Information Model in Zusammenarbeit mit einer Reihe von Firmen entstanden ist und aufgrund der großen Verbreitung von Microsoft Produkten hat das MSR eine gute Chance zu einem Industriestandard zu werden. Da darüberhinaus die Möglichkeit besteht, existierende Informationsmodelle zu erweitern bzw. gänzlich neue zu entwerfen, bietet das MSR ein hohes Maß an Flexibilität und Einsetzbarkeit.
Allerdings stellt das MSR nicht alle Dienste zur Verfügung, die von [BDa94] gefordert werden. So enthält das System keine Workflow-Control-Komponente und auch keine Möglichkeit zur Notifikation.

Quellen

[BDa94] BERNSTEIN, P.A., DAYAL, U.: *An overview of repository technology*, Proc. 20th Int. Conf. on Very Large Databases (VLDB '94), San Francisco, 1994, Morgan Kaufmann Publishers, pp.705-713

[Ber97] BERNSTEIN, P.A., HARRY, B., SANDERS, P., SHUTT, D., ZANDER, J.: *The Microsoft Repository*

[Ber98] BERNSTEIN, P.A*.: Repositories and Object-Oriented Databases*, ACM SIGMOD Record 27:1, 1998, pp.88-96

[Ber99] BERNSTEIN, P.A., BERGSTRAESSER, T., CARLSON, J., PAL, S., SANDERS, P., SHUTT, D.: *Microsoft Repository Version 2 and the Open Information Model*, Information Systems 24:2, April 1999, pp.71-98

[Hof98] HOFFMANN, Oliver: *Reintegration von Entwurfsdokumenten zur Unterstützung des Rountrip Engineering in der SPEARMINT-Umgebung*. Fachbereich Informatik, Universität Kaiserslautern, Diplomarbeit, Mai 1998.

[Jae99] JAEGER, Andreas: *Entwicklung eines komponentenbasierten Programmtransformationssystems*. Fachbereich Informatik, Universität Kaiserslautern, Diplomarbeit, August 1999.

[MSDN] http://msdn.microsoft.com/repository

[Tan94] TANNENBAUM, A*.: Implementing a Corporate Repository, The Models Meet Reality*. 1994

[Orr96] ORR, Ken: *Data Warehousing Technology*. 1996

[Ort99] ORTNER, Erich: *Repository Systems*. In: Informatik Spektrum, Nr.4, S.235-251 und Nr.5, S.351-363

[Val95] VALETTO G., KAISER, G.E.: *Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments*, IEEE Seventh International Workshop on Computer-Aided Software Engineering Environments, July 1995, S.40-48

umfangreich, da die Möglichkeiten der Schemaspezifikation in den Datenbanken nicht sehr von den Standard-SQL-Möglichkeiten abweichen.

Component Description Model (CDE)

Das Component Description Model (CDE) spezialisiert die Konzepte aus UML, um Werkzeuge der komponentenbasierten Entwicklung zu unterstützen. Dabei wird der Begriff Komponente definiert als „ein Softwarepaket, das Dienste über eine bestimmten Schnittstelle anbietet".

CDE enthält Modellierungsmöglichkeiten für Vor- und Nachbedingungen, Methodensignaturen und viele andere Konzepte, die in den bekannten Objektmodellen: COM, CORBA und Java Beans verwendet werden.
Das Untermodell COM erweitert CDE um eine Reihe von Details des Component Object Models, wie z. B. Klassen- und Schnittstellen-Identifikation.

Hilfsmodelle

OIM enthält 3 Modelle die als Hilfsmittel in den meisten anderen Modellen verwendet werden: das UML Extension Model (UMX), das Datatype Model (DTM) und das Generic Model (GEN).
UMX enthält einige Konzepte, die sich zwar auf derselben Abstraktionsebene wie UML befinden, aber nicht Bestandteil des UML-Standards sind. Dazu gehören z. B. Punkte, Linien, grafische Elemente oder Zeichensätze. Diese Konzepte sind in dem UMX Modell gesammelt.
UML enthält keine Elemente, mit denen man allgemein verwendete Datentypen modellieren kann. Das Modell DTM enthält solche Elemente, mit denen man atomare Datentypen wie String, Integer usw. oder auch komplexere Datentypen wie z. B. Felder, Aufzählungen u.a. darstellen kann.
Das Generic Model (GEN) ist eine Sammlung von Konzepten, die in vielen Teilen des OIM angewendet werden können, aber keinem der Untermodelle eindeutig zugeordnet werden können. Hier befinden sich z. B. Modelle für Hilfsinformationen, Menüs, Grafische Oberflächen u.v.a.

4. Zusammenfassung und Schlußbemerkung

Repositories sind Datenbankanwendungen, mit denen die Portierung von Metadaten oder Anwendungselementen (Komponenten) zwischen verschiedenen Systemen verwaltet wird. Die Bedeutung der Metainformation (und der Metainformationsverarbeitung) wächst ständig. Sie spielt heute im Kontext von Programmiersprachen wie Java, im Zusammenhang mit Componentware und CORBA eine ebenso bedeutsame Rolle wie auf dem Gebiet des Data Warehousing und der OLAP (Online Analytical Processing-Anwendungen) oder der Entwicklung von W3-Anwendungen für das „e-Business" [Ort99]. Repositories bieten in heterogenen Umgebungen eine effektive Integrationsbasis, indem sie es Anwendungen erlauben, Metadaten auszutauschen.

Indem das Microsoft Repository bereits vordefinierte Informationsmodelle enthält, unterstützt es diese Standardisierungsbemühungen von Repositories. Da das Open

(das I steht für Interface). Im Folgenden sollen eine Reihe wichtiger Untermodelle ausführlicher vorgestellt werden.

Unified Modeling Language (UML)

Das wichtigste Untermodell des Open Information Model, von dem alle anderen Modelle abgeleitet sind, ist dasjenige, das die Unified Modeling Language (UML) spezifiziert. UML ist ein semantisches Modell und eine grafische Notation, die der Analyse, Spezifikation, Design, Konstruktion und Visualisierung von Software-Artefakten dient.

