Integrationskonzepte für heterogene Anwendungssysteme bei DaimlerChrysler auf Basis internationaler Standards (Kurzbeitrag)

Stefan Sarstedt[1,2], Günter Sauter[1], Jürgen Sellentin[1,2], Bernhard Mitschang[2]
e-mail: mitsch@informatik.tu-muenchen.de
{Stefan.Sarstedt, Guenter.Sauter, Juergen.Sellentin}@DaimlerChrysler.com

DaimlerChrysler[1]
Forschung & Technologie
Lab. IT für Engineering (FT3/EK)
Abt. Prozeßkette Produktentwicklung
Postfach 2360, D-89013 Ulm

Technische Universität München[2]
Fakultät für Informatik
Forschungs- und Lehreinheit Informatik III
- Datenbanksysteme und Wissensbasen -
Orleansstr. 34, D-81667 München

Kurzfassung: Aufbauend auf den Anforderungen von DaimlerChrysler und unter Verwendung der Standards STEP und CORBA wird eine Architektur und Vorgehensweise für die Integration von Daten und Funktionen heterogener Anwendungssysteme entwickelt. Die eingebrachten Systemkonzepte sowie die dadurch zu erwartende Optimierung des Entwicklungsprozesses werden am Beispiel des Bereiches PKW-Entwicklung diskutiert.

Schlagworte: Funktionsintegration, API-Integration, Heterogenität, STEP, CORBA.

1. Einleitung

Nach wie vor ist heute in den meisten Unternehmen eine heterogene Systemlandschaft vorzufinden. So werden verschiedenartige Hardware-Komponenten, Netzwerksysteme, Betriebssysteme, Datenbanksysteme und Anwendungsprogramme eingesetzt, um den Produktlebenszyklus vom Design über Konstruktion und Produktion bis zur Wartung bzw. Archivierung abzudecken. Seit mehreren Jahren sind heterogene Datenbanken und die Unterstützung von Interoperabilität Schwerpunkt zahlreicher Forschungsarbeiten. Es wurden Konzepte und Prototypen sog. Föderierter Datenbanksysteme oder Multidatenbanksysteme entworfen, um Datenbanken mit unterschiedlichen Datenmodellen und Schemastrukturierungen integrieren zu können. Auch kommerzielle Produkte, die man den sog. Datenbank-Gateways oder Datenbank-Middleware-Produkten zuordnen kann, sind mittlerweile verfügbar [RH98]. Wenngleich noch wichtige Fragestellungen bei der Datenbank-Integration offen sind, so sind doch mittlerweile mächtige Ansätze zur Lösung der essentiellen Probleme auf diesem Gebiet vorhanden [SL90, BE96, Ki95, HBP94, Sa98, Da96].

Ein damit eng verbundenes Problemfeld, das in Zukunft mehr und mehr an Bedeutung gewinnen wird, ist die Integration sog. Anwendungssysteme. Darunter sind Systeme zu verstehen, welche das Datenbanksystem und die zugehörigen Anwendungsprogramme

Veröffentlicht im Tagungsband der 8. GI-Fachtagung BTW '99, Freiburg, März 1999

zusammenfassen und nach außen hin lediglich eine Programmierschnittstelle, ein sog. API (Application Programming Interface), bereitstellen. Eine Datenbankschnittstelle ist durch dieses Konzept der Kapselung damit nicht mehr verfügbar. Hervorzuhebende Stellvertreter für die Kategorie der Anwendungssysteme sind beispielsweise SAP R/3 [SAP98] oder PDM (Produktdatenmanagement)-Systeme wie Metaphase [SDRC98] oder ENOVIAVPM [ENV98]. Aber nicht nur diese kommerziellen Produkte, die oft unter der Bezeichnung "Standardsoftware" zusammengefaßt werden, sind ausschließlich über ein API zugänglich, sondern häufig auch proprietäre, also von den Unternehmen selbst implementierte Software-Lösungen. Denn sowohl die Überwachung von Integritätsbedingungen als auch die Überwachung der Sicherheit (Autorisierung) wird meist durch das Anwendungsprogramm und nicht durch das Datenbanksystem unterstützt. Ferner sind konzeptionelle Schemata und damit ausdrucksstarke Bezeichner nur in seltenen Fällen vorhanden. Damit Schemaänderungen nicht die Konsistenz, die Integrität und den Schutz der Daten gefährden, wird nur ein API für den Zugriff auf die Daten (und die Funktionalität) freigegeben.

