1 Projektberichte

Preprint of: J. Bodemann, W. Hasselbring: Visualisierung eines zentralen Repository für Softwareentwicklungsdokumente. In: Projekterfahrungen und Projektvorhaben mit Java, dpunkt Verlag, 1999 (Hrsg.: S. Maffeis, F. Toenniessen, C. Zeidler)

1.1 Visualisierung eines zentralen Repository für Softwareentwicklungsdokumente

Das hier beschriebene Projekt ist Teil des Gesamtprojektes REPTIL (Repository im Intranet) der Dresdner Bank, das mit dem Ziel initiiert wurde, die hausinterne Softwareentwicklung zu verbessern. Dieses Teilprojekt wurde vom Fraunhofer ISST für die Dresdner Bank AG durchgeführt. Wichtigstes Ziel ist es, den Softwareentwicklern den Zugriff auf die verschiedenen Informationsquellen, die für die Softwareentwicklung relevante Dokumente zur Verfügung stellen, durch eine einheitliche Benutzungsschnittstelle von möglichst vielen Plattformen aus zu ermöglichen. Das Fraunhofer ISST konzipierte und realisierte einen Informations-Browser, der die Navigation durch einen definierten Teil der Dokumente ermöglicht.

Das Papier diskutiert einerseits generelle Fragestellungen zum Aufbau von Repositories für Software-Entwicklungsdokumente und andererseits den Einsatz von Java als Entwicklungsplattform und Implementierungssprache. Während der erste Teil im wesentlichen auf konzeptionellen Überlegungen des zweiten Autors zum Gesamtprojekt der Dresdner Bank basiert, ist der zweite Teil ein Erfahrungsbericht des ersten Autors aus dem konkret für die Dresdner Bank realisierten Teilprojekt.

1.1.1 Fachlichkeit und Funktionalität

Im vorgestellten Projekt wurde die Visualisierung eines Informationssystem im Intranet der Dresdner Bank realisiert. Es war das Ziel, die Doku-

mente der gesamten hausinternen Softwareentwicklung zentral zur Verfügung zu stellen. Diese Dokumente sollten unter einer gemeinsamen, möglichst einheitlichen Oberfläche von verschiedenen Stellen aus abrufbar sein. Dadurch sollten allen direkt an der Anwendungsentwicklung beteiligten Personen, vom Projektleiter bis zum Programmierer und Tester, alle benötigten Informationen zur Verfügung gestellt werden. Diese Informationen umfassen Dokumente zur Analyse und zum Entwurf, Datenbankschemata, Modulbeziehungen, Quellcode einzelner Module, Testdaten, etc.

Anlaß für das Projekt war die unbefriedigende Situation auf verschiedene Altsysteme zugreifen zu müssen, um die notwendigen Informationen zusammenzutragen. Jedes dieser Systeme verfügt über ein eigenes Zugriffskonzept, das sich in Darstellung und Funktionalität von den anderen mehr oder weniger unterscheidet. Für den Benutzer stellte sich demnach das Problem, die Benutzungsschnittstellen verschiedener Systeme erlernen zu müssen, die zum Teil wenig intuitiv zu bedienen sind, und oft auch widersprüchliche Konzepte verfolgen. Zudem war es vor allem für neue Entwickler schwierig zu erlernen, welche Informationen über welches System wo zur Verfügung stehen.

Um dem Leser ein Gefühl für die Komplexität der Informationsbeschaffung zu geben, und den Grad der Arbeitserleichterung, der sich durch das Informationssystem im Intranet ergibt, einschätzbar zu machen, soll an dieser Stelle die Behebung eines Softwarefehlers als typischer Arbeitsprozeß grob skizziert werden. Einen aufgetretenen Fehler kann ein Entwickler aufgrund des Fehlercodes üblicherweise einer kleinen Menge von Entwicklungs-Dokumenten zuordnen. Um die Tragweite einer Änderung im Source-Code einschätzen zu können, sucht der Entwickler dann alle Dokumente, die vom zu ändernden Dokument abhängig sind, und um den Kontext verstehen zu können, alle Dokumente, von denen das zu ändernden Dokument abhängig ist. Eventuell ist es notwendig, verwendete Datenbankschemata zu verstehen oder Inter-Dokument-Beziehungen zu verfolgen, die ihrerseits wieder durch Beziehungen näher beschrieben sein können. Dadurch ergibt sich ein komplexes Geflecht von Dokumenten, die als eine Sicht auf eine Gesamtmenge von mehreren 10.000 Dokumenten zu verstehen ist.

Zusammengefaßt ergeben sich mit einem intranetbasierten Repository folgende Vorteile:

1. Der Zugriff auf die Dokumente wird durch eine einheitliche Schnittstelle möglich
2. Der Zugriff auf die Dokumente erfolgt durch einen einzigen Zugangspunkt
3. Eine Steigerung der Anfragegeschwindigkeit durch Daten-Replikation im Vergleich zu einigen Alt-Systemen

4. Anfragen über mehr als eine Datenquelle werden möglich
5. Die Zugriffssoftware wird auf allen Java-fähigen Plattformen verfügbar

1.1.2 Projektmanagement und -verlauf

Zum Zeitpunkt des Projekteinstiegs des Fraunhofer ISST war das REPTIL-System bereits lauffähig und im operativen Einsatz. REPTIL erwies sich bereits als nützlich für den Anwender und sollte jetzt verbessert werden. Grundlage für den Verbesserungsprozeß bildete das Benutzer-Feedback und die eigenen Tests mit dem alten System. Als Schwachpunkt in dieser älteren Version wurde neben einigen funktionalen Einschränkungen vor allem die graphische Präsentation identifiziert. Das graphische Design aller Benutzungsschnittstellen sollte komplett überarbeitet werden, wobei das Corporate Design zu beachten war. Der Dokument-Browser, eine der zentralen Komponenten, sollte zudem in der Bedienung verbessert werden.