UML wird bereits von einer großen Anzahl von Modellierungswerkzeugen verwendet. Daher bietet sich ihre Integration in das Open Information Model an, da viele potentielle Nutzer des Repositories somit ihre Daten leicht anpassen können. Darüberhinaus ist UML so umfangreich und auf einem so hohen Abstraktionsniveau, daß es sich sehr gut als Grundmodell eignet, von dem andere Modelle durch Vererbung und Spezialisierung abgeleitet werden können. Andere Modelle im OIM von UML abzuleiten hat mehrere Vorteile:

- Es verringert den Umfang und die Komplexität des OIM, indem viele Konzepte und Elemente von UML in Untermodellen wiederverwendet werden. So stellt etwa die IUmlPackage-Owns-IUmlElement Beziehung eine allgemeine Form der Enthaltenseins-Relation dar. In Untermodellen wird sie spezialisiert, etwa im Datenbankenmodell (Dbm). Dort wird sie verwendet, um die Relation von Katalogen (IDbmCatalog) zu modellieren, die Schemata (IDbmSchema) und Tabellen (IDbmTable) enthalten können.
- Es vereinfacht den Austausch zwischen verschiedenen Arten von Werkzeugen. Eine sinnvolle Funktion etwa, besteht in dem Erzeugen von Klassendefinitionen aus Datenbankinhalten. Dies erfordert gewöhnlich das Übersetzen der Definition einer Tabelle in Einzelheiten einer Klasse. Da im OIM eine Tabelle (IDbmTable) und eine Klassendefinition (IComClass) von demselben Element (IUmlElement) abgeleitet sind, ist es möglich Elemente der Tabellendefinition automatisch als Elemente der Klassendefinition zu interpretieren.

Database Model (DBM)

Eines der wichtigsten Anwendungsgebiete des Microsoft Repositories besteht in der Unterstützung des Entwurfs von Datenbankschemata. OIM enthält daher eine Reihe von Untermodellen die Datenbankanwendungen unterstützen. Das zentrale Modell, von dem alle Datenbankmodelle abgeleitet sind, ist das Database Model (DBM).

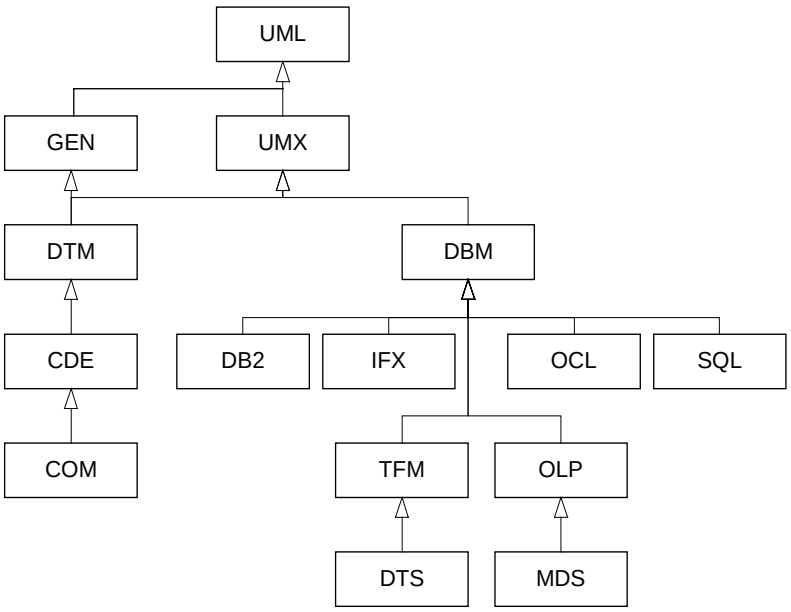
DBM enthält eine Reihe von Konzepten aus SQL und ähnlichen Datenmodellen. Z. B. sind Modelle enthalten für: Tabellen, Sichten, Bedingungen, Trigger und Spalten. All diese Konzepte sind als Spezialisierung von UML-Elementen abgeleitet. Bedingung etwa wird von der UML-constraint-Klasse abgeleitet. Trigger sind abgeleitet von UML-Methoden usw.

DBM wird darüberhinaus noch weiter spezialisiert, um die Eigenschaften einiger weit verbreiteter Datenbanken modellieren zu können, namentlich DB2 von IBM, Informix, Microsoft SQL Server und Oracle. Diese Untermodelle sind jedoch nicht sehr

Konfigurationsverwaltung. Die Versionsverwaltung erfüllt die vorgestellten Anforderungen und enthält einen komplexen und anpassbaren Mechanismus zur Vereinigung verschiedener Versionen. Über die genaue Realisierung der Konfigurationsverwaltung macht [Ber99] allerdings keine Aussagen. Die Möglichkeit der Notifikation ist im MSR nicht vorgesehen. Auch eine Prozeßsteuerungs-komponente ist nicht enthalten.

3.4. Das Open Information Model

Im Microsoft Repository ist bereits eine Reihe von Informationsmodellen integriert, die Anwender entweder unverändert verwenden, oder ihren speziellen Bedürfnissen anpassen und erweitern können. Zusammen bilden diese Informationsmodellen das sogenannte Open Information Model (OIM). Es besteht aus momentan etwa 250 Schnittstellendefinition und etwa 100 Beziehungsdefinitionen. Diese sind in 15 Untermodelle aufgeteilt, die die Bereiche Objektmodellierung, Datenbank-modellierung und (COM-) Komponentenwiederverwendung abdecken. Die folgende Abbildung stellt die Hierarchie der Modelle im Open Information Model dar.



UML	Unified Modeling Language
UMX	UML Extensions
GEN	Generic
DTM	Data Type Model
CDE	Component Description
COM	Component Object Model
DBM	Database Model
DB2	IBM DB2
IFX	Informix
OCL	Oracle
SQL	Microsoft SQL Server
TFM	Transformations
DTS	Microsoft Data Transformation Service
OLP	OLAP
MDS	Microsoft Decision Support

Abbildung: Hierarchie der Modelle im Open Information Model

Das Ziel des Open Information Model besteht darin, Austausch und Wiederverwendung von Metadaten zu ermöglichen. Daher ist es notwendig, daß das Modell eine gute Unterstützung bietet für einen möglichst großen Anteil der Hersteller von Anwendungen, die das Repository verwenden. Um das zu erreichen, werden die Untermodelle in Zusammenarbeit mit wichtigen Herstellern auf den jeweiligen Gebieten erarbeitet. Außerdem basieren die Modelle auf bereits akzeptierten und vielfach eingesetzten Industriestandards, wie z. B. UML.

Um das Auffinden von Schnittstellen zu vereinfachen, die einer bestimmten Domäne angehören, enthalten die Namen der Schnittstellen jeweils einen Präfix aus drei Buchstaben, der die Zugehörigkeit zu einem bestimmten Untermodell signalisiert. In dem Untermodell zur Datenbankmodellierung etwa, beginnen die Namen mit IDbm

funktioniert dann so, daß zunächst der jüngste Vorfahre beider Objekte gesucht wird. Werte, die in nur einer Version verändert wurden, werden auch in dem resultierenden Objekt geändert. Wenn ein Wert in beiden Versionen geändert ist, dann wird der Wert der dominierenden Version übernommen.
Dieser Algorithmus zur Vereinigung zweier Versionen kann allerdings von Anwendungen überschrieben und den jeweiligen Bedürfnissen angepaßt werden.