Bei DaimlerChrysler (DC), Sparte PKW, werden beispielsweise Daten zu einem Auto u.a. in einem Geometrie-Informationssystem (GIS, Verwaltung der eigentlichen Geometrie), in einem Stammdatenverwaltungssystem[1] (SDVS) und in einem Dokumentverwaltungssystem (DVS, Verwaltung von 2D-Zeichnungen und sonstigen Dokumenten) gespeichert. Als Datenbanksysteme werden IMS und DB2 auf MVS-Host und Oracle auf UNIX eingesetzt. Alle Systeme sind jeweils nur über ein API zugänglich. Deshalb besteht die Notwendigkeit, nicht nur die Datenbank- bzw. Schema-Integration, sondern in Ergänzung dazu auch die Anwendungssystem- bzw. API-Integration zu unterstützen.

In diesem Aufsatz werden wir eine Architektur zur API-Integration basierend auf Standards entwerfen. Dazu beschreiben wir in Kapitel 2 die notwendigen Grundlagen, nämlich die Schema-Architektur von [SL90], die internationale Norm STEP [ISO94a] und den internationalen Standard CORBA [OMG98a]. Des weiteren erfolgt dort eine Abgrenzung zu vorhandenen Integrationsvorschlägen. Der wesentliche Beitrag ist ein Architekturansatz zur API-Integration, den wir in Kapitel 3 beschreiben. Abschließend fassen wir die Ergebnisse in Kapitel 4 zusammen und geben einen Ausblick auf unser weiteres Vorgehen.

2. Grundlagen

Dieses Kapitel vermittelt einige Grundlagen für das Verständnis unseres Integrationsansatzes. Dabei wird zunächst auf die herkömmliche Architektur der Schemaintegration eingegangen, da die Funktionsintegration darauf basiert. Daraufhin folgt eine kurze Beschreibung der Norm STEP, weil für einen unternehmensübergreifenden Verbund von Systemen eine einheitliche Repräsentation der Daten durch normierte Schemata notwendig ist. Schließlich stellen wir CORBA als einen Mechanismus zur Überwindung der Plattformheterogenität vor.

1.) Die direkt einem Teil zugeordneten Daten, wie z.B. die Identifikation, Größe, Gewicht etc., werden als Stammdaten bezeichnet.

2.1 Die Schema-Architektur nach Sheth/Larson

In [SL90] wird ein allgemein anerkanntes und verwendetes Modell für die Integration von Daten definiert (siehe Abbildung 1). Ausgehend von Komponentenschemata, die auf einem einheitlichen Datenmodell basieren und somit von der Heterogenität der zugrundeliegenden Datenbanken abstrahieren, wird ein föderiertes Schema gebildet. Dies ermöglicht globalen Anwendungen einen transparenten Zugriff auf die integrierten Datenbanken.

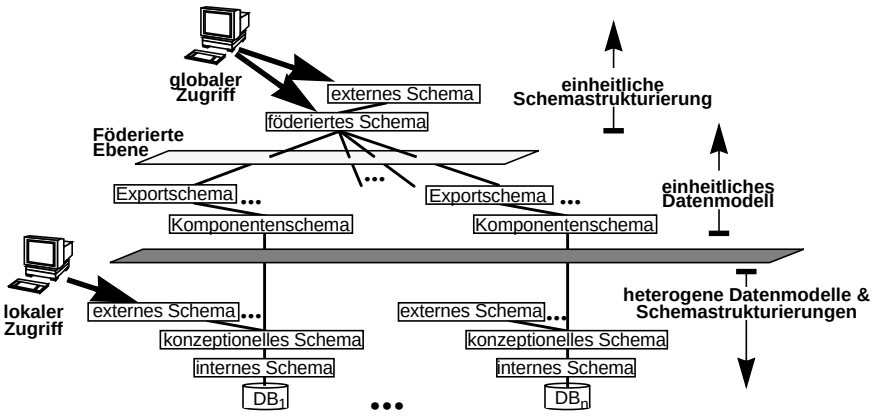

Abbildung 1: Schema-Architektur Föderierter Datenbanksysteme [SL90]

