MODELLBASIERTE REPOSITORY-INTEGRATION: PRAXISBERICHT DER DEUTSCHEN TELEKOM

Bei einer „Service Oriented Architecture" (SOA) werden Informationen rund um Services und Anwendungen auf unterschiedlichen Abstraktionsebenen erfasst. Für die Verwaltung der Informationen werden heterogene Repository-Produkte verschiedener Hersteller verwendet. Um diese verteilten Informationen zueinander in Relation setzen und sie synchronisieren zu können, benötigt man eine Repository-Integration. Dieser Artikel beschreibt eine Modell-Repository-Integrationsarchitektur der T-Mobile (jetzt Deutsche Telekom), die im Rahmen einer großen SOA-Infrastruktur entwickelt wurde.

die autoren

Marcus Greuel
(E-Mail: Marcus.Greuel@external.telekom.de)
ist freiberuflicher Softwarearchitekt. In den letzten fünf Jahren hat er die Architektur und Realisierung der Repository-Integration innerhalb der Deutschen Telekom maßgeblich vorangetrieben.

Frank Kimmlingen
(E-Mail: Frank.Kimmlingen@telekom.de)
ist Softwareentwickler, Architekt und Projektleiter im Framework- und Enterprise-Integration-Umfeld und bringt seit etwa 2003 die modellgetriebene Softwareentwicklung in verschiedenen SOA-Umgebungen der T-Mobile voran.

Ohne eine einheitliche Repository-Integrationsarchitektur wird man schnell mit der aus der *Enterprise Application Integration (EAI)* bekannten *n*m*-Problematik bei Punkt-zu-Punkt-Verbindungen konfrontiert. Durch die Verwendung einer einheitlichen werkzeuggestützten Architektur wird verhindert, dass grundlegende Mechanismen für jedes Integrationsprojekt redundant entwickelt werden.

Das Modell-Repository-Integrations-Framework

Im Rahmen der Entwicklung der internationalen SOA-Infrastruktur *SOA Backplane (SOABP)* entstand das SOA-Modell-Repository *CEISeR (Central Enterprise Integration Service Repository)*. Dieses Modell-Repository ist in der Lage, große Mengen von historisierten Daten transaktional zu verwalten und zu integrieren.

Den von der Anwendungsdomäne unabhängigen Teil des Modell-Repositorys bezeichnen wir im Folgenden als *Modell Repository Integrations Framework (MRIF)*. Dieses Framework dient als Plattform für die aus UML2-Klassenmodellen generierten Domänenklassen. Eine Anwendungsdomäne des MRIF ist CEISeR, das seit über vier Jahren erfolgreich historisierte Modelldaten in Form von über 1.400 Service-Definitionen und über 2.500 Kommunikationsbeziehungen verwaltet. Mit CEISeR wurden folgende Themen bereits erfolgreich in der Praxis umsetzt:

- Qualitätsmanagement und Governance von Modellen
- Modelltransformation und Konsistenzprüfung
- Management großer Modelle

In diesem Artikel erläutern wir nicht die einzelnen aufgezählten Punkte (vgl. hierzu [Sen09], [Kim09]), sondern die darüber hinausgehenden Techniken für die Integration heterogener Repository-Produkte auf der Basis von Modellen und deren Metamodellen mit den in CEISeR entwickelten Basis-Technologien.

Anforderung an das MRIF

Dem Thema „Repository-Integration" messen viele Unternehmen eine hohe Bedeutung zu, weil Metamodelle und die dazugehörigen Modelle an unterschiedlichen Stellen verwaltet werden. Ähnliche Daten, wie wir sie in CEISeR verwalten, werden unter anderen Verantwortlichkeiten und anderen Gesichtspunkten mit anderen für diesen Zweck geeigneten Repositorys gepflegt. Beispiele für solche Repository-Produkte sind „ARIS" (vgl. [IDS]), „planningIT" (vgl. [Alf]) und „Centrasite" (vgl. [Sof]).

Das ARIS-Repository bietet eine Sicht auf die fachlichen Prozesse und die verwendeten Daten, in deren Kontext Services genutzt werden. Bei der Deutschen Telekom wird das planningIT-Repository genutzt, um konzernweit Informationen der IT-Applikationen zu verwalten. In Centrasite hingegen wird zum Beispiel ein *EAM*-Modell (*Enterprise Architecture Management*) verwaltet, wo Services und Applikationen auf einem abstrakteren Level als in CEISeR fachlogisch betrachtet werden. Zusätzlich zu den Informationen aus CEISeR werden beispielsweise Informationsflüsse zwischen den Applikationen erfasst. Mit den Daten aus CEISeR wird die Konfiguration für die Laufzeitkomponenten der SOABP generiert (vgl. [Sen08]). Die Daten in Centrasite nutzen die Enterprise-Architekten für Governance-Themen, wie z. B. das Service-Portfolio-Management. Um eine Verlinkung zwischen der abstrakten Betrachtung aus dem EAM und der Laufzeit-Perspektive von CEISeR abbilden zu können, werden Mechanismen für die Integration der verschiedenen Repositorys benötigt. Die Anforderung besteht darin, strukturierte und stark vernetzte Informationen zu integrieren. Diese Informationen steuern unterschiedliche Quellen unabhängig voneinander zu unterschiedlichen Zeiten bei. Ziel ist es, zu einem konsistenten Gesamtzustand zu gelangen. Da in großen Organisationen wie der Deutschen Telekom eine heterogene Werkzeug- und Prozess-Landschaft sowie unterschiedliche Verantwortlichkeiten für bestimmte Daten existieren, ist weder ein gemeinsamer Modellierungseditor noch ein gemeinsames Repository