Zusammen mit einem erfahrenen externen Industriedesigner hat sich ein Mitarbeiter des Fraunhofer ISST in mehreren Sitzungen in den Problembereich vor Ort durch Interviews und Analyse der existierenden Systeme eingearbeitet. Dabei war es von großem Nutzen, daß REPTIL in einer ausführbaren Version zur Verfügung stand. So konnte das System in der verfügbaren Form bedient, und die funktionalen Schwachstellen identifiziert werden. Daraufhin wurde in enger Kooperation mit dem Industriedesigner ein Darstellungskonzept für die Visualisierung entwickelt. Dabei konnte das Fachwissen des Informatikers und das des Designers kombiniert werden; es entstanden Synergieeffekte. Voraussetzung für diese Effekte waren das beiderseitige Grundverständnis des jeweils fremden Fachgebietes. In diesem konkreten Fall verfügte der Informatiker über eine Vorstellung ergonomischer Aspekte, die als Designwissen verstanden werden können. Der Industriedesigner war seinerseits in der Lage, seine Arbeitsergebnisse als HTML-Seiten zu liefern. Er verfügt somit über Erfahrungen mit der Repräsentation von Designvorgaben durch Internet-Technik und ist deshalb in der Lage die Möglichkeiten und Grenzen des Mediums abzuschätzen.

Die Ergebnisse der Arbeiten wurden vom Fraunhofer ISST in einem in Java implementierten Prototyp umgesetzt. Da das Ergebnis die Projektverantwortlichen der Dresdner Bank AG überzeugen konnte, die Gesamtkonzeption allerdings zu erheblichen Umstellungsarbeiten an REPTIL führen würden, wurde das Design und die Funktionalität der Browser-Komponente, soweit möglich, vorab in HTML nachgebildet und zusammen mit einigen weiteren erarbeiteten Verbesserungen den Benutzern

zur Verfügung gestellt. Das folgende Nutzer-Feedback führte zusammen mit den Erfahrungen der Projektgruppe und den neuen Möglichkeiten von Java gegenüber HTML zu der Planung erheblicher funktionaler Erweiterungen des Browsers, so daß der Java-Browser aus organisatorischen Gründen erst zu einem späteren Zeitpunkt in der Einsatzumgebung zum Einsatz kommen wird, obwohl die Java-Implementierung voll funktionsfähig ist und den Ansprüchen bezüglich Ablaufgeschwindigkeit und Stabilität voll genügt. Die Umstellung von HTML nach Java wird in der Einsatzumgebung erst mit einer wesentlich erweiterten Version vollzogen werden.

Für die Durchführung des Projektes in dieser Form sprachen die guten Erfahrungen, die bereits an anderer Stelle mit Java gemacht wurden. Die Entscheidung für Java als Implementierungsplattform war vom Auftraggeber vorgegeben. Die Plattformunabhängigkeit der Java-Virtuellen-Maschine wird als großer Vorteil gesehen, vor allem im stark heterogenen Umfeld. Durch den Einsatz verschiedener Entwicklungsumgebungen auf verschiedenen Plattformen würden sonst zahlreiche Inseln entstehen, die den Austausch von Erfahrungen und Ressourcen erheblich erschweren würden. Zudem ermöglicht Java den Einsatz von Mitarbeitern über Projekte und Abteilungen hinweg, in denen Java verwendet wird, da immer die gleiche Plattform genutzt wird. Dadurch können die Vorteile einer großen Zahl von Mitarbeitern besser genutzt werden, indem Mitarbeiter bei Bedarf schneller an anderen Stellen eingesetzt werden können. Dies ist ein Vorteil, der zu Zeiten des Mainframe-Einsatzes vorhanden war, und durch heterogene Client-Server-Systeme weitgehend verloren gegangen ist. Bei einer Entwicklung mit C++ ist ein Austausch von Mitarbeitern in dieser Form nicht so einfach möglich, da sich hier insbesondere die Bibliotheken auf unterschiedlichen Systemen sehr stark unterscheiden.

1.1.3 Technische Architektur

Die IT-Infrastruktur der Dresdner Bank ist heterogen. Als Arbeitsplätze stehen Hostterminals, X-Terminals, sowie Windows 3.1, 95 und NT zur Verfügung. Somit bot sich eine Intranet-Lösung an, um an allen Arbeitsplätzen eine einheitliche Oberfläche zu erhalten. Auf Serverseite finden sich Mainframes, sowie Unix- und NT-Server.

Die zur Verfügung zu stellenden Dokumente sind über die gesamte Firma verteilt. Alle Daten, auf die innerhalb einer zu tolerierenden Zeit aus den lokalen Repositories zugegriffen werden kann, werden bei Bedarf direkt von dort kopiert. Alle anderen werden in ein zentrales Repository

eingestellt, wobei die Replikate einmal täglich aktualisiert werden. Die Information darüber, welche Daten in dem Repository zur Verfügung stehen, und welche durch direkte Zugriffe auf die originalen Datenbestände zu beziehen sind, ist in einer Zwischenschicht (einem Informations-Broker) verkapselt.

Abbildung 1 stellt die Grobarchitektur der Anwendung dar. Die Pfeile symbolisieren den Datenfluß, nicht den Kontrollfluß zwischen den einzelnen Komponenten. Die Applets erhalten über die Daten per RMI vom RMI-Server. Der RMI-Server ließt die Daten systemabhängig entweder per JDBC aus dem zentralen Repository oder direkt aus einigen operativen Systemen.