Da in der ersten Version des Repositories keine Versionierung unterstützt wurde, ist ein Mechanismus nötig, der bei Zugriffen auf das Repository über die Schnittstellen der ersten Version auflöst, welche Version eines Objektes zurückgegeben werden soll. Das ist möglich, da die sich die Methoden, mit denen man auf Versionen zugreifen kann, gegenüber den alten Schnittstellen orthogonal verhalten. D.h. daß in der zweiten Version die Objekte im Repository nur eine weitere Schnittstelle implementieren, durch die auf die Versionen zugegriffen werden kann. Es ist aber weiterhin möglich auf die Objekte zuzugreifen, ohne von der zusätzlichen Dimension der Versionen Gebrauch zu machen.

Kontextverwaltung

Die Version 2 des Microsoft Repositories unterstützt ein Kontextverwaltung in Form sogenannter Workspaces. Ein Workspace, eine Arbeitsumgebung also, erlaubt es mehreren Anwendern des Repositories, Objekte, auf die sie alleine zugreifen, zu isolieren und später in einer kontrollierten Weise wieder zusammenzuführen.

Ein Workspace ist ein virtuelles Repository, das eine Teilmenge der Objekte aus dem eigentlichen Repository aufnehmen kann. Ein Workspace ist dabei selbst nur ein Objekt im Repository. Allerdings ist es nicht möglich, Workspaces ineinander zu verschachteln. Versionen eines Objektes aus dem Repository werden explizit einem Workspace hinzugefügt oder daraus gelöscht. Ein Workspace kann immer nur eine Version eines Objektes aufnehmen.

Während ein Objekt einem bestimmten Workspace zugeordnet ist, wird auf dieses Objekt im Repository eine Sperre gesetzt. Das bedeutet, daß während dieser Zeit kein anderer Benutzer auf dieses Objekt zugreifen kann. Wird das Objekt wieder aus dem Workspace entfernt, dann wird auch die Sperre wieder aufgehoben. An dieser Stelle wird also eine Form des Checkout/Checkin eingesetzt.

Bewertung

Abschließend soll noch kurz erörtert werden, in welchem Maße das Microsoft Repository die allgemeinen Anforderungen erfüllt, die im ersten Abschnitt formuliert wurden. Die Schnittstellen, über die externe Werkzeuge auf das Repository zugreifen können, sind beim MSR objektorientiert realisiert, d. h. daß Werkzeuge auf das MSR wie auf eine objektorientierte Datenbank zugreifen können. Die Kontextverwaltung ist durch die Workspaces realisiert. Das Hinzufügen bzw. das Löschen einzelner Objekte zu einem Workspace entspricht dem Checkout/Checkin-Vorgang. Im MSR ist dieser so realisiert, daß auf ein Objekt immer nur von ein Benutzer zugreifen kann. Allerdings können verschiedene Benutzer auf verschiedene Versionen des Objektes zugreifen. Die zweite Version des MSR unterstützt eine Versions- und

3.3. Dienste des Microsoft Repositories

In den folgenden Abschnitten sollen einige der grundlegende Dienste, die das Microsoft Repository zur Verfügung steht, näher erläutert werden. Neben der Navigation innerhalb der Objekte, werden die Versions- und die Kontextverwaltung beschrieben.

Navigation in den Repository-Objekten

Um das Repository zu verwenden, erzeugt man zunächst eine Instanz der Klasse Repository Session. Mit der Methode *Create* kann dann eine neue Repository-Datenbank erzeugt werden oder mit der Methode *Open* eine bereits bestehende Datenbank geöffnet. Anschließend ist das Objekt an ein Repository gebunden und man kann auf dessen Objekte zugreifen und neue Objekte darin erzeugen.
Die *CreateObject* -Methode erzeugt ein neues Objekt einer bestimmten Klasse. Auf bereits vorhandene Objekte kann auf mehrere Arten zugegriffen werden:

- *Create* bzw. *Open* geben das eindeutige Root-Objekt in einem Repository zurück. Alle anderen Objekte sind direkt oder indirekt mit diesem Objekt verbunden.
- Die *ObjectInstances* -Methode erzeugt eine Kollektion aller Objekte einer bestimmten Klasse oder aller Objekte, die eine bestimmte Schnittstelle implementieren.
- Man kann Objekte anhand ihrer Identifikation im Repository suchen
- Man kann auf den Objekten eine SQL-Anfrage formulieren
- Man kann den Beziehungen, die von einem Objekt ausgehen, folgen, um andere Objekte zu erreichen.

Versionsverwaltung

Das Microsoft Repository unterstützt die Versionierung von Objekten. Die verschiedenen Versionen eines Objektes sind durch eine Nachfolger-Relation miteinander verknüpft. Jede Version hat darüber hinaus eine eindeutige Versions-Identifikation.

Jedes Repository-Objekt kann einen von zwei Zuständen haben: *frozen* oder *unfrozen*. Eine neue Version eines Objektes kann nur dann erzeugt werden, wenn das Objekt im Zustand frozen ist, d. h. es können keine weiteren Änderungen mehr an dem Objekt vorgenommen werden. Indem man verlangt, daß die Objekte „eingefroren" werden, also ihr Zustand festgehalten wird, genügt es bei der Erzeugung einer neuen Version, nur die Teile abzuspeichern, die sich auch tatsächlich gegenüber dem Vorgänger geändert haben.

Ein Objekt kann mehr als einen Nachfolger haben. Das bedeutet, daß der Graph, dessen Knoten die Versionen und dessen Kanten durch die Nachfolger-Relation gebildet werden, auch mehrere parallele Äste haben kann. Natürlich enthält dieser Graph aber keine Zyklen. Verschiedene Äste können durch eine Merge-Operation wieder zusammengeführt werden. Diese verlangt zum einen ein Objekt im einen Ast, daß sich im Zustand unfrozen befindet, zum anderen eine „gefrorene" Version des anderen Astes. Außerdem wird über einen Flag angezeigt, welche der beiden Versionen, die vereinigt werden sollen, dominiert. Der Vereinigungsvorgang

3.2. Das Objektmodell des Microsoft Repositories

COM wird von der überwiegenden Mehrzahl von Programmen für Microsoft Betriebssysteme als Objektmodell verwendet. Daher ist es erforderlich, daß auch die Funktionalität des Repositories als COM-Objekte zur Verfügung gestellt werden.

Das Microsoft Repository untersützt vier Arten von Objekten:

- *Repository Session* – modelliert das Repository selbst. D. h. um auf das Repository zuzugreifen, erzeugen Anwendungen eine Instanz dieser Klasse, die an das jeweilige Repository gebunden ist. Über dieses Objekt können sie auf das Repository zugreifen.
- *Repository Object* – modelliert den persistenten Zustand eines Objekts im Repository.
- *Relationship Object* – modelliert eine Beziehung zwischen zwei Objekten im Repository. Eine solche Beziehung kann selbst Datenfelder enthalten. Die jeweiligen Endpunkte einer Beziehung, sowie ihre Datenfelder sind im Repository gespeichert.
- *Collection Object* – modelliert eine Kollektion. Diese Klasse wird verwendet, um die Kollektion der Verbindungen, die von einem Objekt ausgehen, zu fassen. Sie implementiert die oben beschriebenen Methoden einer Kollektion: Add, Count, Remove, Item und _NewEnum.