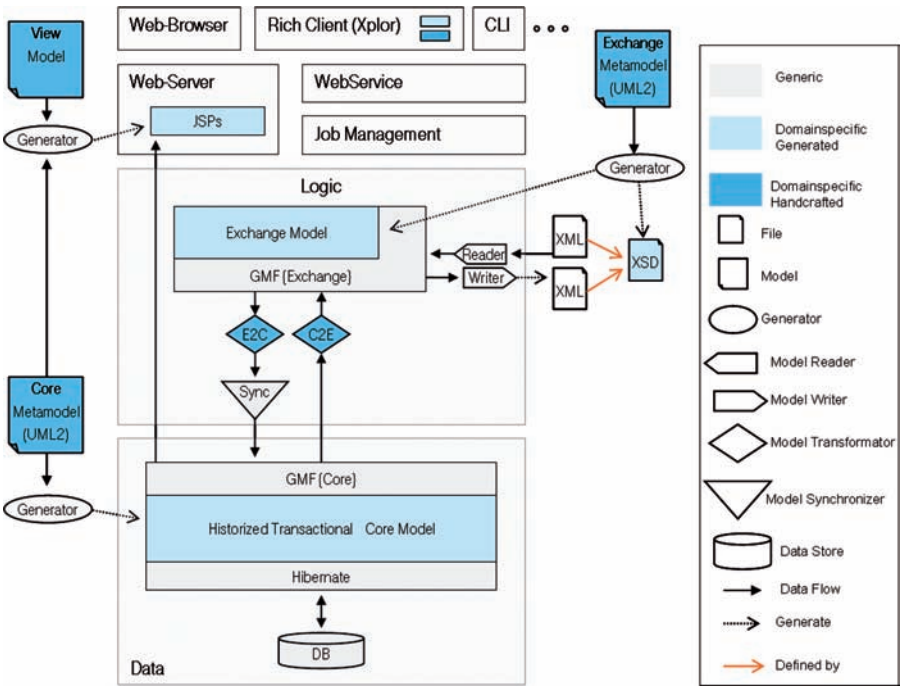

Abb. 1: Die Repository-Architektur.

durchsetzbar und sinnvoll. Um diese Anforderungen zu erfüllen, wurde im Rahmen der SOABP für CEISeR ein MRIF entwickelt.

Grundlagen der modellbasierten Repository-Integration

Bei der Entwicklung von CEISeR haben wir Wert darauf gelegt, möglichst viele Anteile des Modell-Repositorys aus dem auf UML basierenden CEISeR-Metamodell generieren zu können. Basierend auf Erfahrungen, die wir in der Vorgänger-SOA-Infrastruktur von SOABP gemacht haben, wurde die UML jedoch nicht als Basis eines einheitlichen Modellierungswerkzeugs gewählt.

XML-Dateien als Schnittstelle für den Datenaustausch

Um Daten in das Modell-Repository importieren oder exportieren zu können, werden XML-Dateien gelesen bzw. geschrieben. Das Format der verschiedenen XML-Dateien wird durch XML-Schema-Dateien beschrieben, die aus dem so genannten Exchange-Metamodell generiert werden (siehe Abbildung 1). Diese XML-Dateien beschreiben in einer menschenlesbaren Form Modelle, die dem Exchange-Metamodell entsprechen. Um die Lesbarkeit zu gewährleisten, werden IDs und Referenzen von Objekten in einer an Programmiersprachen wie Java angelehnten Form abgebildet.

Java und Subversion als Vorbild

Einige fundamentale Designentscheidungen des MRIF haben wir den Konzepten der verteilten Softwareentwicklung mit Java und dem Versionsverwaltungssystem Subversion entnommen.

Programmiersprachen wie Java implementieren wohl verstandene Mechanismen zum Integrieren von verteilten, stark vernetzten Informationen zu einem konsistenten Gesamtbild. Im MRIF werden symbolische Verknüpfungen anstelle von technischen *Universally Unique Identifier (UUIDs)* verwendet. Durch die Verwendung von symbolischen Verknüpfungen ist die Lesbarkeit der XML-Dateien gegeben. Die Darstellung der Verknüpfungen ist unabhängig von konkreten Modellierungswerkzeugen.

Das Konzept der *Namespaces* wird zur Bildung von verteilten Namensräumen und zu Entkoppelung der Vergabe der symbolischen Namen verwendet. IDs und Referenzen sind zusammengesetzt aus dem Namensraum und dem Namen des Objekts und werden *Fully Qualified Name (FQN)* genannt (siehe Kasten 1). Basierend auf dem Namensraum-Konzept können Berechtigungen rekursiv vergeben werden. Auf dem *Root*-Namensraum darf initial nur der *Superuser* schreiben. Ein Benutzer, der auf einem Namensraum Schreibrechte besitzt, darf rekursiv in allen Unternamensräumen schreiben und an andere Benutzern Schreibrechte vergeben.