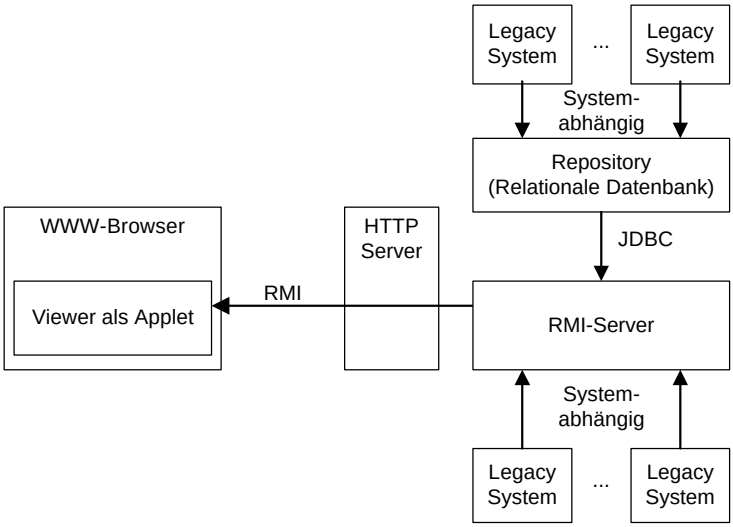

Abbildung 1: Grobarchitektur der Anwendung.

Die Architektur kann mit dem Broker-Architekturmuster [Buschmann et al. 96] grob beschrieben werden. Das Broker-Architekturmuster strukturiert verteilte Systeme als entkoppelte Komponenten, die über den Aufruf von Diensten kooperieren. Eine Broker-Komponente ist dabei für die Koordination der Kooperation (Kommunikation und Synchronisation) zuständig, z.B. die Weiterleitung von Anfragen oder die Rückgabe von Ergebnissen. In der vorgestellten Architektur übernimmt der RMI-Server diese Rolle. Feingranulare Entwurfsmuster, wie sie z.B. in [Gamma et al. 95] beschrieben werden, wurden im vorgestellten Projekt nicht explizit zur Strukturierung des Feinentwurfs eingesetzt.

Die graphische Visualisierung der verschiedenen Dokumentbeziehungen ist aufgrund der sehr verschiedenartigen Dokumenttypabhängig-

keiten recht anspruchsvoll, so daß hierfür Java eingesetzt wird. Die folgenden beiden Screenshots zeigen Beziehungsgeflechte zwischen mehreren Dokumenten. In der täglichen Praxis kann die Darstellung sehr umfangreich werden. Die Komplexität entsteht auch durch die große Zahl verschiedener Beziehungstypen (mehr als 30). Die Darstellung der Dokumentbeziehungen übernimmt ein Applet, das auf den Arbeitsplätzen in Netscape abläuft. Die Applets greifen durch RMI auf den Server zu. Der RMI-Server realisiert eine JDBC-Verbindung zu einer relationalen Datenbank (Informix), in der das Repository zentral gespeichert ist. Der RMI-Server dient als Broker zwischen den Klienten (hier den Applets) und den verschiedenen Servern. Das zentrale Repository kann als eine Art Proxy für die entsprechenden operativen Systeme angesehen werden, deren Daten in das Repository kopiert werden.

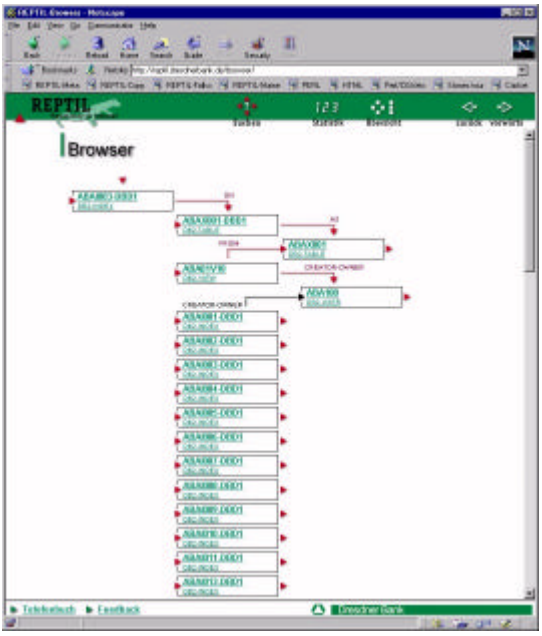

Abbildung 2: Darstellung der Dokumentenbeziehungen im Browser

Da das JDBC-Protokoll nicht netzwerktransparent ist, sollte im Projekt ein RMI-Server eingesetzt werden, um auf die Daten über das Netz zuzugreifen. Da RMI erst mit Java in der Version 1.1 zur Verfügung steht, die Mehrzahl der weit über 1000 Clients aber den Netscape Navigator in der Version 3.x verwenden, und damit nicht Java 1.1 fähig sind,

ergibt sich ein Kompatibilitätsproblem. Zur Lösung bieten sich zwei Alternativen an:

1. Der Umstieg auf die zum damaligen Zeitpunkt neuste Version des Netscape Navigators 4.04, für die ein Patch zur Verfügung stand, der zwar nicht den vollen Java 1.1 Funktionsumfang zur Verfügung stellte, allerdings alle benötigten Funktionen bot.
2. Die Verwendung des Java-Activators, der mittlerweile technisch modifiziert als Java-Plug-In bezeichnet wird, und den Einsatz einer beliebigen JVM (Java Virtuelle Maschine) für den Microsoft Internet Explorer und den Netscape Navigator erlaubt.