Das Repository ermöglicht es dem Benutzer, ein eigenes Informationsmodell wie oben beschrieben zu definieren und zu bearbeiten. Informationsmodelle sind eine Beschreibung der Struktur und der Semantik von Daten. Der Repository Manager stellt Methoden zur Verfügung, mit denen man Objekte erzeugen und speichern kann, die durch das Informationsmodell beschrieben werden. Die Elemente eines Informationsmodells (Klassen, Schnittstellen, Datenfelder, Methoden, Beziehungen und Kollektionen) sind dabei selbst gewöhnliche Objekte im Repository. So ist etwa eine Klassendefinition ein normales Repository-Objekt, daß z. B. das Datenfeld globale Identifikation enthält, sowie eine Beziehung zu den Schnittstellen, die durch die Klasse implementiert werden. Durch das Typmodell werden die zulässigen Elemente eines Informationsmodells beschrieben.

Das Typmodell des Microsoft Repositories besteht aus den folgenden Klassen:

- *ClassDef* – modelliert die Definition einer Klasse. Implementiert die Schnittstellen IReposTypeInfo, die allgemeine Informationen über ein Repository-Objekt enthält und die Schnittstelle IClassDef.
- *RelationshipDef* – repräsentiert eine Beziehung zwischen zwei Klassen.
- *InterfaceDef* – modelliert eine Schnittstelle, die Datenfelder und Methoden enthält.
- *PropertyDef* – modelliert ein Datenfeld.
- *CollectionDef* – definiert das Verhalten einer Kollektion. Z. B. kann man die minimale und maximale Kardinalität einer Kollektion festlegen.
- *MethodDef* – modelliert eine Methode, d.h. eine Funktion innerhalb einer Klasse.

Entwicklern geteilt werden, ist eine effiziente Versions- und Konfigurationsverwaltung nötig.

3.1. Das Component Object Model

Das Component Object Model (COM) bildet die Grundlage für die Objektarchitektur der Betriebssysteme von Microsoft. Es handelt sich um einen binären Standard der die Art und Weise, wie sich Komponenten gegenseitig aufrufen können, beschreibt. Binär bedeutet, daß die Schnittstelle von kompilierten Komponenten definiert ist. Der Standard ist also unabhängig von der verwendeten Programmiersprache und erlaubt, daß Komponenten, die in verschiedenen Sprachen geschrieben sind, miteinander kommunizieren können.

In COM bezeichnet man mit dem Begriff Klasse das Abbild eines ausführbaren Programmes. Eine Klasse kann mehrere Schnittstellen implementieren, wobei man mit dem Begriff Schnittstelle (interface) eine Sammlung von Methoden bezeichnet. Die Schnittstellen des Microsoft Repositories sind solche COM-Schnittstellen.

Jede Klasse und jede Schnittstelle hat eine eindeutige Identifikation. Diese sind in der Registrierungsdatenbank (registry) abgelegt. Mit der Funktion CoCreateInstance kann, indem man die eindeutige Identifikation übergibt, ein Objekt, d. h. die Instanz einer Klasse erzeugt werden. Auch die Schnittstellendefinitionen werden in der Registry abgelegt. Einmal definierte und veröffentlichte Schnittstellen können nicht mehr verändert werden, um sicherzustellen, daß jede Installation eines Microsoft Betriebssystem diesselben Schnittstellendefinitionen kennt. D. h., um Schnittstellen zu ändern oder zu erweitern, muß eine neue Schnittstelle definiert werden.

Jede COM-Klasse implementiert das Interface IUnknown. Diese Schnittstelle enthält die Funktion QueryInterface, die es ermöglicht festzustellen, ob die Klasse ein bestimmtes Interface implementiert, indem man die eindeutige Identifikation der gefragten Schnittstelle übergibt. Eine Klasse kann mehrere Versionen einer Schnittstelle implementieren. Anwendungen, die daß Objekt verwenden, können durch den QueryInterface-Mechanismus feststellen, ob die Schnittstelle, die sie benötigen unterstützt wird. Dadurch ist es möglich, Klassen und die Anwendungen, die sie verwenden, unabhängig voneinander weiterzuentwickeln.

Eine besondere Sorte von Objekten stellen die Kollektionen (collections) dar. Sie können eine Anzahl gleichartiger Objekte aufnehmen. Dazu unterstützen sie die folgenden Methoden:

- *Add* – fügt der Collection ein neues Objekt hinzu
- *Count* – liefert die Anzahl der Objekte zurück
- *Item* – gibt ein bestimmtes Objekt zurück, entweder anhand eines Indexes oder eines Schlüsselwertes
- *Remove* – löscht ein Objekt aus einer Collection
- *_NewEnum* – liefert einen Iterator über die Collection zurück. Dieser kann mit der *Next* - Methode durchlaufen werden.

derungen, Spezifikationen, Klassendiagramme, Datenstrukturen u. ä. Viele Daten durchlaufen dabei einen wohldefinierten, festgelegten Prozeß. D. h. sie besitzen einen Status ihrer Bearbeitung und es ist deterministisch zu entscheiden welche Bearbeitungsstation sie jeweils als nächstes zu durchlaufen haben.

Das Konzept der Prozeßsteuerung (workflow control) kann dabei helfen, diesen Vorgang zu automatisieren. Es wird dazu eine Operation benötigt, mit der man den Status eines Objekts ändern kann. Diese kann sowohl manuell durch den Benutzer ausgelöst werden, oder durch eine Notifikationsregel, wenn z. B. eine bestimmte Bearbeitungsphase abgeschlossen ist.

Es sollte möglich sein, die einzelnen Schritte innerhalb eines Prozesses, sowie die Daten- und Kontrollflüsse innerhalb der Schritte zu beschreiben. Durch eine Steuerungskomponente könnte die Durchführung der einzelnen Schritte überwacht werden. Eine solche Komponente ist natürlich auch unabhängig von einem Repository von großem Nutzen. Allerdings ist es sinnvoll, daß die Beschreibung des Prozesses, bei der es sich ja um Metadaten handelt, in dem Repository abgelegt werden kann.

3. Das Microsoft Repository

Nachdem bisher einige allgemeine Anforderungen und Eigenschaften von Repositories beschrieben wurden, soll in den folgenden Abschnitten nun ausführlicher auf ein bestimmtes, real existierendes Produkt eingegangen werden: das Microsoft Repository. Neben einer Beschreibung einiger grundsätzlicher Ansätze soll dabei darauf eingegangen werden, inwieweit die im vorangegangenen Abschnitt vorgestellten Konzepte tatsächlich realisiert wurden.