Bei der Softwareentwicklung mit Java können einzelne .java-Dateien unabhängig voneinander von unterschiedlichen Entwicklern geschrieben werden. Die Referenzen der verschiedenen Java-Klassen in den Dateien werden über symbolische Verknüpfungen (import [namespace].[Klassenname];) aufgelöst. Erst der Compiler löst die import-Befehle in den .java-Dateien auf und liefert bei fehlenden .java-Dateien entsprechende Fehlermeldungen. Im MRIF erfolgt die Überprüfung der Konsistenz

- Die Aufwände für die Trennung zwischen internem und externem Datenformat lohnen sich und sind vom Aufwand akzeptabel.
- Modelladapter auf der Basis einer einheitlichen Modell- und Metamodell-Fassade sind der Ausweg aus der *n*m*-EAI-Problematik bezogen auf Modell-Repositorys.
- Eine einheitliche Modell- und Metamodell-Fassade ermöglicht die Wiederverwendung von mächtigen generischen Werkzeugen.
- Die Modellfragmente (Kompositionsbeziehung) als kleinste Granularität für den Datenaustausch und die Persistenz haben sich als querschnittliches Konzept bewährt.
- Der flexible MDSD-Ansatz macht den modellbasierten Ansatz für die Repository-Integration erst praktikabel.
- Der Einsatz von IDs und Referenzen auf Basis von symbolischen Links hat sich bisher als guter Ansatz erwiesen. In weiterführenden Integrationsszenarien muss die Definition der ID flexibler gestaltet werden, um z. B. UUIDs von anderen Repositorys besser zuordnen zu können.
- Im Umfeld von MDSD und der modellbasierten Entwicklung setzen sich EMF (vgl. [Ste09]) und darauf aufbauende Frameworks als eine Art Industriestandard durch. Da das GMF nicht direkt kompatibel zu EMF ist, gibt es immer wieder kontroverse Diskussionen. Um in Zukunft einfacher mit auf EMF basierenden Modellen umgehen zu können, gibt es im MRIF die Umsetzung eines EMF-Modelladapters, der in der Lage ist, EMF-Ressourcen zu verarbeiten.

Kasten 1: Erfahrungen mit und Konsequenzen aus der Lösung.

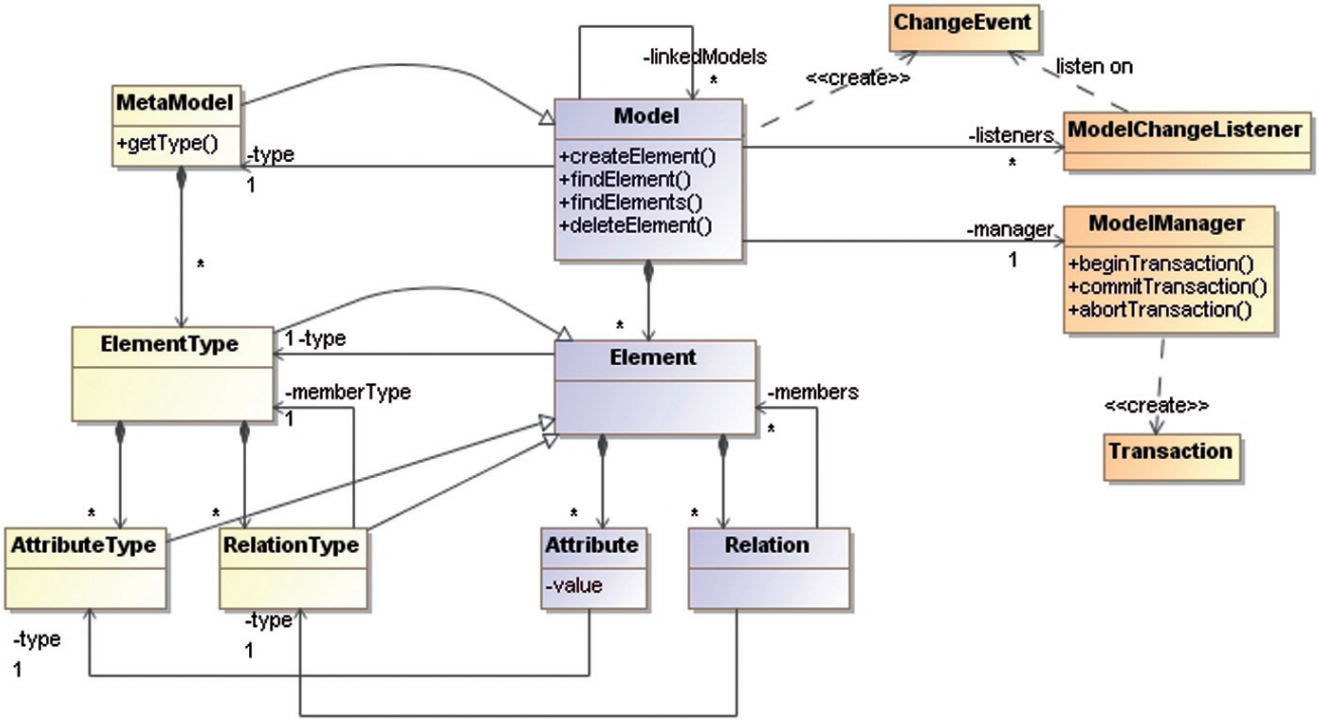

Abb. 2: Das Generic Modeling Framework (GMF).

ebenfalls unabhängig vom Zeitpunkt des Erfassens. Erst wenn alle Beteiligten ihre Informationen eines Anwendungsfalls beigesteuert haben, muss ein für diesen Anwendungsfall konsistentes Gesamtbild vorhanden sein. Insbesondere das Auflösen der symbolischen Verknüpfungen ist erst dann fehlerfrei, wenn alle ihre Informationen beigesteuert haben. Fehlermeldungen werden erzeugt, wenn nicht alle Informationen vorhanden sind (vgl. [Kim09]).