Arbeiten zur Schemaintegration [SL90, BE96, Ki95, HBP94, Sa98, Da96] vernachlässigen jedoch die Integration von Funktionen. Da aber häufig nicht direkt auf die Datenbanken der Komponentensysteme zugegriffen werden kann, ist ein reiner Schemaintegrationsansatz nur selten möglich. Viele Arbeiten verwenden deshalb ein (oftmals proprietäres) objektorientiertes globales Datenmodell mit dessen Hilfe sich auch Funktionen (als Objektmethoden) definieren lassen [GCO90, BNS95, HD92, RS97]. Diese Methoden werden als sog. Wrapper eingesetzt. Das heißt, daß durch sie zur globalen Anwendung hin die lokalen Schnittstellen gekapselt und in der Implementierung der Methoden die vorhandenen Funktionen der Komponentensysteme (wie z.B. die Berechnung einer Stückliste) aufgerufen werden. Bei diesen referenzierten Ansätzen wird keine allgemeine Methodik zur API-Integration, vergleichbar etwa zu [SL90] für die Schemaintegration, vorgegeben. Statt dessen werden in den Implementierungen der globalen Methoden alle Plattformheterogenitäten (Programmiersprachen, Netzwerkprotokolle etc.) proprietär überwunden. Weiterhin werden oftmals keine oder nur wenige Sprachmittel zur Modellierung von Semantik in dem globalen Schema (Vor- und Nachbedingungen, Abhängigkeiten zwischen Objekten, Invarianten etc.) zur Verfügung gestellt. Eine umfangreiche Spezifikationssprache kann jedoch den Integrationsprozeß erleichtern und zum Verständnis der System- und Datenabhängigkeiten beitragen. Dies ist auch Voraussetzung für eine mögliche Automatisierung der Siche-

rung globaler Integritätsbedingungen [NW94]. Im Kontext der DC Systeme müssen die Probleme semantischer Heterogenität (adressiert mit STEP, siehe Kapitel 2.2), und sehr unterschiedlicher Plattformen, Kommunikationsmechanismen und Programmierumgebungen (siehe Kapitel 2.3 über CORBA) gelöst werden.

2.2 Die internationale Norm STEP

Ziel des ISO-Standards STEP (Standard for the Exchange of Product Model Data, [ISO94a]) ist die eindeutige Repräsentation von Produktdaten während des gesamten Produktlebenszyklus zur Unterstützung des Datenaustausches und der Datenintegration. Bisherige Industriestandards (IGES, VDAFS, u.a. [OII98]) beschränken sich hauptsächlich auf den Austausch von Geometriedaten. STEP kann darüberhinaus flexibel auf weitere Anwendungsgebiete erweitert werden. So sind beispielsweise schon heute Stücklisten, Materialeigenschaften, Toleranzen und Arbeitsplanungsinformationen durch den Standard abgedeckt. Mit Hilfe der ebenfalls im Standard beschriebenen Modellierungssprache EXPRESS [ISO94b] werden sog. Anwendungsprotokolle (AP) definiert, welche branchen- bzw. anwendungsspezifische Schemata bereitstellen. Für den Automobilbereich ist dies das AP 214 [ISO97]. Ebenfalls in STEP festgelegt ist das *Standard Data Access Interface* (SDAI) [ISO98b], eine programmiersprachenunabhängige Zugriffsschnittstelle für EXPRESS Daten.

Bei DC spielt STEP (insbesondere das AP 214) bereits eine große Rolle. Die unternehmensübergreifenden, einheitlichen Schemata werden zur Unterstützung des Datenaustausches eingesetzt, so daß sie sich auch als Grundlage zur Schemaintegration anbieten [Sa98]. Für eine Anwendungsintegration ist STEP alleine allerdings nicht ausreichend, da EXPRESS keine Modellierung von Funktionen zuläßt und die technischen Probleme der Plattformheterogenität dort nicht adressiert werden. Mit SDAI steht zwar ein API für EXPRESS-Daten zur Verfügung, diese ist jedoch eine reine Datenzugriffsschnittstelle und somit für eine Funktionsintegration nicht geeignet.

2.3 Der internationale Standard CORBA

Die *Common Object Request Broker Architecture* der Object Management Group (OMG), genannt CORBA [OMG98a], stellt eine Architektur zur Verfügung, die Interoperabilität zwischen verteilten Objekten ermöglichen soll. CORBA-Objekte kommunizieren miteinander über den sog. ORB (Object Request Broker), der im wesentlichen für die Lokation des Serverobjektes und den Aufruf von Methoden unter Gewährleistung von Orts-, Plattform- und Programmiersprachentransparenz zuständig ist. Die Definition von Objektschnittstellen erfolgt mittels der *Interface Definition Language* (IDL). Für die eigentliche Implementierung der Objekte definiert der Standard Abbildungen von IDL auf vorhandene Programmiersprachen (C++, Java, Cobol, u.a.). Zusatzdienste und -schnittstellen, die für die Implementierung verteilter Objekte von Nutzen sein können (wie z.B. Naming, Events und Transactions), werden durch die *CORBA Services* [OMG98b] spezifiziert. Des weiteren gibt es höhere Dienste, die sog. *CORBA Facilities*, die standardisierte Schnittstellen (z.B. für Workflow Systeme) bereitstellen.