Das Microsoft Repository besteht aus zwei Komponenten: Zum einen dem Open Information Model, einer Menge vordefinierter Informationsmodelle, die etwa die Bereiche Objektmodellierung, Datenbankenmodellierung und (COM-)Komponenten-wiederverwendung abdecken. Die zweite Komponente ist der Repository Manager, der eine Reihe objektorientierter Schnittstellen implementiert. Mit seiner Hilfe kann ein Entwickler das bestehende Informationsmodell erweitern oder ein neues entwickeln, sowie Instanzen dieses Informationsmodells verwalten. Der Repository Manager setzt entweder auf Microsoft SQL Server oder auf Microsoft Jet (dem Datenbanksystem in Microsoft Access) auf.
Bernstein [Ber99] definiert die technischen Ziele bei der Entwicklung des Repositories als:

- Kompatibilität zum COM-Objektmodell: Das Objektmodell des Repositories sollte sich natürlich an die Objektarchitektur der Microsoft Betriebssysteme einfügen lassen. Dies ist das Component Object Model (COM). Wenn möglich, verwendet das Repository bereits bestehende COM-Schnittstellen und stellt seine eigene Funktionalität in Form von COM-Objekten zur Verfügung.
- Erweiterbarkeit: Es sollte einfach und effizient möglich sein, das Repository an die individuellen Anforderungen von Kunden anzupassen.
- Version- und Konfigurationsverwaltung: Da die Daten in dem Repository i. A. einer ständigen Weiterentwicklung unterliegen und da sie von mehreren

2.5. Notifikation

Der Notifikation-Dienst dient dem automatisierten Anstoßen weiterer Operationen, die ausgelöst werden sollen, falls an einem bestimmten Repository-Objekt Änderungen vorgenommen wurden oder falls ein bestimmtes Ereignis in dem Repository eingetreten ist.

Dieser Dienst stellt damit zum einen ein Werkzeug der Automatisierung dar: Wurde z. B. ein Modul in einem Softwareprojekt verändert, könnte durch Notifikation automatisch das Neukompilieren der betroffenen Komponente ausgelöst werden. Zum anderen bietet Notifikation die Möglichkeit der Kommunikation zwischen den Benutzer. So könnte etwa ein Kommentar, den ein Benutzer zur Änderung eines Repository-Objekts angegeben hat, automatisch beim Checkin des Objekts an andere Benutzer weitergeleitet werden, die ebenfalls an diesem Objekt arbeiten.

Einige Notifikation-Regeln könnten sinnvollerweise fest in die Steuerung des Repositories eingebaut sein, z. B. die Benachrichtigung anderer Benutzer eines Objekts bei dessen Änderung. Darüber hinaus wird aber auch ein Mechanismus benötigt, Notifikation-Regeln neu zu definieren und in dem Repository einzuführen. Solche Regeln wären im einfachsten Falle von der Form „Wenn Ereignis E eintritt, und Bedingung B gilt, dann führe Operation O aus".

2.6. Kontextverwaltung

Mit dem Begriff Kontext bezeichnet man die logische Arbeitsumgebung eines Benutzers. Ein Kontext ist also eine Teilmenge der Objekte in einem Repository.

Um Objekte in einem Repository zu bearbeiten, muß ein Benutzer zunächst einen Kontext öffnen. Das kann sowohl ein bereits bestehender Kontext sein, aber auch ein neu angelegter. Der Benutzer kann Objekte aus dem Repository in dem Kontext einblenden. Objekte, die innerhalb des Kontexts sichtbar sind, können dann bearbeitet werden. Nachdem die Änderungen an den Objekten abgeschlossen sind, schließt der Benutzer den Kontext wieder. Dieser kann später wieder von demselben oder einem anderen Benutzer weiterverwendet werden.

Das Kontext-Konzept kann von Vorteil sein, um die Komplexität und die Anzahl der Objekte die ein Benutzer bearbeiten kann, zu beschränken. Dadurch wird ein einfacheres und übersichtlicheres Bearbeiten der Objekte ermöglicht. Außerdem ist es natürlich ein einfacher Mechanismus, um Objekte vor versehentlichem oder unerlaubtem Ändern zu schützen – Objekte, die sich nicht in dem Kontext eines Benutzers befinden, können von diesem auch nicht verändert werden.

2.7. Workflow control – Prozeßsteuerung

Die Daten die in einem Repository gespeichert sind, sind typischerweise dynamische Daten. D. h., während der Dauer ihres Bestehens durchlaufen sie eine Reihe von Änderungen und Weiterentwicklungen. Metadaten, die während eines Softwareprojektes entstehen und ständig weiterentwickelt werden, sind z. B. die Anfor-

- Versionen eines Objekts sollen als jeweils einzelne Objekte im Repository gespeichert werden.
- Möglichkeiten, die Versionen eines Objektes zu benennen, sollten vorhanden sein. Dies kann entweder durch Standardwerte oder durch vom Benutzer vorgegebene Werte geschehen.
- Es sollten Operationen vorhanden sein, die eine neue Version eines Objektes erzeugen. Dabei muß festzulegen sein, welche Elemente der alten Version in welcher Weise in die neue Version einfließen.
- Die Möglichkeit sollte bestehen, eine bestimmte Version eines Objektes zu identifizieren und in einem Kontext zu verwenden.
- Die Checkout-Operation kann alternativ zur oben vorgestellten Lösung so gestaltet werden, daß dabei automatisch eine neue Version eines Objekts erzeugt wird. Beim Checkin ist dann festzulegen, ob verschiedene Versionen zu vereinigen sind, oder ob eine bestimmte Version Vorrang hat.
- Mehrere Versionen eines Objekts, die nebeneinander existieren können, sollten zu einem Objekt vereinigt werden können. Dabei muß spezifiziert werden, wie mit Konflikten zwischen den Versionen verfahren werden soll. Möglich wäre etwa, daß einer Version Vorrang vor der anderen gegeben wird. Falls die Domäne der Metadaten genügend genau spezifiziert ist, könnte aber auch eine andere sinnvolle Vereinigungsoperation definiert werden. Am besten sollte dieses Verhalten auch einstellbar sein.

2.4. Konfigurationsverwaltung

In einem Repository kann es Objekte geben, die eine hierarchische Struktur haben, d. h. die aus anderen Objekten aufgebaut sind. Wie jedes Objekt in einem Repository kann auch ein solches zusammengesetztes Objekt versioniert sein. Mit dem Begriff Konfiguration bezeichnet man nach [BDa94] die Bindung bestimmter Versionen der Komponenten an eine Version des zusammengesetzten Objektes. Natürlich müssen nicht alle Komponenten versioniert sein, auch nicht-versionierte, einfache Objekte können Komponenten einer Konfiguration sein.