Versionsverwaltungssysteme wie Subversion bringen Konzepte wie Transaktionen, Historisierung und Branching mit. Übertragen auf Modelle bedeutet das, dass Änderungen über das MRIF über die Vergabe von Transaktionsnummern erfolgen. Um Konflikte bei konkurrierenden Änderungen erkennen zu können, wurde auf der Basis der Transaktionsnummer ein optimistisches Lock-Verfahren implementiert. Änderungen, die über das MRIF gemacht werden, sind nachvollziehbar, und es ist möglich, auf historische Stände von Modellen zuzugreifen. Differenzen zwischen einzelnen Transaktionen können auf Objektebene ausgewertet werden.

Trennung von externem und internem Datenformat

Eine Designentscheidung, die bei CEISeR früh getroffen wurde, ist die Unterscheidung zwischen dem externen und dem internen Datenformat (siehe Abbildung 1). Durch diese Trennung ist es möglich, die externe Schnittstelle über mehrere Releases stabil zu halten. Das externe Format (*Exchange Model*) ist hinsichtlich des Austauschs und der Kompatibilität optimiert, wohingegen das interne Format (*Core Model*) Performance und andere technische Aspekte berücksichtigt. Im MRIF werden aus Metamodellen die domänenspezifischen Klassen generiert (hellblaue Bereiche in Abbildung 1). Neben den Klassen werden für das *Exchange Modell XML Schema (XSD)* Dateien generiert, die die Strukturen der XML-Dateien für den Datenaustausch definieren (siehe Kasten 1).

Metamodell- und modellbasiert

Innerhalb des MRIF werden alle Informationen als Modelle mit entsprechenden Metamodellen behandelt. Neben den XSD-Dateien werden ebenfalls alle Modellimplementierungen generiert (*Exchange Model* und *Core Model*). Für die Generierung wird das *openArchitectureWare Framework* (*OAW*), ein MDSD-Generator-Framework (vgl. [Sta07]), verwendet. Insbesondere sind die transaktionalen und historisierten Repositorys spezielle Modellimplementierungen und werden somit ebenfalls automatisch aus dem Metamodell erstellt. Eine weitere wichtige Designentscheidung ist die Aufteilung der Modelle in Modellfragmente, die unteilbare Teilbäume des Modells sind. Ein Modellfragment ist die kleinste Granularität für den Datenaustausch und für die Persistenz. Das Konzept entspricht der Kompositionsassoziation in UML2 zwischen Klassen.

Metamodellbasierte generische Modellfassade für alle beteiligten Modelle

Um die modellbasierten Informationen einheitlich verwalten zu können, wurde das *Generic Modeling Framework (GMF)* entwickelt (siehe Abbildung 2). Die Konzepte des GMF sind mit den EMF-Konzepten (vgl. [Ste09]) vergleichbar. Mit GMF ist nicht das gleich abgekürzte „Eclipse Graphical Modeling Framework" gemeint, sondern die im Rahmen von SOABP entwickelte Framework-Lösung.

Durch GMF wird eine einheitliche generische Modell- und Metamodell-Fassade bereitgestellt (siehe Kasten 1). Die Fassade enthält:

- eine einheitliche *CRUD*-Schnittstelle (*Create Read Update Delete*) für Modellobjekte
- eine einheitliche Schnittstelle für die Navigation durch die Elemente der Modelle und die Elemente der dazugehörigen Metamodelle