Eine solche Schnittstelle für PDM-Systeme ist der *PDM Enabler*. Dessen Verwendung als globale API zur Anwendungsintegration scheitert aber vor allem an der mangelnden Konformität zum AP 214. Des weiteren sind beim PDM Enabler durch die Verwendung des CORBA Relationship Service [OMG98b] Effizienzprobleme - insbesondere im Zusammenhang mit großen Produktstrukturen - zu erwarten. Dies resultiert aus der unverhältnismäßig großen Anzahl von Rollen- und Beziehungsobjekten im Vergleich zu den in Relation gesetzten Objekten.

3. Architektur zur API-Integration

Die STEP- und CORBA-Standards stellen eine notwendige Basistechnologie für die Integration insbesondere von technischen Informationssystemen dar. Wie im vorherigen Kapitel gezeigt wurde, bieten diese Standards jeweils für sich alleine betrachtet keine vollständige Lösung für die API-Integration an. Aus diesem Grund wird in diesem Kapitel nun eine auf STEP- und CORBA-Konzepten aufbauende Integrationsarchitektur vorgestellt.

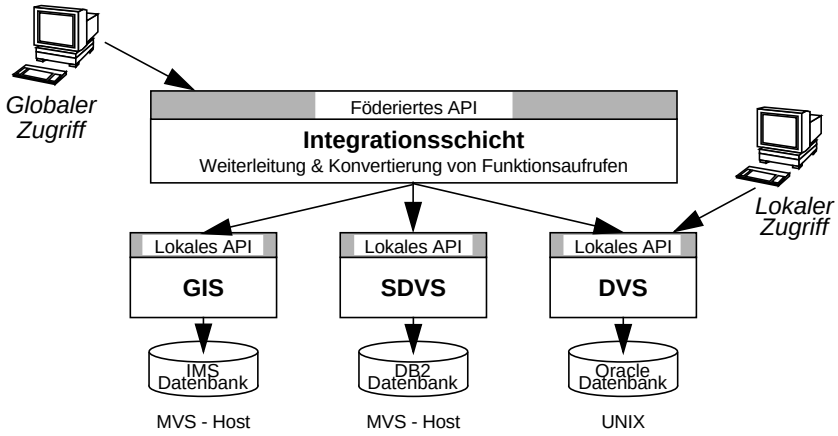

Abbildung 2: Schematische Architektur integrierter Systeme am Beispiel PKW-Entwicklung

3.1 Überblick über die Integrationsarchitektur

Ausgehend von den proprietären Altsystemen bei DC und einem einheitlichen Datenschema (STEP AP 214) wurde eine Architektur und ein Vorgehensmodell für die Integration von Daten und Funktionen entwickelt. Abbildung 2 zeigt die grundlegende Integrationsarchitektur mit einigen typischen Systemen von DC, die bereits in der Einleitung kurz vorgestellt wurden. Den Kern der Integrationsarchitektur bildet das

föderierte API, welches eine vereinheitlichte Daten- und Funktionssicht auf die Komponentensysteme ermöglicht. Dabei wird für eine einheitliche Datensicht die Norm STEP und für die Überwindung von Plattformheterogenität der Standard CORBA eingesetzt. Das föderierte API steht für die Programmierung von globalen Anwendungen (z.B. zur Ansicht von Produktstrukturen/Stücklisten) zur Verfügung. Durch Wiederverwendung der Funktionalität der Komponentensysteme ist der Umfang und Programmieraufwand dieser Anwendungen typischerweise sehr gering ("Thin-Client"). Dadurch ist es insbesondere möglich, relativ einfach Anpassungen der Oberflächen für unterschiedliche Benutzergruppen zu implementieren.

Die durch [SL90] vorgegebene Architektur zur Schemaintegration wird auf den Bereich der API-Integration übertragen, indem wir eine einheitliche Modellierungssprache der Föderation vorschlagen, die nicht mehr auf die Modellierung statischer Aspekte beschränkt ist, sondern auch Methoden und damit ein API beschreiben kann. Analog zu der Ebene der Komponentenschemata aus [SL90] führen wir eine zusätzliche Schicht und damit gleichzeitig eine Abstraktion unterschiedlicher API-Beschreibungssprachen und Programmiersprachen ein. Die korrespondierenden Elemente zu [SL90] werden im folgenden Abschnitt skizziert.

