden, ob die Eigenschaft verpflichtend oder optional ist. Solche Eigenschaften können auch den in JAXR vordefinierten Typen hinzugefügt werden. Typische Anwendungen sind beispielsweise typspezifische Statusinformationen.
- Objektspezifische Eigenschaften, also solche, die keiner vorherigen Definition bedürfen. Diese werden oft verwendet, um bestimmten Objekten Kommentare oder Zusatzinformationen anzufügen.

Die beiden letzteren Typen entsprechen den Slots in JAXR. Neben einem Namen haben sie auch einen Namensraum, um Namenskonflikten vorzubeugen.

Die Definition von Eigenschaften auf Typebene wird auch benutzt, um Eingabeformulare für Objekte eines Typs zu generieren.

### 5.5 Aktives Verhalten

Die Nutzer eines Service, eines Businessprozesses oder eines anderen SOA-Objektes haben oft ein Interesse daran, bei Änderungen dieser Objekte aktiv informiert zu werden. Daher beinhaltet *CentraSite* die Möglichkeit, sich für bestimmte Objekte Benachrichtigungen senden zu lassen. Ein solcher Benachrichtigungswunsch wird über eine spezielle JAXR-Association zwischen dem Objekt und einem *User*-Objekt ausgedrückt. Geschieht eine Änderung, so nimmt *CentraSite* die entsprechende Benachrichtigung vor.

Eine allgemeine aktive Reaktion auf Änderung oder Neueingang eines Objektes erlauben die *CentraSite*-Trigger. Ganz SOA-gemäß kann man einen Web-Service angeben, der bei einem solchen Ereignis aufgerufen wird.

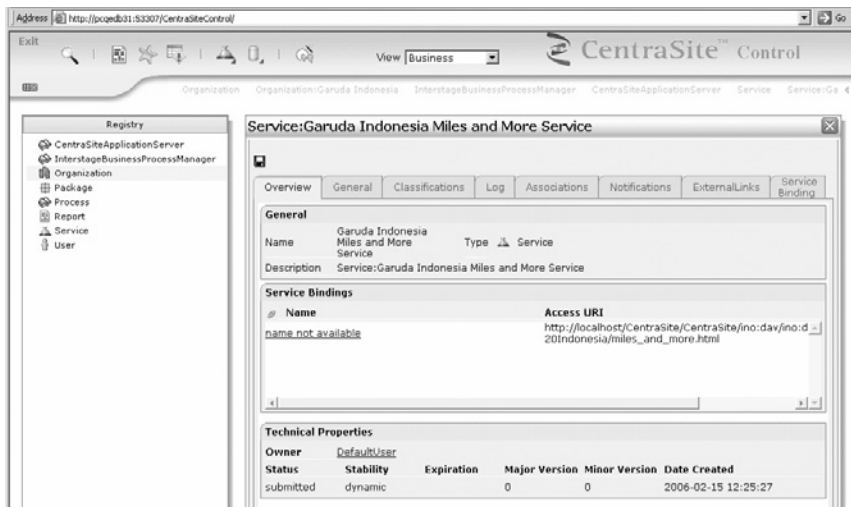
### 5.6 GUI

*CentraSite* liefert unter dem Namen *CentraSite Control* ein Ajax-basiertes Browser-Interface (s. Abb. 4), das verschiedene Ansichten auf Registry und Repository anbietet, je nachdem, ob eher der Administrator oder eher der Techniker es benutzt. Eine baumartige Navigation über die Registry erlaubt den Einstieg über Typen oder Klassifikationen. Daneben können Objekte natürlich auch über eine Suchfunktion aufgefunden werden. Von einem Objekt kann zu den damit verbundenen Objekten navigiert werden. Auch Änderungen an den Objekten sind in dieser Schnittstelle möglich.

Eine Funktion zum Import von WSDL-Dateien registriert die zugehörigen Services entsprechend der Empfehlung [Colgrave & Januszewski 2004], wobei auf JAXR-Ebene auch die WSDL-Datei über einen External Link abgebildet wird, der wiederum Nutzungsbeziehungen zu den importierten WSDL-Dateien und XML-Schemadateien enthält. Somit kann man in der Registry leicht nachvollziehen, welcher Web-Service von welcher XML-Schemadatei abhängt.

Für Entwickler bietet *CentraSite* darüber hinaus ein Eclipse-Plugin an, das im Wesentlichen dieselbe Funktionalität aufweist wie *CentraSite Control*, aber keine verschiedenen Sichten für verschiedene Benutzertypen anbietet.