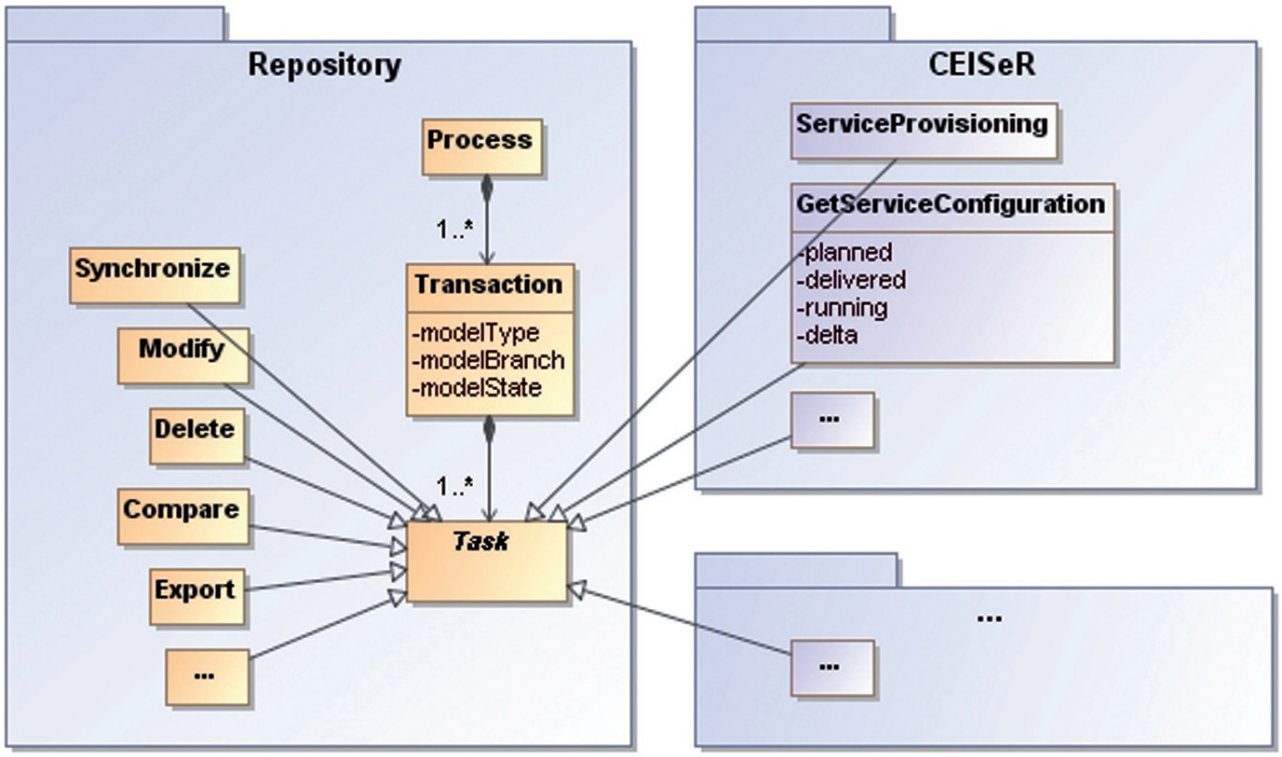

Abb. 3: Das Prozessmodell.

- eine einheitliche Transaktionsschnittstelle

Werkzeuge, die auf der GMF-Modellfassade basieren

Auf den Schnittstellen von GMF gibt es einige Werkzeuge, die für die Repository-Integration verwendet werden (siehe Abbildung 1). Für die XML-Dateien des Datenaustauschs existieren Serialisierer und Deserialisierer (*Model Reader, Model Writer*), die das XSD-Format des Exchange-Modells verarbeiten können. Die Unterstützung anderer Formate, zum Beispiel eine Implementierung für Ressourcen des EMF (vgl. [Ste09]), sind denkbar. *Model Synchronizer* sind für die Synchronisierung zwischen Modellteilen verantwortlich. Ein Modell-Transformations-Framework bildet die Grundlage zur Erstellung Java-basierter, typsicherer Modell-zu-Modell-Transformationen (*Model Transformator*). Weitere Werkzeuge (nicht in Abbildung 1 dargestellt) erledigen die Validation von Modellteilen (*Model Validator*) und einen Zwei- und Dreiwege-Vergleich, dessen Ergebnis wieder ein Modell ist (*Model Diff*).

Repository-Server

Nach der Beschreibung der Grundlagen der modellbasierten Repository-Integration gehen wir auf darauf aufbauende Server-Komponenten des Repository-Servers ein. (siehe Abbildung 1).

Prozessmodell zur Programmierung des Servers

Das Prozessmodell dient der Programmierung des Repository-Servers. Der Anwender des Repository-Servers konfiguriert ein Prozessmodell durch so genannte Manifest-XML-Konfigurationsdateien. Durch das Prozessmodell wird die transaktionsgesteuerte Programmierschnittstelle des Servers gebildet (siehe auch Abbildung 3):

- Ein *Prozess* fasst eine Menge von Transaktionen und deren Abhängigkeiten untereinander zusammen.
- *Transaktionen* definieren das Modell, den *Branch* und den Zustand, auf denen die eingebetteten Tasks arbeiten, und bilden die Klammer für die Persistierung der Änderungen durch diese Tasks.
- *ModelLocators* definieren Teilmengen eines Modells anhand der Datentypen, Daten oder zeitlicher Aspekte.
- *Tasks* sind die Funktionsbausteine und können *ModelLocator* verwenden, um die Elemente zu identifizieren, auf denen sie arbeiten. Beispiele für *Tasks* sind „Exportiere Elemente in XML", „Synchronisiere XML-Informationen in Repository" oder „Versorge SOABP mit Service-Konfiguration".

Job-Framework

Ein Prozess wird vom Job-Framework ausgeführt (siehe Abbildung 4). Durch die Job-Semantik wird die Asynchronität bereit gestellt, die notwendig ist, da die Ausführung der Prozesse oft mehrere Minuten dauern kann. Die Eingabe für die Ausführung eines Prozesses besteht aus der Prozessbeschreibung (Manifest-XML-Datei) und einer Menge von Eingabedateien, die prozessiert werden sollen. Die Ausgabe besteht aus dem Ausführungsprotokoll-Modell und einer Menge von Ausgabe- und Log-Dateien. Jobs können sowohl manuell als auch zeitgesteuert über den Repository-Web-Service und das *Command Line Interface (CLI)* eingestellt werden.

Web-Service-Interface

Der Repository-Server bietet einen generischen Web-Service für das Job-Framework an. Dieser bietet Services an, um einen Job einzustellen und dessen Status zu erfragen. Wenn der Jobstatus-Service als Antwort „erledigt" zurückliefert, kann über einen weiteren Service das Job-Ergebnis abgeholt werden.

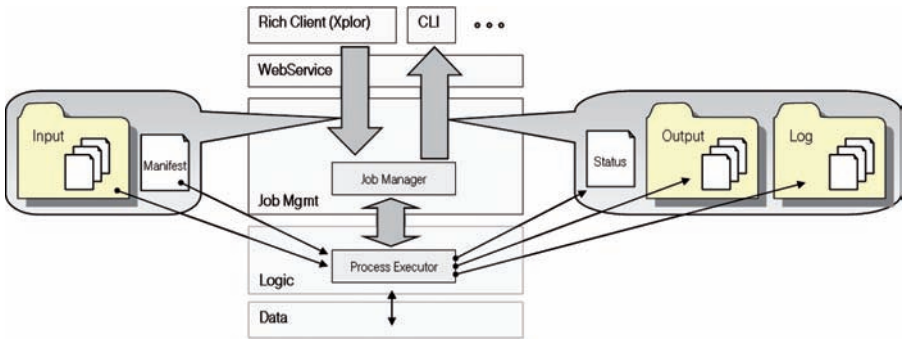

Abb. 4: Das Job-Framework.