Da der Einsatz des REPTIL-Systems den Mitarbeitern der Dresdner Bank nicht vorgeschrieben wird, sondern vielmehr durch eine bessere Leistung die potentiellen Anwender zum Aufgeben der bisher verwendeten Methoden der Informationsbeschaffung überzeugen muß, spielte der Aufwand, der für die Anwender entsteht, wenn er das neue System erproben will, eine große Rolle. Es schien günstiger, die Anwender zur Installation des Activators zu veranlassen, der als Ergänzung zum bisher eingesetzten Browser aufgefaßt werden kann, als zum Testen von REPTIL zunächst einen neuen Internet-Browser installieren zu müssen. Aus diesem Grund fiel die Entscheidung auf den Einsatz des Activators.

Die Struktur der Datenbank für das zentrale Repository ist in Abbildung 3 als UML-Klassendiagramm [Fowler & Scott 1997] dargestellt. Die zentralen Klassen sind Dokumente und deren Beziehungen zu anderen Dokumenten. Im zentralen Repository wird die interne Struktur der einzelnen Dokumente nicht explizit berücksichtigt. Zu jedem Dokument werden die Details in der relationalen Datenbank in entsprechenden Feldern (Blobs) als Text-Dateien und/oder Bitmap-Graphiken gespeichert (Klasse Inhalt). Abhängig vom Typ der Dokumente (Klasse Information) können diese Felder dann für die Visualisierung aufbereitet werden.

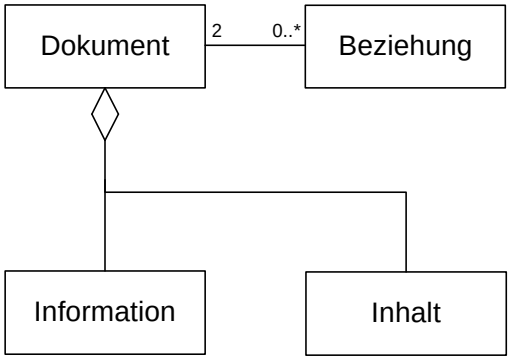

Abbildung 3: Das Datenmodell des Repository (vereinfacht).

Der hier vorgestellte Ansatz für ein zentrales Repository unterscheidet sich grundsätzlich von integrierten Softwareentwicklungsumgebungen, in denen die einzelnen Werkzeuge eng über ein *gemeinsames* Repository gekoppelt werden [Nagl 1996]. Bei eng gekoppelten integrierten Softwareentwicklungsumgebungen arbeiten die Entwicklungswerkzeuge direkt auf dem zentralen Repository. Das ist im vorgestellten Projekt nicht der Fall. Hier werden die Dokumente aus den operativen Datenbanken in das zentrale Repository kopiert. Dieses Verfahren hat sich z.B. auch bei Datawarehouse-Systemen [Inmon 1996] bewährt, um die operativen Systeme möglichst wenig zu belasten.

Ein Problem besteht nun darin, daß die operativen Systeme ihre Dokumente in sehr unterschiedlichen Formaten ablegen. Für derartige Architekturen hat sich das Konzept der föderierten Datenbanken bewährt [Sheth & Larson 1990]. In föderierten Datenbanken wird angestrebt, heterogene Komponentendatenbanken über eine Föderierungsschicht zu koppeln, ohne die Autonomie der Komponentendatenbanken unnötig einzuschränken. Der Begriff Föderation kommt aus der Politik. Die Bundesrepublik Deutschland ist z.B. eine Föderation aus 16 teilweise autonomen Ländern.

Für die Integration der heterogenen Datenbankschemata der Komponentendatenbanken hat es sich bewährt, die 5-Ebenen-Schemaarchitektur aus [Sheth & Larson 1990], die in Abbildung ??? dargestellt ist, als Grundlage zu nehmen [Conrad et al. 97, Hasselbring 1997]. Auf der untersten Ebene sind die lokalen Schemata der Komponenten Datenbanksysteme dargestellt. Diese lokalen Schemata entsprechen den konzeptuellen Schemata in der klassischen 3-Ebenen-Architektur für relationele Datenbanken. Für lokale Anwendungen können von diesen konzeptuellen Schemata Sichten abgeleitet werden (unten links in der Abbildung). Für eine Föderation werden alle lokalen Schemata der zu integrierenden Datenbanken in Komponenten-Schemata transformiert, die dann alle in einem einheitlichen kanonischen Datenmodell spezifiziert sind. Dadurch wird die Datenmodellheterogenität überwunden. Aus den Komponenten-Schemata werden dann Export-Schemata gefiltert, um die für die Föderation relevanten Teile zu erhalten. Diese Export-Schemata werden dann in möglicherweise mehrere föderierte Schemata integriert. Aus den föderierten Schemata können für gloable Anwendungen externe Schemata abgeleitet werden; analog zu den externen Schemata, die für lokale Anwendungen aus den lokalen, konzeptuellen Schemata abgeleitet werden.