Zur Erstellung von Auswertungen wurde ein Anschluss an das Open-Source-Werkzeug BIRT (Business Intelligence and Reporting Tools) [BIRT] geschaffen, ein Business-Intelligence-Werkzeug, das im Rahmen von *CentraSite* seine Reports direkt aus dem JAXR-Modell, aber auch aus XQuery-Ergebnissen generieren kann. So kann man beispielsweise Statistiken über den Wiederverwendungsgrad von Web-Services erzeugen.



**Abb. 4:**  
Das Browser-Interface  
CentraSite Control

## 5.7 Die XML-Datenbank dahinter

Wie bereits erwähnt, wurde CentraSite auf Basis des XML-Datenbanksystems Tamino entwickelt. Die bereits in Tamino vorhandene WebDAV-Funktionalität konnte für die Repository-Implementierung verwendet werden. Die Tatsache, dass JAXR und UDDI XML-basiert sind, ermöglichte eine einfache Realisierung: Die JAXR-Objekte werden direkt in ihrer XML-Repräsentation abgelegt, die UDDI-Sicht kann daraus mit XQuery einfach erzeugt werden. Einen besonderen Mehrwert zeigt die unterliegende XML-Infrastruktur bei der freien Anfrage über XQuery, denn nur mit einer XML-fähigen Anfragesprache kann eine einfache Verknüpfung von XML-basierten Dateien (WSDL-Dokumente, XML-Schemadefinitionen, BPEL-Dateien) mit den Registry-Objekten realisiert werden. Außerdem sind Themen wie Sicherung/Wiederherstellung, Hochverfügbarkeit und andere Datenbankfunktionen automatisch abgedeckt.

Allerdings wurde Wert darauf gelegt, CentraSite im Wesentlichen administrationsfrei betreiben zu können – der Start der Datenbank erfolgt automatisch, Sicherungen geschehen im laufenden Betrieb und können durch entsprechende Konfiguration regelmäßig automatisch durchgeführt werden.

## 6 Zusammenfassung und Ausblick

In einer serviceorientierten Architektur gibt es Services auf verschiedenen Ebenen, mit möglicherweise komplexen Abhängigkeiten. Die mit UDDI gegebene Registry-Schnittstelle reicht nicht aus, um diese Abhängigkeiten zu verwalten. Im System CentraSite wird daher neben der UDDI-Schnittstelle eine JAXR-Schnittstelle angeboten, die ein flexibles und erweiterbares Datenmodell unterstützt. Integriert ist außerdem ein Repository, das nicht nur eine direkte Verknüpfung mit der Registry hat, sondern auch in Abfragen über XQuery kombinierbar ist.

Eine Registry in der hier beschriebenen Form deckt zunächst die statischen Aspekte einer SOA-Landschaft ab. Hier werden Informationen über Schnittstellen, Verantwortlichkeiten, Richtlinien

(Policies) usw. abgelegt, aber zunächst keine Daten über das dynamische Verhalten der SOA-Komponenten zur Laufzeit, also etwa Antwortzeiten und Verfügbarkeitsstatistiken, oder dynamische Benutzungsbeziehungen, die durch Auswahl der Services zur Laufzeit entstehen können. Für einen Administrator sind jedoch statische und dynamische Daten von Interesse. Ziel muss es daher sein, diese Daten auch in CentraSite verfügbar zu machen. Nun gibt es auf dem Markt einige Werkzeuge, die sich auf die Erhebung solcher Daten spezialisiert haben. Über die CentraSite Community arbeitet die Software AG mit den Herstellern solcher Werkzeuge zusammen, um eine Integration dynamischer Daten zu ermöglichen.

Umgekehrt beziehen aber solche Werkzeuge ihre Daten auch aus CentraSite, z.B. werden die an einen Service gebundenen Richtlinien von CentraSite verwaltet und über den Standard WS-Policy-Attachment [Bajaj et al. 2006] für den Zugriff verfügbar gemacht, sodass darauf spezialisierte Produkte die Einhaltung der Richtlinien prüfen und erzwingen können.

Darüber hinaus bietet die Software AG unter dem Namen *cross-vision* eine Familie von Produkten an, die CentraSite als zentrales Repository benutzen und beispielsweise Benutzungsbeziehungen automatisch in CentraSite registrieren.

Aus den bisherigen Ausführungen ergibt sich, dass ein kombiniertes Registry/Repository während verschiedener Phasen in der Softwareentwicklung sinnvolle Dienste leisten kann – bei der Entwicklung zur Suche wiederverwendbarer Komponenten, zur Laufzeit für die Verwaltung von Policies etc. Daher ist es nahe liegend, ein Registry/Repository um die Unterstützung eines SOA-Lebenszyklus zu erweitern (der über eine bloße Versionierung von Objekten hinausgeht). Eine solche Erweiterung existiert auch für CentraSite – aus Platzgründen kann sie hier nicht detailliert vorgestellt werden.