Klienten des Repository-Servers

Wir beschreiben nun die unterschiedlichen Zugangswege der Klienten zu den Daten auf dem Repository-Server.

Dynamischer HTML-Report

Der einfachste Zugang zu den Daten des Repository-Servers erfolgt über eine HTML-Seite. Die HTML-Seite bietet eine einfache Navigation und Ansicht des Repository-Modells. Über diesen Zugang können sowohl der letzte Stand als auch beliebige historische Stände über die jeweilige Transaktionsnummer betrachtet werden. Die HTML-Seite ist ein rein lesender dynamischer Report, der durch *Java Server Pages (JSP)* – basierend auf dem Metamodell des Repositorys, verwoben mit einem UI-Modell – generiert wird (siehe Abbildung 1).

Ein auf Eclipse basierender Rich Client

Der Hauptzugang zu den Daten und den Prozessmodellen des Repository-Servers ist der auf Eclipse RCP (vgl. [Ecl-a]) basierende „Xplor" (siehe Abbildung 1). Die einheitliche Schnittstelle, auf der dieser arbeitet, ist die Modellfassade GMF. Zur Laufzeit interpretiert Xplor das Metamodell und metamodellspezifische Konfigurationsmodelle. Er bietet Baumansichten von beliebigen Modellen. Querschnittliche Ansichten der Modelle werden über Suchen ermöglicht. Sowohl die Baumansichten als auch die Suchen sind für jedes Metamodell spezifisch über die Konfigurationsmodelle einstellbar.

Neben dem Browsen von Modellen ist mit Xplor auch das Editieren von Modellen möglich. In Editoren werden Vorschlagslisten für sinnvolle Werte („Content Assists") für das Befüllen von Referenzen zwischen Modellelementen angeboten. Dies ist – wie in der Eclipse-Java-Entwicklungsumgebung – über den Tastatur-Shortcut „<Strg>+<Space>" möglich. Durch Modell-Validierung wird das Editieren vor dem eigentlichen Abspeichern im Repository-Server abgesichert. Das gleichzeitige Verbinden zu mehreren Repository-Servern über das Webservice-Job-Interface ermöglicht die Integration verschiedener Repositorys, basierend auf Benutzungsschnittstellen. Lokale, voll funktionstüchtige Repositorys werden zu Test- und Entwicklungszwecken unterstützt. Verteiltes Arbeiten wird ermöglicht, indem Teamfunktionalität für das Arbeiten mit dem Repository-Server angeboten wird. Die Teamfunktionalität besteht aus dem *Checkout* und *Commit* von Modellfragmenten über die *ModelLocator*, einem Dreiwege-*Diff* auf den editierten Modellfragmenten und einer *Merge*-Funktionalität für Modellfragmente.

Anbindung von externen Repositorys über Modelladapter

Beliebige externe Repositorys können durch das MRIF integriert werden, indem auf der Basis der jeweiligen Programmierschnittstelle des Herstellers ein Modelladapter implementiert wird (siehe Kasten 1).

Bei der Anbindung des auf CentraSite basierenden Repositorys für das EAM-Modell wurde z. B. ein JAXR-Modell-