3.2 Modellierung des föderierten APIs

Für die Modellierung des föderierten APIs bietet sich die Verwendung des CORBA-Standards an, da er bereits zur Überwindung der Plattformheterogenität eingesetzt wird. Deshalb favorisierten wir zunächst die sog. *Component Definition Language* (CDL, siehe [OMG98c]). Die CDL ist eine echte Obermenge der IDL und reichert diese um zusätzliche Sprachmittel zur Beschreibung von Semantik an. Beispielsweise ist es mit ihrer Hilfe möglich, Regeln (Invarianten, Zustandsübergänge, usw.), Abhängigkeiten zwischen Objekten sowie Vor- und Nachbedingungen von Objektmethoden zu definieren. Ein wesentlicher Punkt bei der Entwicklung der CDL und der zugehörigen Architektur (BOCA [OMG98c]) war die Entkopplung der Analysephase von der Implementierung. Durch eine mächtige und einfache Spezifikationssprache sollten die wesentlichen Merkmale der Geschäftsobjekte (wie z.B. Teile, Personen, Dokumente, usw.) auf einer hohen Abstraktionsstufe beschrieben werden können. Die eigentliche Implementierung der Objekte erfolgt in IDL (hierzu wurde im OMG-Proposal eine Abbildung von CDL auf IDL definiert), wodurch eine Abstraktion von verschiedenen zugrundeliegenden Programmiersprachen erreicht wird. Dies entspricht der Ebene der Komponentenschemata in [SL90].

Die CDL wurde im Rahmen eines Proposals für "Business Objects" (Geschäftsobjekte) in der OMG eingereicht. Derzeit ist aber zu erwarten, daß die CDL zugunsten einer - noch zu definierenden - textuellen Repräsentation der ebenfalls durch die OMG standardisierten *Unified Modeling Language* (UML, siehe [Oe98]) aufgegeben wird. Dies bietet uns die Möglichkeit, die Konzepte und Techniken von CDL im praktischen Einsatz zu evaluieren und Verbesserungsvorschläge in den Standardisierungsprozeß der textuellen UML einzubringen. Entsprechend erwarten wir, daß der Wegfall von CDL unserer Architektur nicht die Grundlage entzieht, sondern die Möglichkeit bietet, mögliche Schwachstellen von CDL zu eliminieren und ein noch geeigneteres Modell

zu entwickeln. Des weiteren würde durch die Verwendung der UML gleichzeitig die Möglichkeit einer grafischen Modellierung geboten. Der Vorteil wäre ein noch durchgängigerer Entwicklungsprozeß von der Analyse über das Design bis hin zur technischen Umsetzung der Daten- und Funktionsintegration.

3.3 Vorgehensmodell für die Integration

Zur Unterstützung der Integration wurde ein Vorgehensmodell entwickelt (Abbildung 3). Basierend auf dem vorhandenen - in EXPRESS definierten - Ausgangsdatenschema (Schritt 1) wird zunächst (mittels STEP SDAI/IDL-Mapping [ISO98b]) eine Abbildung auf IDL vorgenommen (Schritt 2). Ausgehend von den lokalen APIs

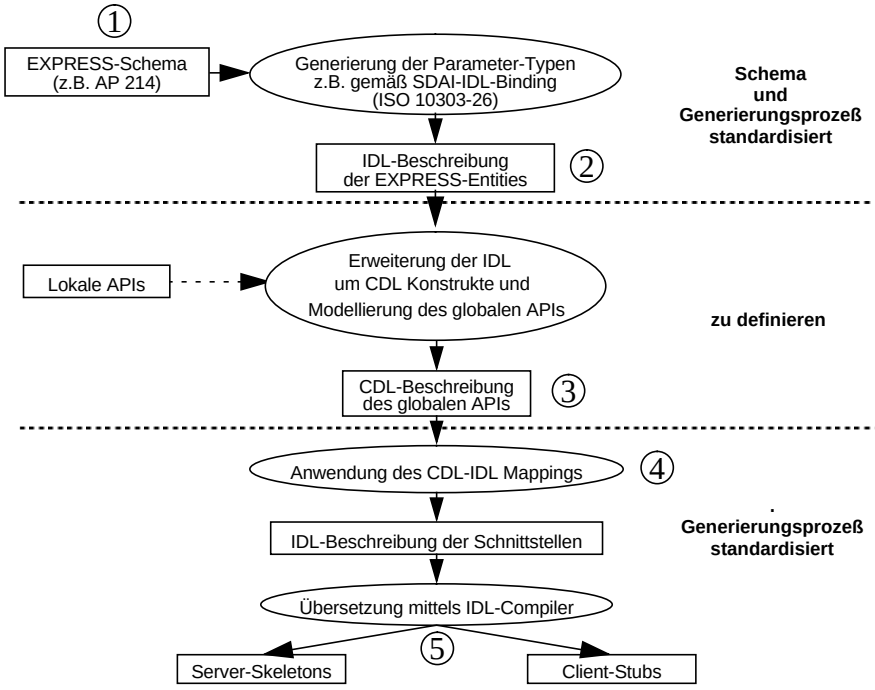

Abbildung 3: Vorgehensmodell zur API-Integration

werden die generierten IDL-Schnittstellen im Anschluß daran um geeignete globale Funktionen erweitert. Weitere Sprachkonstrukte der CDL können zusätzlich zur Beschreibung von Semantik genutzt werden - auf diese Weise erfolgt eine klare CDL-Spezifikation der föderierten Schnittstelle (Schritt 3). Für die eigentliche Implementierung erfolgt wiederum eine (standardisierte) Abbildung von CDL nach IDL (Schritt 4). Die Übersetzung der IDL-Schnittstellen in die entsprechenden Client-Stubs und Server-Skeletons der Zielsprache (C, COBOL, etc.) erfolgt schließlich mittels eines