Alle diese Erweiterungen lassen sich dank der unterliegenden XML-Technologie einfach und schnell in CentraSite integrieren.

## Literatur

- [Bajaj et al. 2006] *Bajaj, S. et al.*: Web Services Policy 1.2 – Attachment (WS-PolicyAttachment), W3C Member Submission 25 April 2006, [www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/](http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/).
- [BIRT] *BIRT: Business Intelligence and Reporting Tools*, [www.eclipse.org/birt/phenix/](http://www.eclipse.org/birt/phenix/).
- [CentraSite] Real-World SOA: Definition, Implementation and Use of SOA with CentraSite™, White Paper, [http://soaworks.com/pdf/White\\_Paper\\_CentraSite.pdf](http://soaworks.com/pdf/White_Paper_CentraSite.pdf).
- [Channabasavaiah et al. 2003] *Channabasavaiah, K.; Holley, K.; Tuggle, E.*: Migrating to a service-oriented architecture, Part 1, IBM developerWorks, 2003, <http://www-128.ibm.com/developerworks/library/ws-migratesoa/>.
- [Christensen et al. 2001] *Christensen, E. et al.*: Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, [www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl).
- [Clement et al. 2004] *Clement, L. et al.*: UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, OASIS 2004, <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>.
- [Colgrave & Januszewski 2004] *Colgrave, J.; Januszewski, K.*: Using WSDL in a UDDI Registry, Version 2.0.2, Technical Note, OASIS, 2004, [www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm](http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm).
- [CORBA] Catalog of OMG CORBA®/IIOP® Specifications, [www.omg.org/technology/documents/corba\\_spec\\_catalog.htm](http://www.omg.org/technology/documents/corba_spec_catalog.htm).
- [Erl 2005] *Erl, T.*: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, 2005.
- [Goland et al. 1999] *Goland, Y. et al.*: HTTP Extensions for Distributed Authoring – WEBDAV, Request for Comments: 2518, 1999, [www.webdav.org/specs/rfc2518.pdf](http://www.webdav.org/specs/rfc2518.pdf).
- [Gudgin et al. 2003] *Gudgin, M. et al. (Hrsg.)*: SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation 24 June 2003, [www.w3.org/TR/2003/REC-soap12-part1-20030624/](http://www.w3.org/TR/2003/REC-soap12-part1-20030624/).
- [JAXR] Java API for XML Registries (JAXR), <http://java.sun.com/webservices/jaxr/>.
- [MacKenzie et al. 2006] *MacKenzie, C. M. et al. (Hrsg.)*: Reference Model for Service Oriented Architecture 1.0, Public Review Draft 2, 31 May 2006, [www.oasis-open.org/committees/download.php/18487/wd-soa-rm-pr2.doc](http://www.oasis-open.org/committees/download.php/18487/wd-soa-rm-pr2.doc).
- [OWL-S] *The OWL Services Coalition OWL-S: Semantic Markup for Web Services*, [www.daml.org/services/owl-s/1.0/owl-s.html](http://www.daml.org/services/owl-s/1.0/owl-s.html).
- [Schöning 2003] *Schöning, H.*: Tamino – A Database System Combining Text Retrieval and XML. In: Blanken, H. et al.: Intelligent Search on XML Data. Springer-Verlag, 2003.
- [Schulte & Natis 1996] *Schulte, R. W.; Natis, Y. V.*: »Service Oriented« Architectures, Part 1, Gartner SPA-00-7425, 1996, [www.gartner.com/DisplayDocument?doc\\_cd=29201](http://www.gartner.com/DisplayDocument?doc_cd=29201).
- [von Riegen & Trickovic 2004] *von Riegen, C.; Trickovic, I.*: Using BPEL4WS in a UDDI registry, Technical Note, OASIS, 2004, [www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm](http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm).



### Harald Schöning

studierte Informatik an der Universität Kaiserslautern. Seit seiner Promotion 1993 ist er bei der Software AG beschäftigt, anfangs als Entwickler und Projektleiter für das Hochleistungsdatenbanksystem ADABAS, derzeit als Architekt des XML-Datenbanksystems Tamino, dem Marktführer unter den rein XML-basierten Datenbanksystemen. Lehraufträge an verschiedenen deutschen Universitäten und regelmäßige Seminare zum Thema »XML und Datenbanken« für die Deutsche Informatik-Akademie belegen sein Engagement in der Lehre.

Dr.-Ing. Harald Schöning  
Software AG  
Uhlandstr. 12  
64297 Darmstadt  
[Harald.Schoening@softwareag.com](mailto:Harald.Schoening@softwareag.com)  
[www.softwareag.com](http://www.softwareag.com)