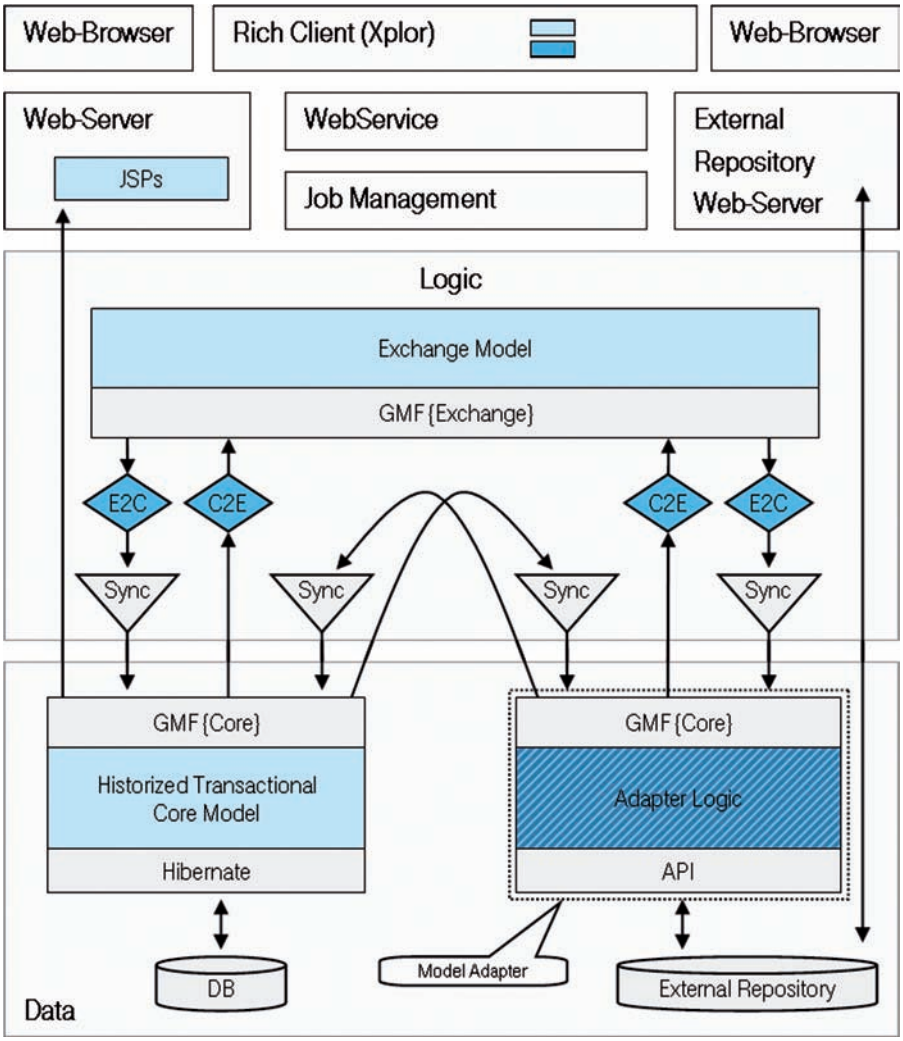

Abb. 5: Die externe Repository-Erweiterungsarchitektur.

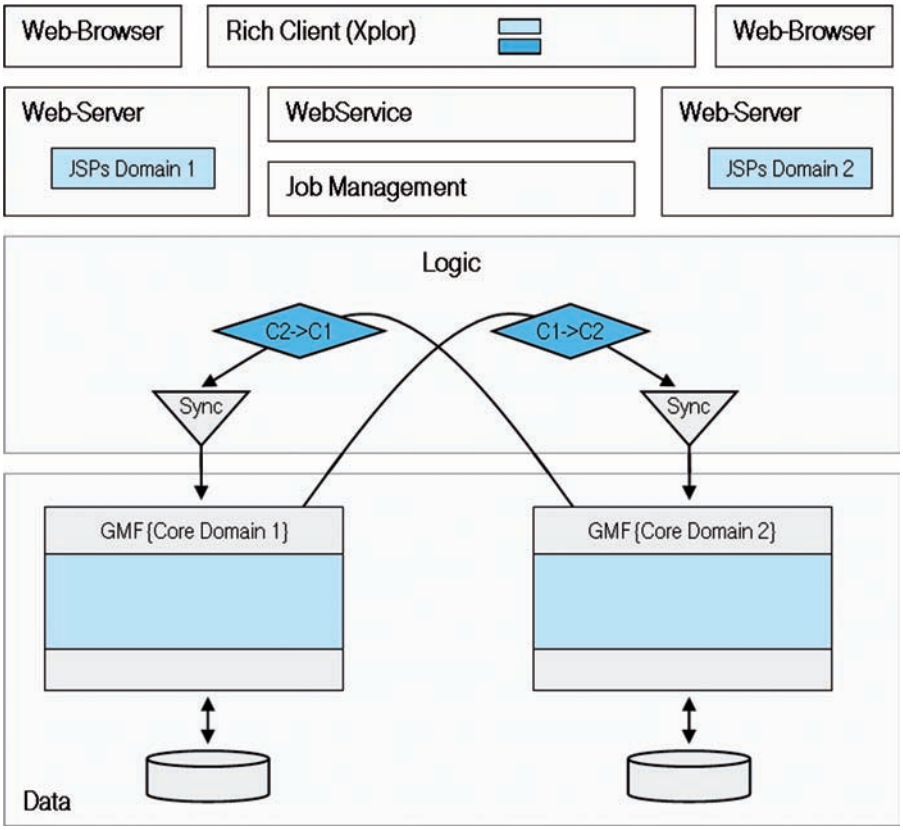

Abb. 6: Die Repository-Integrationsarchitektur.

adapter entwickelt (siehe Abbildung 5 rechts unten). Die Datenstruktur des externen Repositorys wird als Metamodell modelliert. Im Falle von CentraSite ist das Metamodell das JAXR-Information-Modell, erweitert um die benutzerdefinierten *RegistryObjects* für das EAM. Der Modelladapter implementiert die GMF-Modellfassade, wodurch sämtliche auf GMF basierenden Werkzeuge wie gewohnt eingesetzt werden können. Im Zusammenspiel mit der standardmäßigen transaktionalen- und historisierten datenbankbasierten Modellimplementierung kann somit ein externes Repository um diese Fähigkeiten erweitert werden. Zusätzlich kann dem externen Repository durch den Modelladapter weitere Funktionalität, z. B. Unterstützung von Vererbung und Modellfragmenten, hinzugefügt werden. Der Zugang zu den Daten eines Repository-Produkts wie Centrasite erfolgt über die dort vorhandene Web-Browser-Schnittstelle (siehe Abbildung 5 rechts oben).

Repository-Integrationsszenarien

Im Folgenden skizzieren wir einige Synchronisationsszenarien, die bei der Deutschen Telekom auf Basis des Frameworks umgesetzt wurden.