IDL-Compilers (Schritt 5). Die Implementierungen der Server-Skeletons kapseln die lokalen Funktionen und Datenformate der heterogenen Komponentensysteme. Durch die Verwendung von CORBA als Middleware wird Plattform- und Programmiersprachenunabhängigkeit erreicht. Die Schnittstellen können auf diese Weise unabhängig von der Implementierungsplattform (MVS, UNIX, PC) spezifiziert werden.

Muß beim Aufruf von Methoden des verwendeten föderierten APIs überwiegend eine größere Anzahl von Objekten als Parameter übergeben werden, so ist es sinnvoll vor dem letzten Transformationsschritt (Übersetzung durch den IDL-Compiler) die von dem in EXPRESS definierten Datenmodell (hier AP 214) abhängigen IDL-Datenstrukturen in generische, vom Schema unabhängige IDL-Konstrukte zu konvertieren. Dadurch können die in [SM97] beschriebenen Effizienz-Probleme beim "Data Shipping" in CORBA-Umgebungen vermieden werden. Eine Abbildung von EXPRESS auf generische IDL-Strukturen wird z.B. in [SM98] beschrieben.

3.4 Beispiel anhand von STEP AP 214

Zur näheren Erläuterung des Vorgehensmodells folgt ein kurzes Beispiel. Abbildung 4 zeigt einen Auszug der EXPRESS-Spezifikation des STEP AP 214, der die Grundlage für die integrierte Sicht darstellt. Dies entspricht Schritt 1 in Abbildung 3. Der Einfach-

```
ENTITY item;                                          ENTITY item_version;
   id                  : STRING;                          ...
   name                : STRING;                      END_ENTITY;
   description         : STRING;
   associated_versions : SET [1:?] OF item_version;
END_ENTITY;
```

Abbildung 4: Auszug aus STEP AP 214

heit halber beschränken wir uns hier auf die beiden Entities `item` und `item_version`, die ein Teil bzw. dessen Version repräsentieren können. Jedes Teil besteht aus einer eindeutigen Id, einem Namen sowie einer Beschreibung. Des weiteren existiert das Attribut `associated_versions`, welches die 1:n-Beziehung zu den zugehörigen Versionen modelliert (aus Platzgründen wird `item_version` in den folgenden Schritten nicht näher beschrieben). Nach Anwendung des in STEP spezifizierten EXPRESS/IDL-Mappings (Schritt 2 im Vorgehensmodell) erhält man die in Abbildung 5 gezeigte

```
INTERFACE item {
   ATTRIBUTE STRING id;                  // item id
   ATTRIBUTE STRING name;                // item name
   ATTRIBUTE STRING description;         // item description
                                         // associated item versions
   ATTRIBUTE SEQUENCE<item_version> associated_versions;
}
```

Abbildung 5: Vorläufige IDL-Beschreibung von `item`

(vorläufige) IDL-Beschreibung des Entities `item`. Hierbei fällt insbesondere auf, daß die in [ISO98b] vorgegebene Verwendung von IDL-Sequenzen zur Modellierung von Beziehungen nur bedingt geeignet ist. So ist es auf diesem Wege generell nicht möglich Kardinalitätsrestriktionen, Integritätsbedingungen oder inverse Attribute zu spezifizieren. In Schritt 3 erfolgt schließlich die Transformation der IDL-Schnittstellen nach CDL (Abbildung 6) mit entsprechender Spezifikation der globalen Funktionen (in diesem Fall die Funktion `get_latest_version`, welche in gängigen PDM-Systemen vorhanden ist und hier global zur Verfügung gestellt werden soll). Durch Anreicherung

```
[KEYS={id}] ENTITY item {
   [REQUIRED] ATTRIBUTE STRING id;          // item id
   ATTRIBUTE STRING name;                   // item name
   ATTRIBUTE STRING description;            // item description
   // associated item versions
   RELATIONSHIP associated_versions MANY item_version;
   // invariant
   APPLY INVARIANT part_requires_version {
      guard = associated_versions.count > 0;
   }
   // global operations
   item_version get_latest_version();
   ...
}
```