Nach [BDa94] werden die folgenden Anforderungen an einen Mechanismus zur Verwaltung der Konfigurationen gestellt:

- Konfigurationen sollten im Repository darstellbar und veränderbar sein.
- Die Versionen der Komponenten einer Konfiguration sollten festzulegen sein. Dies kann sowohl explizit geschehen, indem man etwa die Namen der Versionen angibt. Oder implizit, indem die zu verwendenden Versionen durch Regeln festgelegt werden, z. B. könnte man verlangen, daß immer die neuesten Versionen der Komponenten verwendet werden.
- Operationen sollten vorhanden sein, um Versionen einer Komponente an eine Konfiguration zu binden, oder sie aus ihr zu löschen
- An Konfigurationen sollten Bedingungen formulierbar sein. So könnte z. B. verlangt werden, daß alle Komponenten desselben Typs in einer Konfiguration auch von derselben Version sind.
- Festzulegen ist ferner, wie sich die Änderung der Version einer Komponente auf die Version der Konfiguration auswirkt. Dabei kann etwa verlangt werden, daß bei einer Änderung einer Komponente eine neue Version der Konfiguration erzeugt wird.

Systemabsturz während einer solcher Aktivität etwa, könnte die Arbeit mehrerer Tage zunichte machen.

Daher sollte der Repository Manager die Möglichkeit eines Checkout und eines Checkin von Repository-Objekten vorsehen. Die Checkout-Operation erzeugt eine lokale Kopie eines Repository-Objekt in der Arbeitsumgebung eines Benutzers. Nachdem die Bearbeitung des Objekts abgeschlossen ist, führt der Benutzer eine Checkin-Operation auf dem Objekt aus, wodurch es wieder in das Repository zurückgeschrieben wird. Die Checkout- und die Checkin-Operation werden dabei als separate Transaktionen ausgeführt. Bei einem Checkin wäre die Möglichkeit sinnvoll, einen Kommentar zu den erfolgten Änderungen anzugeben, der anderen Benutzer beim erneuten Checkout angezeigt wird oder der automatisch weitergeleitet wird (vgl. Notifikation).

Ein weiterer Vorteil von Checkout/Checkin-Operationen besteht darin, daß Benutzer auf lokalen Kopien von Repository-Objekten arbeiten können. Der Zugriffsweg zu dem bearbeiteten Objekt wird dadurch also verkürzt.

Durch den Checkout-Vorgang wird eine lokale Kopie des Repository-Objektes erzeugt. Es müssen also Vorkehrungen getroffen werden, um Inkonsistenzen zwischen mehreren Kopien desselben Objektes zu verhindern. Eine Möglichkeit, diesem Problem zu begegnen, besteht darin, daß auf das angeforderte Objekt eine temporäre Sperre im Repository gesetzt wird. Diese kann sowohl eine Lesesperre sein, d. h. alle Anfragen die nur lesenden Zugriff verlangen, können bedient werden. Oder es handelt sich um eine Schreibsperre, d. h. daß während der Bearbeitung keine anderen Anfragen auf dieses Objekt bedient werden.

Ein anderer Ansatz den Checkout-Vorgang zu realisieren bestünde darin, mehreren Anfragen gleichzeitig schreibenden Zugriff auf ein Objekt zu gestatten und eventuelle Änderungen im Repository zu vereinigen. Dies setzt allerdings voraus, daß genaue Regeln für die Vereinigung von Änderungen definiert sind, was wiederum ein großes Wissen über die Objekte im Repository bedingt.

2.3. Versionsverwaltung

Repositories werden in Bereichen wie z. B. der Softwareentwicklung oder dem Data Warehousing eingesetzt, um typischerweise dynamische Daten zu speichern. Daten also, die während der Zeit ihres Bestehens eine Reihe von Änderungen und Weiterentwicklungen durchlaufen. Mit dem Begriff „Version" bezeichnet man dabei einen Schnappschuß eines Objektes.

In dem Repository werden die verschiedenen Versionen eines Objektes wie ein eigenes Objekt behandelt und abgelegt. Darüber hinaus speichert das Repository auch die Abfolge zwischen den verschiedenen Versionen eines Objektes. D. h., es ist möglich, die Vorgänger bzw. Nachfolger eines Objektes zu bestimmen.

Nach [BDa94] sollte ein Repository, u. a. die folgenden Anforderung in bezug auf die Versionsverwaltung erfüllen:

Die Verwendung eines zentralen Repositories vereinfacht den Datenaustausch zwischen den Komponenten entscheidend: Würde jede der n Komponenten ein eigenes, spezifisches Datenformat verwenden, dann wäre es nötig, zum Datenaustausch zwischen den Komponenten über n^2 Konvertierungsmodule zu verfügen. Indem man ein gemeinsames, kanonisches Datenmodell verwendet, verringert sich dieser Aufwand schon immerhin auf O(n). Den größten Freiheitsgrad erreicht man aber, indem man Daten über die verwendeten Datenformate in einem Repository speichert, zu dem alle Komponenten Zugriff haben.

Daneben hat das Konzept, alle Daten über eine zentrale Komponente anzusprechen, auch noch weitere Vorteile:

- die Daten befinden sich alle an einem wohldefinierten Ort: im Repository. Sie sind nicht über mehrere Datenbanken oder Dateisysteme verstreut.
- jedes gemeinsam verwendete Datenobjekt existiert in nur genau einer Kopie. Es können keine Inkonsistenzen zwischen mehreren Kopien eines Objektes entstehen.
- wird ein Objekt von mehreren Werkzeugen verwendet, kann keine Information verloren gehen. Teile eines Objektes, die von einem Werkzeug nicht benutzt werden, bleiben einfach unverändert im Repository stehen.
- auf alle Datenobjekte wird auf dieselbe Art und Weise zugegriffen.
- zu den Datenobjekten kann die Reihenfolge der Zugriffe aufgezeichnet werden.

In jedem Falle aber muß die Implementierung eines Repositories bestimmte Mechanismen zur Verfügung stellen, die den Zugriff auf die Inhalte und Dienste des Repositories aus anderen Programmen heraus ermöglichen.

Alle Komponenten einer Umgebung, d. h. alle Werkzeuge und das Repository, definieren sich über ihre Schnittstellen. Die Schnittstelle des Repositories muß den Werkzeugen bekannt sein, damit diese die richtigen Methoden des Repositories aufrufen können und die Rückgabewerte richtig interpretieren. Um die Integration von Werkzeugen und Repositories zu vereinfachen, wäre es wünschenswert, daß alle verfügbaren Repositories die gleiche API (application programming interface) unterstützen, was jedoch in der Realität nicht der Fall ist.