Synchronisation externer Datenquellen in einem Repository

Dieses Szenario wurde bei CEISeR realisiert (siehe Abbildung 1). Innerhalb des Repository-Servers gibt es eine Trennung zwischen *Exchange Model* und *Core Model*. Der Datenaustausch basiert auf XML-Dateien. Folgende Schritte werden bei der Synchronisation automatisiert durchgeführt:

- *Einlesen* der Exchange-XML-Dateien in *Exchange*-Modelle mit dem *Exchange*-Metamodell.
- *Transformation* der *Exchange*-Modelle in ein temporäres *Core*-Modell mit dem *Core*-Metamodell über einen speziell erstellten Modell-Transformator, basierend auf dem Modell-Transformations-Framework.
- *Auswahl* der gewünschten Elemente aus dem temporären Modell mit dem *Core*-Metamodell über einen *Model Locator*.
- *Synchronisation* der ausgewählten Elemente aus dem temporären Modell mit dem *Core*-Modell über den *Model Synchronizer*.

Synchronisation zweier Repositorys mit demselben Metamodell

Wie bereits erwähnt, wurde dieses Szenario bei der Anbindung des auf CentraSite basierenden Repositorys für das EAM-Modell gewählt (siehe Abbildung 5). Folgende Schritte werden bei der Synchronisation automatisiert durchgeführt:

- *Auswahl* der gewünschten Elemente aus dem Quell-Repository-Modell über einen *ModelLocator*.
- *Synchronisation* der ausgewählten Elemente mit dem Ziel-Repository-Modell über den *Model Synchronizer*.

Synchronisation zweier Repositorys mit unterschiedlichem Metamodell

Dieses Szenario wurde ebenfalls bei der Anbindung des auf CentraSite basierenden Repositorys für das EAM-Modell umgesetzt, um Daten zwischen CEISeR und dem EAM-Modell zu synchronisieren (siehe Abbildung 6). Folgende Schritte werden bei der Synchronisation automatisiert durchgeführt:

- *Auswahl* der gewünschten Elemente aus dem Quell-Repository-Modell über einen *ModelLocator*.
- *Transformation* der ausgewählten Elemente des Quell-Repository-Modells in ein temporäres Modell mit dem Metamodell des Ziel-Repositorys über einen speziell erstellten Modell-Transformator, basierend auf dem Modell-Transformations-Framework.
- *Synchronisation* der ausgewählten Elemente aus dem temporären Modell mit dem Ziel-Repository-Modell über den *Model Synchronizer*.

UI-Integration via Xplor

Da Xplor auf der GMF-Modellfassade operiert und Zugriff auf mehrere Repository-Server hat, ist dieser in der Lage, Verknüpfungen von Modellelementen der verschiedenen Modelle und Navigationsmöglichkeiten zwischen diesen anzubieten (z. B. CEISER<->EAM, EAM<->PlanningIT). Durch diese Eigenschaften kann der Xplor eine Verwebung der verschiedenen Modelle zu einem neuen Gesamtmodell auf Benutzungsschnittstellen-Ebene darstellen. Zusätzlich ist eine Anreicherung von Modellelementen um Informationen, die aus anderen Modellen kommen, möglich. Dem Benutzer gegenüber erscheint die Darstellung als ein großes virtuelles

Modell, in dem er durchgängig durch die einzelnen Aspekte browsen kann.

Fazit

Die fachliche und technische Integration von Daten zwischen heterogenen Repository-Produkten unterschiedlicher Hersteller wird durch ein Modell-Repository-Integrations-Framework ermöglicht. Ein solches Framework stellt sicher, dass die Integration einheitlich und konsistent erfolgt, ohne grundlegende Mechanismen für jedes Integrationsprojekt redundant zu entwickeln. Das Framework muss in der Lage sein, grundlegende Aufgaben – wie Modell-Synchronisation, -Transformation, -Lesen und -Schreiben – automatisiert durchzuführen. Die Kernidee bei dem hier vorgestellten modellbasierten Ansatz ist, dass Repositorys über Modelladapter, die alle ein und dieselbe generische Modellfassade implementieren, angesprochen werden. Hierdurch wird die aus der EAI bekannte *n*m*-Problematik vermieden. Auf Basis der Modellfassade können generische Werkzeuge und ein Repository-Server implementiert werden. Die Werkzeuge und der Repository-Server erledigen die grundlegenden Aufgaben der Integration. Zusätzlich kann weitere Funktionalität, wie eine Graph-Ansicht und Teamfunktionalität, implementiert werden. ■

Literatur & Links

[Alf] Alfabet Homepage, siehe: www.alfabet.de

[Ecl-a] Eclipse, Rich Client Platform, siehe: http://wiki.eclipse.org/index.php/Rich_Client_Platform

[Ecl-b] Eclipse, Zest: The Eclipse Visualization Toolkit, siehe: http://www.eclipse.org/gef/zest/

[IDS] IDS Scheer AG Homepage, siehe: www.ids-scheer.de

[Kim09] F. Kimmlingen, Konsistenz im SOA Modellrepository bei T-Mobile, in: JavaSPEKTRUM 06/2009

[Sen08] C. Sensler, A. Karalus, SOA@T-Mobile – Vollautomatische Service Provisionierung auf dem ESB – Teil 1-3, in: Java Magazin, 10/2008–12/2008

[Sen09] C. Sensler, A. Karalus, M. Märtens, Ein Blick hinter die Kulissen: Modell-repository@T-Mobile, in: JavaSPEKTRUM 01/2009

[Sof] Software AG Homepage, siehe: www.softwareag.com

[Sta07] T. Stahl, M. Völter, S. Efftinge, A. Haase, Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management, dpunkt.verlag (2. aktualisierte und erweiterte Auflage) 2007

[Ste09] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, Eclipse Modeling Framework, Addison-Wesley 2009