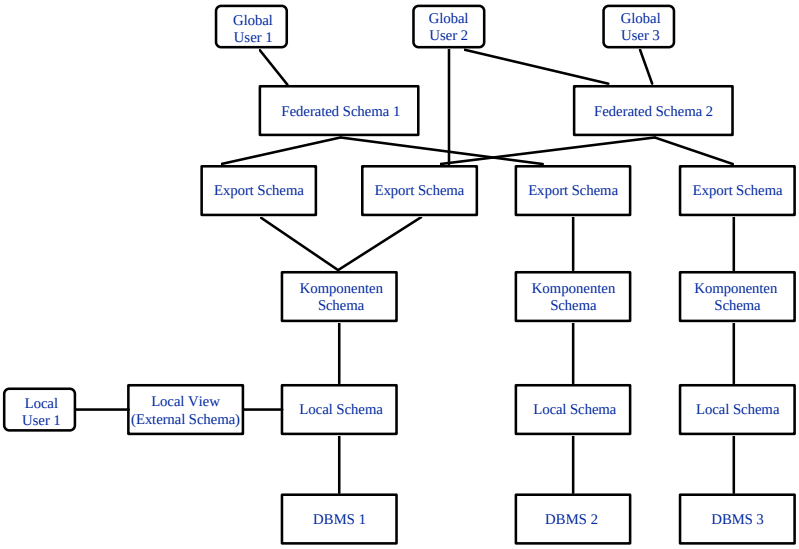

Abbildung 4: Die 5-Ebenen-Schemaarchitektur

Die im vorgestellten Projekt realisierte Schemaarchitektur ist in Abbildung 5 skizziert. Die Schemata der lokalen operativen Systeme werden durch entsprechende Softwarekomponenten direkt auf das Schema des zentralen Repository abgebildet, welches dem globalen, föderierten Schema der 5-Ebenen-Schemaarchitektur entspricht. Die Pfeile zwischen den lokalen Schemata und dem föderierten Schema in Abbildung 5 entsprechen diesen Softwarekomponenten. Das föderierte Schema beschreibt die Datenstrukturen aus Abbildung 3 als SQL-Schema (create table ...) für die Daten, die in der der relationalen Datenbank aus Abbildung 1 gespeichert werden (also im zentralen Repository). Für die Intranet-Applikationen werden externe Sichten auf das föderierte Schema durch Datenbank-Views (create view ...) abgeleitet. Auf Komponenten- und Export-Schemata wurde verzichtet, so daß hier nur drei Schemaebenen vorhanden sind. Diese Architektur ist trotzdem konform mit der 5-Ebenen-Schemaarchitektur aus [Sheth & Larson 1990]; die zwei Ebenen für Komponenten- und Export-Schemata enthalten hier keine Schemata.

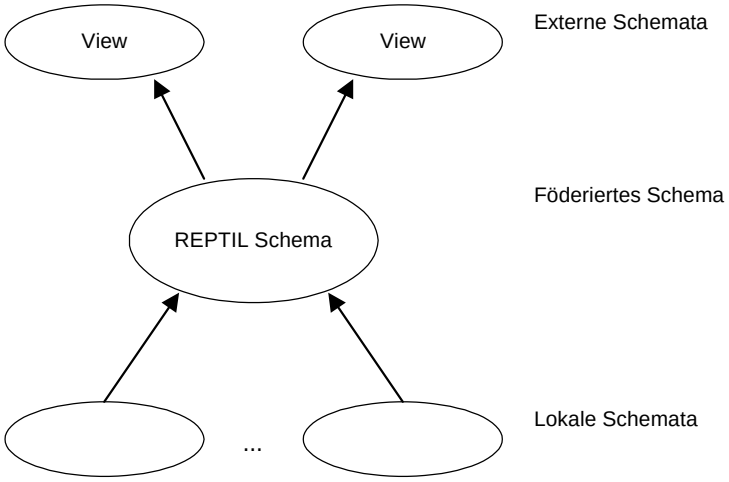

Abbildung 5: Die Schemaarchitektur.

1.1.4 Mengengerüste und Leistungsdaten

Der im Repository verfügbare Datenbestand hat zur Zeit einen Umfang von ca. 2 GB. Aufgrund der guten Akzeptanz des Systems bei den Anwendern, werden zügig weitere Datenquellen eingefügt, so daß zum Ende des Jahres 1998 mit einem 50 % höheren Datenaufkommen zu rechnen ist. Zur Zeit wird das System von ca. 1500 Anwendern eingesetzt. Der potentielle Anwenderkreis umfaßt ca. 2500 Entwickler. Durch die Aufnahme weiterer Datenquellen aus bisher unberücksichtigten Organisationseinheiten wird sich der Kreis der potentiellen Benutzer in Zukunft stark erhöhen.

Die Visualisierungskomponente ist mit ca. 2100 Zeilen Java Quellcode realisiert, der sich auf 18 Klassen verteilt. Die Klassen sind mit Ausnahme der Beziehungen zum Java API ausschließlich durch Aggregationsbeziehungen miteinander verbunden.

Der RMI-Server ist mit ca. 900 Zeilen Java Quellcode und 10 Klassen realisiert. Auch innerhalb dieser Klassen gibt es keine Vererbungsbeziehungen.

1.2 Der Entwicklungsprozeß, die Methodik

Das dem Projekt zugrundeliegende Entwicklungsmodell ist im Kapitel „Projektmanagement und –verlauf“ bereits skizziert worden, und soll

hier nicht noch einmal dargestellt werden. Im folgenden werden auch unsere Erfahrungen mit den Entwicklungsumgebungen beschrieben.

Größere Projekte, wie z.B. das Nachfolgeprojekt, das im Kapitel „Zusammenfassung, Ausblick in die Zukunft“ kurz beschrieben wird, werden nicht im vorhinein vollständig spezifiziert. Statt dessen werden zunächst die Basisfunktionalität oder besonders kritische Teile umgesetzt, um das Projekt dann Schritt für Schritt zu vervollständigen. Dabei kommt es in der Praxis häufig zu der Situation, daß Designvorgaben und Implementierung auseinander driften. Beispielsweise, weil die Implementierung vor allem in der Anfangsphase schneller voranschreitet als vorgesehen, oder weil sich einige Vorgaben in der Praxis als ungünstig erweisen. Das führt häufig dazu, daß Programme im Nachhinein dokumentiert werden müssen, was bei komplexen Projekten nur schwer zu organisieren ist. Die resultierenden Konsistenzprobleme lassen sich zum Teil mit „Together/J“ der Firma Object International entschärfen [Object International 1998]. Es handelt sich dabei um ein CASE Werkzeug, das einen großen Teil der UML-Notation unterstützt [Fowler & Scott]. Da das Werkzeug als Datenbasis den Source-Code verwendet, und die Darstellung der Notation generiert, wird die oben beschriebene Arbeitsweise recht gut unterstützt. Damit kann ein Projekt so weit wie nötig oder möglich spezifiziert werden. Anschließend wird es von einigen Programmierern umgesetzt. Diese Umsetzung ist dann wieder die Basis für die Aktualisierung der Spezifikation.