An die API eines Repositories sind eine Reihe von Anforderungen zu stellen [BDa94]. Insbesondere sollten in einer objektorientierten API eines Repositories die folgenden Punkte vorgesehen werden:

- Es sollten Klassen enthalten sein, die Eigenschaften und Funktionen des Repositories modellieren, z. B. Versionen, Konfigurationen, Regeln usw.
- Es sollten Methoden zur Verfügung stehen, um Repository-Objekte in die Objekte der jeweiligen Anwendung zu transformieren.
- Die Möglichkeit, zwischen einzelnen Objekten zu navigieren, sollte vorhanden sein (object-at-a-time)

2.2 Unterstützung von Checkout/Checkin

Aktivitäten, die auf ein Repository zugreifen, können von sehr langer Dauer sein, möglicherweise einige Tage oder sogar Monate. Eine vollständige Aktivität als eine einzelne Transaktion zu behandeln, erscheint daher unzweckmäßig. Ein

Ein Repository besteht aus dem so genannten Repository Manager (auch Repository Engine genannt) und einem Informationsmodell. Der Repository Manager ist eine Softwareschicht, die auf einem Datenbanksystem aufsetzt, und dieses um eine bestimmte Semantik, sowie bestimmte Dienste erweitert. Wenn ein Objekt aus dem Repository von einem Benutzer oder einer Softwarekomponente angefordert wird, stellt der Repository Manager fest, wo dieses Objekt gespeichert ist und kopiert das Objekt dann in die jeweilige Arbeitsumgebung, z. B. in ein lokales Dateisystem oder eine Datenbank.

Der Austausch von Metadaten mittels eines Repositories setzt natürlich voraus, daß eine Vereinbarung über die Struktur und die Semantik der Metadaten besteht. Das verwendete Informationsmodell stellt die exakte Spezifikation einer solchen Vereinbarung dar. Durch das Informationsmodell wird sichergestellt, daß alle Benutzer eines Repositories die Daten, die sie austauschen wollen, auf dieselbe Weise kodieren und interpretieren [MSDN]. Der Begriff Informationsmodell steht im Kontext der Repositories also für das Schema [Ber98].

Offensichtlich muß das Informationsmodell an die jeweilige Anwendungsdomäne angepaßt sein. Anhand des Open Information Models, daß vom Microsoft Repository verwendet wird, soll später ausführlich ein Informationsmodell vorgestellt werden, das der Objektmodellierung, der Datenbankmodellierung und Komponentenwiederverwendung dient.

Neben den konventionellen Datenbankdiensten, wie z. B. dem Wiederanlauf im Fehlerfall, Zugriffsschutz, Gewährleistung der Datenintegrität oder einer SQL-Schnittstelle, stellt das Repository weitergehende Modellierungsfunktionen zur Bearbeitung, Verwaltung und Weiterentwicklung der Repository-Objekte zur Verfügung. Dazu gehören neben den Schnittstellen zur Integration von Werkzeugen, die Unterstützung von Checkin/Checkout, Versionsverwaltung, Konfigurationsmanagement, Notifikation, Kontextverwaltung und Workflow control [Ort99][BDa94]. Diese Konzepte sollen in den folgenden Abschnitten ausführlicher beschrieben werden.

2.1. Schnittstellen zur Integration von Werkzeugen

Es ist denkbar, daß ein Repository nicht zur Integration verschiedener Werkzeuge eingesetzt wird, sondern nur aufgrund der Dienste, die es einer Datenbank hinzufügt, in Verbindung mit einer einzelnen Softwarekomponente verwendet wird. Die Erweiterung einer solchen Umgebung durch weitere Komponenten ist jedoch wahrscheinlich.

In einer Umgebung heterogener Softwarekomponenten, wie sie in der Einleitung beschrieben ist, kann ein Repository den Datenaustausch zwischen den einzelnen Komponenten unterstützen und entscheidend vereinfachen, indem es für alle Komponenten ein gemeinsames Informationsmodell, sowie eine Menge gemeinsam nutzbarer Dienste zur Verfügung stellt. Innerhalb der integrierten Umgebung, die aus einer dynamischen Menge von Werkzeugen besteht, ist das Repository dann eine zentrale Komponente, auf die alle Werkzeuge zugreifen können [BDa94].

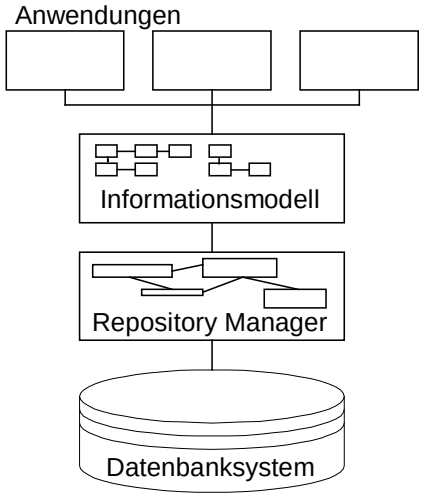
Protokoll angepaßt ist. Dies erfordert möglicherweise einen aufwendigen Eingriff in den Kode der Komponente. Wünschenswert wäre an dieser Stelle ein Mechanismus über den die Komponenten von der Umgebung Informationen über das verwendete Protokoll und Datenmodell erfragen können, und über den sie Daten über die gemeinsam verwendeten Objekte und Schnittstellen erhalten können.

In den beiden vorgestellten Beispielen wird also ein Mechanismus benötigt, mit dessen Hilfe man Metadaten beschreiben und verwalten kann. Ein Repository dient genau dieser Aufgabe: Es bietet die Möglichkeit, die Struktur und die Semantik von Metadaten zu modellieren. Anwendungen, denen es diese Informationen zur Verfügung stellt, werden dadurch in die Lage versetzt, Daten auszutauschen, da sie die Daten auf dieselbe Weise kodieren und interpretieren.

Im folgenden Kapitel soll zunächst allgemein auf Konzepte eingegangen werden, die durch ein Repository realisiert werden. Einige Eigenschaften eines Repositories werden einzeln ausführlich beschrieben. Diese sind: Schnittstellen zum Metadatenaustausch und zur Integration von Werkzeugen (2.1), Unterstützung von Checkin/Checkout (2.2), Verwaltung verschiedener Versionen der Objekte im Repository (2.3), Verwaltung von Konfigurationen hierarchischer Objekte (2.4), Möglichkeiten der Notifikation (2.5), Verwaltung von Arbeitsumgebungen (Kontextverwaltung) (2.6), Unterstützung von Workflow control (2.7).
Im 3. Kapitel folgt dann eine Betrachtung des Microsoft Repositories. Neben der Beschreibung des Component Object Models (3.1), soll auf einige allgemeine Dienste des Repositories eingangen werden (3.2). Dabei wird untersucht, inwieweit die allgemeinen Konzepte im Microsoft Repository tatsächlich realisiert wurden bzw. auf welche Weise. Im letzten Abschnitt (3.3) wird noch einmal das vom Microsoft Repository verwendete Informationsmodell Open Information Model ausführlich beschrieben.