Abbildung 6: CDL-Beschreibung von `item`

des Entities um weitere CDL-Sprachkonstrukte kann eine genauere Beschreibung der föderierten Schnittstelle erreicht werden. Im diesem Fall kann beispielsweise ein Schlüssel (Id) für `item` angegeben und die Beziehung zu `item_version` direkt mittels des CDL-Relationship-Konstruktes beschrieben werden. Da das Schlüsselwort `MANY` für 0:n-Beziehungen steht, wird die ursprünglich im EXPRESS-Modell definierte Kardinalität durch eine zusätzliche Invariante (`part_requires_version`) zugesichert[2]. Das für Schritt 4 benötigte CDL/IDL-Mapping für die letztendliche Implementierung der föderierten API kann in [OMG98d] nachgelesen werden. Im Prinzip werden dabei hauptsächlich CDL-Entities auf IDL-Interfaces, sowie CDL-Relationships auf einen Satz von IDL-Methoden (`add`, `remove`, `list`, u.a.) abgebildet. Andere Sprachkonstrukte der CDL, wie beispielsweise Invarianten und Conditions, werden nicht auf IDL abgebildet, weswegen die CDL-Spezifikation immer zur eigentlichen Implementierung der Schnittstellen vorliegen sollte. Schritt 5 des Vorgehensmodells (Abbildung der IDL-Konstrukte auf Elemente der jeweiligen Programmiersprache) entspricht dem Aufruf des zum CORBA-System gehörenden IDL-Compilers [OMG98a].

2.) Es können auch explizit Kardinalitäten beim Relationship-Konstrukt angegeben werden. Darauf wurde an dieser Stelle jedoch bewußt verzichtet um das Konzept der Invarianten vorzustellen.

4. Zusammenfassung, Bewertung und Ausblick

In dieser Arbeit haben wir eine Architektur sowie ein Vorgehensmodell vorgestellt, um die Integration von Daten und Funktionen zu ermöglichen. Die Konzepte wurden primär anhand der Anforderungen innerhalb von DC entwickelt, sind aufgrund ihrer allgemein gehaltenen und auf Standards basierenden Struktur jedoch leicht auf andere Bereiche zu übertragen. Mit Ausnahme der Modellierung des globalen (und derzeit proprietären), föderierten APIs stützen sich alle Eingaben und Transformationsschritte auf internationale Standards (siehe Abbildung 3): STEP definiert das globale Datenmodell (AP 214) sowie die Abbildung auf IDL- bzw. CDL-Schnittstellen, CORBA enthält die Sprachen IDL und CDL (bzw. deren Nachfolger, siehe Kapitel 3) sowie die nötigen Abbildungen auf konkrete Programmiersprachen. Dadurch wird es ermöglicht, Anwendungssysteme zu integrieren, die auf beliebigen Rechnern und Betriebssystemen mit Hilfe einer beliebigen Programmiersprache erstellt wurden.

Die zu erwartende Ersetzung der Sprache CDL durch eine (noch zu entwickelnde) textuelle Version der UML bietet uns die Möglichkeit Schwachstellen der CDL aufzuzeigen. Diese sollen dann in den weiteren Standardisierungsprozeß der UML eingebracht werden, um letztendlich ein noch mächtigeres Konzept zu erhalten. Dafür soll in einer ersten Stufe ein Prototyp entwickelt werden, der zunächst die Funktionalität der bestehenden Anwendungssysteme global verfügbar macht. In einem zweiten Schritt soll dieser dann die Möglichkeit zur Definition erweiterter Funktionen bieten, die nicht direkt auf bestende APIs, sondern nur mittels eines “Mappings” abgebildet werden können.

Von der hier beschriebenen Integrationsarchitektur erwarten wir eine enorme Produktivitäts- und Qualitätssteigerung. Die API-Integration löst ein durch den Benutzer gesteuertes Interagieren separater Anwendungssysteme ab. Dadurch wird die Fehleranfälligkeit hinsichtlich Funktionsaufrufen und Datenübernahmen deutlich reduziert und durch all diese Erleichterungen insgesamt die Entwicklungszeit neuer Produkte verkürzt.

Literatur

BE96 O.A. Bukhres, A.K. Elmagarmid (Hrsg.): *Object-Oriented Multidatabase Systems - A Solution for Advanced Applications*, Prentice Hall, 1996.

BNS95 E.Bertino, M.Negri, L.Sbattella: *An Overview of the Comandos Integration System*, Object-Oriented Multidatabase Systems (O.Bukhres and A.Elmagarmid, eds.), Prentice-Hall, 1995.