Die Implementierung erfolgte mit dem JDK von Sun, das in Zweifelsfällen immer die Referenz darstellt, in Verbindung mit einem beliebigen ASCII-Editor, und teilweise mit einer integrierten Entwicklungsumgebung (IDE). Zum Zeitpunkt der Entwicklung war die Auswahl an IDEs für Java noch relativ klein. Für das beschriebene Projekt wurde Symantecs „Café“ eingesetzt [Symantec 1998]. Die Weiterentwicklung erfolgt seit einiger Zeit mit „Visual Café“ aus dem gleichen Hause. Obwohl bereits andere Werkzeuge der Firma Microsoft am Fraunhofer ISST zum Einsatz kommen, und Visual J++ zur Verfügung steht, wird dieses Werkzeug für die Entwicklung nicht eingesetzt. Das liegt vor allem daran, daß die Einschätzung von Microsoft, Java sei „just another programming language“ nicht geteilt wird. Dem Versuch, Java durch propritäre „Ergänzungen“ zu „verbessern“ steht das Fraunhofer ISST sehr skeptisch gegenüber. Zur Zeit werden die IDEs verschiedener anderer Hersteller evaluiert. Es ist deutlich zu sehen, daß der Reifegrad der Java Entwicklungswerkzeuge den Reifegrad vergleichbarer Werkzeuge anderer Sprachen noch nicht erreicht hat. Vor allem „Visual Basic“ von Microsoft und „Delphi“ von Inprise (früher Borland) sind noch deutlich überlegen. Die Geschwindigkeit, mit der sich die Java-Entwicklungsumgebungen verbessern, läßt dennoch für die Zukunft einiges erwarten.

Abbildung 6 stellt den Entwicklungsprozeß graphisch dar. Nach dem initialen Entwurf durch den Industriedesigner, durch den eine reine HTML-Realisierung zur Evaluierung bezüglich des Corporate Designs implementiert wurde, wurde eine entsprechende HTML-Version mit CGI-Anbindung an die REPTIL-Datenbank realisiert. Diese HTML-Lösung diente als wesentliche Komponente der initialen Anforderungsbeschreibung. Dann wurde in einem inkrementellen, iterativen Entwicklungsprozeß die Umsetzung mit Java durchgeführt. Nach einem Grobentwurf der Softwarearchitektur durch den Systemarchitekten wurde mit Java programmiert und parallel dazu wurden mittels Together/J der Feinentwurf und die Dokumentation aktualisiert. Die Evaluierung der Java-Prototypen dient dann zur Entscheidung, ob weitere Änderungen nötig sind, so daß wir einen evolutionären Entwicklungsprozeß erhalten [Budde et al. 92]. Das hier beschriebene Teilprojekt ist mittlerweile abgeschlossen, allerdings ist das Ende der Entwicklung des Gesamtprojektes noch nicht erreicht, was bei einem evolutionären Entwicklungsprozeß nicht ungewöhnlich ist. Es hatte sich auch schon in anderen Projekten bewährt, objektorientierte Entwicklungsmethoden mit Prototypingtechniken zu kombinieren [Hasselbring & Kröber 1998].

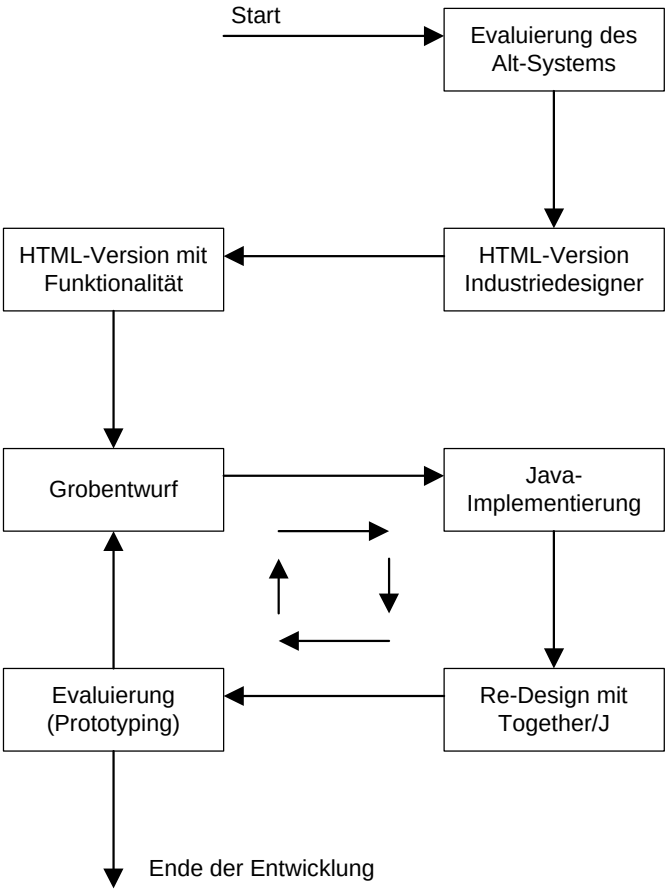

Abbildung 6: Der evolutionäre Entwicklungsprozeß.

1.3 Kritische Bewertung des Einsatzes von Java

Als Vorteil bei der Entwicklungsplanung ist die Politik Suns zu werten, die geplanten Erweiterungen der Bibliotheken in Form zusätzlicher APIs frühzeitig anzukündigen und diese Termine in der Regel auch grob einzuhalten. In dieser Kombination lassen sich Schwächen von Java frühzeitig umgehen und die Konzeption des Projektes auf die zu erwartenden Möglichkeiten ausrichten. Vor allem in Kombination mit dem in den vorhergehenden Kapiteln beschriebenen Entwicklungsmodell ergibt sich ein praxistaugliches Modell.