2. Was ist ein Repository?

Ein Repository ist ein computerunterstütztes Informationssystem, welches Daten *über* Produkte und Prozesse einer bestimmten Domäne enthält. Es bietet die Möglichkeit, Daten über Artifakte, Komponenten, Dokumente und Verfahren, sowie Daten über deren Zusammenwirken und über deren Beziehungen *auf einer Metasprachebene* zu beschreiben und zu verarbeiten. Ein Repository dient also der persistenten Speicherung und der Verwaltung und Weiterverarbeitung von Metadaten.



1. Einleitung: Wozu Repositories?

Repositories markieren den derzeitigen Höhepunkt der mehr als 20jährigen Entwicklung von „Data-Dictionary-Systemen" [Tan94]. Es handelt sich dabei um Datenbankanwendungen, die es erlauben unterschiedliche Objekte wie z. B. Sprachartefakte, Datenstrukturen und Programmkomponenten auf einer Metasprachebene zu beschreiben. Darüber hinaus bieten sie umfangreiche Modellierungsfunktionen mit denen die Dokumentationsstruktur der Repository-Objekte verwaltet und weiterentwickelt werden kann. Bevor die technischen Details im einzelnen besprochen werden, sollen im Folgenden zunächst anhand zweier beispielhafter Szenarien mögliche Einsatzgebiete von Repositories aufgezeigt werden.

1. Beispiel: Data Warehousing

Jeden Tag erzeugen große oder kleine Firmen viele Millionen Bytes an Daten über alle Aspekte ihres Geschäftsbereiches. Doch nur ein kleiner Teil dieser Daten war bislang den leitenden Angestellten und der Geschäftsführung unmittelbar zugänglich. In jüngster Zeit, haben sich eine Reihe neuer Konzepte und Werkzeuge zu einer Technologie entwickelt, die das Ziel verfolgt die Verantwortlichen und die Entscheider innerhalb einer Firma mit dem nötigen Zugriff auf diese Daten zu versorgen [Orr96]. Diese neue Technologie wird mit dem Begriff „data warehouse" bezeichnet. Man versteht darunter die Bereitstellung aller Daten über eine Unternehmung, so daß die Verantwortlichen darauf unbegrenzten Zugriff haben und daraus die entscheidungsrelevanten Daten selbst extrahieren können. Diese Daten sind zum einen sehr verschiedenartig im Bezug auf ihre Struktur und die Semantik ihrer Ausprägungen. Andererseits sollte es möglich sein, auf eine einheitliche und einfache Art und Weise auf alle Daten zuzugreifen. Diese Anforderungen sind dadurch zu erfüllen, daß alle Daten auf einer gemeinsamen Metaebene beschrieben werden. Es ist also ein Mechanismus nötig, der eine solche Beschreibung gestattet und der idealerweise auch über die Möglichkeit verfügt, daß verwendete Informationsmodell weiterzuentwickeln und an die Bedürfnisse des jeweiligen Unternehmens anzupassen.

2. Beispiel: Komponentenintegration

Im Bereich der Softwareentwicklung gewinnt der sogenannte komponentenbasierte Ansatz zunehmend an Bedeutung. Komponenten werden dabei als funktionale Einheiten, d. h. unabhängige, kombinierbare Softwarebausteine beim Zusammenbau großer Programme eingesetzt [Jae99]. Ein Problembereich des komponentenbasierten Ansatzes stellt offensichtlich die Integration der Komponenten dar, d. h. welche Möglichkeiten und Techniken vorgesehen werden, um die Komponenten in die Lage zu versetzen, untereinander zu kommunizieren, also Daten auszutauschen. Eine Möglichkeit der Integration der Werkzeuge stellt nach [Val95] die so genannte Black-Box-Integration dar. Dabei stellt die Umgebung ein Protokoll bereit, welches das Werkzeug wie einen Umschlag umschließt [Hof98]. Dieser Ansatz bietet das höchste Maß an Unabhängigkeit zwischen der Umgebung und den zu integrierenden Komponenten. Eine Komponente, die das Protokoll der Umgebung versteht und verarbeiten kann, kann dabei problemlos in die Umgebung integriert werden. Allerdings verlangt dieser Ansatz, daß die Komponente an das verwendete

Repository-Technologie am Beispiel des Microsoft Repository

Seminar von Rasmus Hofmann
hofmann@informatik.uni-kl.de

Zusammenfassung

Ein Repository ist ein Informationssystem, das Daten *über* Produkte oder Verfahren aus einer bestimmten Domäne enthält. Im Bereich eines Softwareprojektes etwa, kann in einem Repository die Klassenstruktur des Softwaresystems, Informationen über die benutzten Datenstrukturen oder über den Prozess zur Erstellung des Systems selbst enthalten sein. Ein Repository dient unter anderem dem Austausch dieser Metadaten zwischen verschiedenen Herstellern bzw. zwischen den Produkten verschiedener Hersteller, sowie der Dokumentation eines Produktes oder eines verwendeten Verfahrens.

Ein Repository besteht aus einem sogenannten Repository Manager (auch Repository Engine genannt) und einem Informationsmodell. Bei dem Repository Manager handelt sich um eine Softwareschicht, die auf eine Datenbank aufsetzt und diese um eine bestimmte Semantik, sowie spezielle Funktionalität erweitert. Das Informationsmodell ist ein Objektmodell für die Objekte im Repository. D.h. das Werkzeuge das Informationsmodell nutzen, um den Inhalt des Repositories zu interpretieren [Ber98].

Neben den konventionellen Datenbankdiensten wie z. B. der Gewährleistung der Datenintegrität, dem Zugriffsschutz oder dem Wiederanlauf im Fehlerfall bietet ein Repository weitere, darüber hinausgehende Funktionen, die die Bearbeitung und Weiterentwicklung der Daten im Repository unterstützen.
Diese sind insbesondere:

- Schnittstellen zum Metadatenaustausch und zur Integration von Werkzeugen
- Unterstützung von Checkin/Checkout
- Verwaltung verschiedener Versionen der Objekte im Repository
- Verwaltung von Konfigurationen hierarchischer Objekte
- Möglichkeiten der Notifikation
- Verwaltung von Arbeitsumgebungen (Kontextverwaltung)
- Unterstützung von Workflow control

Diese Begriffe werden zunächst inhaltlich erläutert. Anschließend wird am Beispiel des Microsoft Repositories (MSR) ein existierendes Produkt vorgestellt. Dabei werden einige Dienste, die das MSR zur Verfügung stellt, betrachtet. Abschließend wird noch ausführlich das verwendete Informationsmodell beschrieben.

Repository-Technologie
am Beispiel des Microsoft Repository

Seminararbeit von
Rasmus Hofmann[1]

im Rahmen des Seminars
„Datenbankanwendungen"
durchgeführt durch die
AG Datenbanken und Informationssysteme
im Wintersemester 1999/2000

[1] hofmann@informatik.uni-kl.de