Da96 P. Dadam: *Verteilte Datenbanken und Client/Server-Systeme*, Springer Verlag, 1996.

ENV98 Enovia Corp.: *ENOVIAVPM PDM II Solutions*, 1998; zu beziehen über http://www.enovia.com/products/html/enoviavpm.html.

GCO90 R. Gagliardi, M. Caneve, G. Oldano: *An Operational Approach to the Integration of Distributed Heterogeneous Environments*, Databases: Theory, Design, and Applications, postconference publication of PARBASE-90, First International Conference on Databases, Parallel Architectures and their Applications, 1991, ISBN 0-8186-9165-4, S. 110-124

HBP94 A.R. Hurson, M.W. Bright, S.H. Pakzad (Hrsg.): *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, IEEE Comp. Society Press, 1994.

HD92 M. Härtig, K. R. Dittrich: *An object-oriented integration framework for building heterogeneous database systems*, Proceedings of the IFIP DS-5 Conference on Semantics in Interoperable Database Systems, Lorne, Australia, S. 33-53, 1992

ISO94a ISO 10303-1 TC184/SC4: *Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles*, International Standard, 1994.

ISO94b ISO 10303-11 TC184/SC4: *Product Data Representation and Exchange - Part 11: EXPRESS language reference manual*, International Standard, 1994.

ISO98a ISO 10303-22 TC184/SC4: *Product Data Representation and Exchange - Part 22: Standard Data Access Interface*, Draft International Standard, 1998.

ISO98b ISO 10303 TC184/SC4/WG11/N045: *Product Data Representation and Exchange - Part 26: Interface Definition Language Binding to the Standard Data Access Interface*, Draft International Standard, Juni 1998.

ISO97 ISO 10303 TC184/SC4/WG3/N578 : *Product Data Representation and Exchange - Part 214: Core Data for Automotive Mechanical Design Processes*, Working Draft, März 1997.

Ki95 W. Kim (Hrsg.): *Modern Database Systems - The Object Model, Interoperability, and Beyond*, Addison-Wesley, 1995.

NW94 M. Norrie, M. Wunderli et al.: *Coordination Approaches for CIM*, Proceedings European Workshop on Integrated Manufacturing Systems Engineering (IMSE), 1994, S. 223-232.

Oe98 B. Oestereich: *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*, R. Oldenbourg Verlag, 1998.

OII98 Homepage des European Commission's Open Information Interchange (OII) service, http://www2.echo.lu/oii/en/oii-home.html

OMG98a Object Management Group*: The Common Object Request Broker Architecture: Architecture and Specification*, Revision 2.2, http://www.omg.org/corba/corbiiop.htm, February 1998.

OMG98b Object Management Group: *The Common Object Request Broker Architecture: Common Object Services Specification*, http://www.omg.org/corba/sectrans.htm, Upd. July 1998.

OMG98c Object Management Group*: Business Object Component Architecture (BOCA) Proposal*, Revision 1.1, OMG, 1998; zu beziehen über ftp://ftp.omg.org/pub/docs/bom/98-01-07.pdf

OMG98d Object Management Group*: Interoperability Specification Proposal*, OMG, 1998; zu beziehen über ftp://ftp.omg.org/pub/docs/bom/98-01-10.pdf

RH98 F.d.F. Rezende, K. Hergula: *The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways*, VLDB 1998, S. 146-157.

RS97 Mary Tork Roth, Peter M. Schwarz: *Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources*, VLDB 1997; S. 266-275.

Sa98 G. Sauter: *Interoperabilität von Datenbanksystemen bei struktureller Heterogenität*, Dissertation, infix Verlag, 1998.

SAP98 SAP AG: *Das R/3 System*, 1998; zu beziehen über http://www.sap-ag.de/products/r3/.

SDRC98 SDRC Corp.: *Metaphase*, 1998; zu beziehen über http://www.metaphasetech.com/.

Si96 J. Siegel: *CORBA: Fundamentals and Programming*, Jon Wiley & Sons, 1996.

SL90 A.P. Sheth, J.A. Larson: *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, ACM Computing Surveys 22 (3), 1990, Seite 183-236.

SM97 J. Sellentin, B. Mitschang: *Möglichkeiten und Grenzen des Einsatzes von CORBA in DB-basierten Client/Server-Anwendungssystemen*, in: K. R. Dittrich, A. Geppert: Datenbanksysteme in Büro, Technik und Wissenschaft, GI-Fachtagung BTW 97, Ulm, März 1997, Seite 312-321.

SM98 J. Sellentin, B. Mitschang: *Design and Implementation of a CORBA Query-Service Accessing EXPRESS-based Data*, Internal Report, Subject to Further Publication, 1998.