1.3.1 Vor- und Nachteile bzw. Stärken/Schwächen von Java

Die Sprache Java selbst weist im Vergleich zu C++ eine deutlich geringere Komplexität auf. Im Vordergrund steht nicht das Erlernen der Sprache und ihrer zahlreichen Möglichkeiten, sondern vielmehr das Beherrschen der umfangreichen Bibliotheken. Dadurch ist der Einstieg in die Sprache leichter möglich, vor allem, wenn das objektorientierte Paradigma verstanden wurde.

Ein wesentlicher Vorteil von Java gegenüber z. B. C++ ist aus unserer Sicht besonders erwähnenswert: der Garbage-Collector. Viele Probleme beim Einsatz von z. B. C++ entstehen im Kontext der Speicherverwaltung. Schon in der Designphase von C++ Anwendungen ist es schwierig ein Konzept für die Freigabe von angefordertem Speicher konsistent abzubilden. Hinzu kommen zahlreiche Fehler auf Implementierungsebene, die sehr schwer zu finden sind, weil eine zeitliche Zuordnung von Absturz und fehlerhafter Speicherbehandlung nur selten besteht. Zusammen mit der fehlenden Pointerarithmetik führt der Garbage-Collector zu deutlich stabileren Programmen, allerdings zu Lasten des Laufzeitverhaltens.

Das ist auch gleichzeitig ein häufig gegen Java vorgebrachtes Argument, nämlich die im Vergleich zu C und C++ Anwendungen deutlich geringere Ablaufgeschwindigkeit. Im Rahmen dieses Projektes war das Problem allerdings nicht so relevant, wie zunächst angenommen. Das liegt vor allem daran, daß nicht die maximal erreichbare Ablaufgeschwindigkeit, sondern eine Sollgeschwindigkeit erreicht werden mußte, spezifiziert durch die unscharfe Formulierung: „Ziel ist es, dem Anwender ein zügiges Arbeiten zu ermöglichen.“ Dieses Ziel konnte erreicht werden, und es liegt die Vermutung nahe, daß Java dieses Ziel für zahlreiche Anwendungen ebenfalls erreichen kann. Diese Vermutung wird gestärkt durch die Tatsache, daß das oben erwähnte komplexe CASE Werkzeug „Together / J“ vollständig in Java implementiert ist.

Die Anbindung an die relationale Datenbank erfolgte direkt über JDBC. Obwohl dem Repository eine Informix Datenbank zugrunde liegt, wurde der Prototyp mit einer Oracle Datenbank realisiert, weil die JDBC-Treiber für dieses Produkt kostenlos zur Verfügung standen. Die Anwendung ist als unproblematisch zu werten und in einigen Bereichen mehrfach schneller als eine Lösung über die „JDBC / ODBC Bridge“.

1.3.2 Was haben wir gelernt? (Lessons learned)

An dieser Stelle werden die Erfahrungen mit Java noch einmal kurz zusammengefaßt.

Aus unserer Sicht ist Java als Plattform für die Entwicklung von Anwendungen geeignet. Zwei häufig diskutierte Aspekte, die als Argumente für bzw. gegen Java Verwendung finden, sind Plattformunabhängigkeit und Ablaufgeschwindigkeit der erzeugten Anwendungen.

Plattformunabhängigkeit war bei diesem konkreten Projekt durchaus ein Grund für den Einsatz. Langfristig gesehen erscheint der Vorteil, daß sich die Anwendungsentwickler auf eine Entwicklungs-Plattform konzentrieren können, von größerer Bedeutung zu sein. Durch die Abstraktion von den konkreten Maschinen werden die technischen Probleme gebündelt und das Wissen über den fachlichen Problembereich kann stärker in den Vordergrund rücken.

Die Ablaufgeschwindigkeit der in Java entwickelten Anwendungen ist mit den in C bzw. C++ geschriebenen nicht vergleichbar. Daran kann auch der Einsatz von „Just-in-Time-Compilern“ zur Zeit nichts ändern. Allerdings ist die maximal erreichbare Geschwindigkeit nicht immer notwendig. Häufig läßt sich, wie in diesem konkreten Projekt, eine Sollgeschwindigkeit festlegen, die auch mit Java erreicht werden kann. Die Entwicklungen im Hardwarebereich (z. B. Prozessoren, die Java-Byte-Code in Hardware ausführen) werden die Möglichkeiten von Java noch vergrößern.

Java als Programmiersprache ist deutlich weniger komplex als C++, und entsprechend einfacher zu erlernen. Ist das objektorientierte Paradigma verstanden, lassen sich in deutlich kürzerer Zeit Anwendungen konstruieren, die im allgemeinen stabiler ablaufen. Letzteres ist auf das automatische Speichermanagement und auf die fehlende „Pointerarithmetik“ zurückzuführen, die bei C++ Programmen einen Großteil der Laufzeitfehler verursachen dürfte.

Einen deutlich höheren Arbeitsaufwand stellt bei Java die Einarbeitung in die umfangreichen Bibliotheken dar.

1.4 Zusammenfassung, Ausblick in die Zukunft

Dieser Erfahrungsbericht beschreibt die Entwicklung eines firmeninternen Informationssystems, das den Mitarbeitern der Dresdner Bank von verschiedenen Arbeitsplätzen aus Zugriff auf den verteilten Datenbestand von Softwareentwicklungsdokumenten geben soll. Es ist geplant auch weitere Bereiche durch das System zu bedienen.

Zur Entwicklung wurden Internet-Techniken eingesetzt. Ein Teil des Gesamtsystems wurde mit Java realisiert. Die Erfahrungen sind dabei überwiegend positiv, so daß sich Java in diesem Projekt weiter etablieren wird.

Das realisierte Repository ist im Prinzip ein Datawarehouse [Inmon 1996], da viele Daten aus den operativen Systemen in eine zentrale Datenbank kopiert werden, um eine ausreichende Performance für den Zugriff zu erhalten. Der Zugriff auf die Daten erfolgt navigierend. Im Unterschied zu typischen Datawarehouse-Anwendungen ist bisher jedoch keine OLAP-Funktionalität (Online Analytical Processing) vorgesehen. Das zentrale Repository könnte jetzt aber auch zur Auswertung von Metriken für Entwürfe und Programme verwendet werden, um Aussagen über die Qualität der Softwareentwicklung zu erhalten. Außerdem werden nicht alle Daten in das zentrale Repository kopiert. Dadurch ergeben sich unterschiedliche Konsistenzgrade, die aber in der Praxis selten zu Problemen führen werden. Hier sind Erweiterungen auf Java-Ebene denkbar, die es ermöglichen, in speziellen Fällen die angeforderten Daten nicht aus dem Repository, sondern aus der Originalquelle zu holen. Auch Aktualisierungsstrategien für materialisierte Sichten, die im Datenbankbereich entwickelt wurden, könnten eingesetzt werden [Zhuge et al. 1995].

CORBA [Mowbray & Zahavi 1995] wird zukünftig eingesetzt werden, um RMI zu ersetzten und damit flexibler auf die Daten zugreifen zu können. Insbesondere ist dann eine direkte Anbindung von C++ Komponenten in Bereichen möglich, in denen Java (noch) nicht eingesetzt werden kann. Das könnte z.B. bei älteren Datenbanksystemen der Fall sein, die nur über Schnittstellen zu C/C++ verfügen und nicht auf aktuellere Versionen der Datenbanksysteme umgestellt werden können.

Neben der Erweiterung der Funktionalität soll in einem größeren Projekt ein „Computer-Based-Training“ (CBT) Entwicklungssystem erstellt werden, mit dem der Benutzer mit den umfangreichen Möglichkeiten von REPTIL vertraut gemacht werden kann.

1.5 Kurzvorstellung der Autoren

Jörn Bodemann ist wissenschaftlicher Mitarbeiter am Fraunhofer ISST, Dortmund. Er studierte Informatik und Philosophie an der Universität Dortmund, von der er auch sein Informatik-Diplom erhielt. Seine Arbeitsschwerpunkte liegen im Bereich Softwaretechnik, wobei dem Bereich komponentenorientierter Softwareentwicklung besondere Aufmerksamkeit gilt. Er koordiniert die Java-Aktivitäten der Dortmunder Niederlassung und hat in diesem Kontext an verschiedenen Projekten mit deut-

schen Großbanken mitgewirkt. Das hier beschriebene Projekt entstand unter seiner Leitung.

Dr. Wilhelm Hasselbring ist Assistenzprofessor am INFOLAB der Universität Tilburg, Niederlande. Er erhielt sein Informatik-Diplom an der Technischen Universität Braunschweig, wechselte dann an die Universität Essen und promovierte 1994 an der Universität Dortmund, wo er bis 1998 als wissenschaftlicher Assistent tätig war. Zwischenzeitlich besuchte er die Universität Edinburgh und das Trinity College in Dublin. Seine Arbeitsschwerpunkte liegen im Bereich der Softwaretechnik für verteilte und kooperative Informationssysteme. Dabei spielt die softwaretechnische Realisierung von flexiblen Softwarearchitekturen mit CORBA und Java eine wichtige Rolle.

1.6 Literaturverzeichnis

[Budde et al. 92] R. Budde, K. Kautz, K. Kuhlenkamp, H. Züllighoven. Prototyping - An Approach to Evolutionary System Development, Springer-Verlag, 1992.

[Buschmann et al. 96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture - A System of Patterns, Wiley 1996

[Conrad et al. 97] S. Conrad, B. Eaglestone, W. Hasselbring, M. Roantree, F. Saltor, M. Schönhoff, M. Strässler, M. Vermeer. Research Issues in Federated Database Systems (Report of EFDBS '97 Workshop), SIGMOD Record 26(4): 54-56, 1997.

[Fowler & Scott 1997] M. Fowler, K. Scott. UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, Reading, MA, 1997.

[Gamma et al. 95] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns - Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[Hasselbring 1997] W. Hasselbring. Federated Integration of Replicated Information within Hospitals, International Journal on Digital Libraries 1(3): 192-208, 1997.

[Hasselbring & Kröber 1998] W. Hasselbring, A. Kröber. Combining OMT with a Prototyping Approach, The Journal of Systems and Software, 1998.

[Inmon 1996] W.H. Inmon. The Data Warehouse and Data Mining, Communications of the ACM 39(11): 49-50, 1996.

[Mowbray & Zahavi 1995] T.J. Mowbray, R. Zahavi. The Essential CORBA: Systems Integration Using Distributed Objects, Wiley, New York, 1995.

[Nagl 1996] M. Nagl (Hrsg.). Building tightly integrated software development environments: the IPSEN approach, Springer-Verlag, Lecture Notes in Computer Science 1170, Berlin, 1996.

[Object International 1998] Object International Software Ltd. Together/J, http://www.oi.com/, 1998.

[Sheth & Larson 1990] A. Sheth, J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys 22(3): 183-236, 1990.

[Symantec 1998] Symantec Corporation. Symantec Cafe, http://www.symantec.com/, 1998.

[Zhuge et al. 1995] Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Widom. View maintenance in a warehousing environment, SIGMOD Record 24(2): 316-327 (Proc. ACM SIGMOD International Conference on Management of Data, 1995.