Diplomarbeit

Erweiterte Konzepte zur Funktionsintegration

KLAUS WISSMANN

2000

1. Gutachter: PROF. DR. MICHAEL WEBER
2. Gutachter: DR. GÜNTER SAUTER
Betreuerin: DIPL.-INF. KLAUDIA HERGULA

Inhaltsverzeichnis

Abbildungsverzeichnis

Kurzfassung: Anwendungssysteme stellen im Gegensatz zu DBMSen nur ein fest vorgegebenes API zum Zugriff auf die durch sie verwalteten Daten zur Verfügung. In dieser Arbeit soll zunächst ein Ansatz zur Integration von Anwendungssystemen durch die Beschreibung von Parameterabhängigkeiten vorgestellt werden. Darauf aufbauend wird die Anwendung dieses Ansatzes auf objektorientierte Systeme erläutert. Weiterhin werden Konstrukte zur Kontrollflußsteuerung und zur Fehlerbehandlung vorgestellt sowie der schreibende Fall behandelt. Anschließend wird die auf XML basierende Sprache FIX zur Beschreibung von Funktionsabbildungen vorgestellt. Zuletzt werden die Anforderungen an ein Ausführungsmodell beschrieben. Anhand von MQ Workflow wird die Eignung eines Workflow Management Systems als Ausführungsumgebung evaluiert.

Kapitel 1

Einleitung

Der Integration heterogener IT-Strukturen innerhalb von Unternehmen kommt heutzutage eine hohe Bedeutung zu. Einerseits muß der Zugriff auf "historisch gewachsene Altlasten" gewährleistet und deren Interaktion mit modernen Systemen ermöglicht werden, andererseits erfordert die zunehmende Verringerung der Fertigungstiefe, und die damit einhergehende Spezialisierung der Unternehmen, eine immer stärkere Verknüpfung der EDV-Strukturen kooperierender Unternehmen.

Integration bedeutet dabei neben der Überwindung der systembedingten Heterogenitäten wie den unterschiedlichen Hardware- bzw. Betriebssystemplattformen oder der Schaffung standardisierter Dokumentenformate zum Datenaustausch vor allem die Integration von Datenbanken und anderen Serversystemen. Während die Integration von relationalen bzw. objektorientierten Datenbankmanagementsystemen (DBMS) bereits Gegenstand intensiver Forschung ist (beispielsweise in [Sauter, 1998] und [Dadam, 1996]), wurde die Integration sogenannter Anwendungssysteme eher selten betrachtet. Als Anwendungssysteme werden dabei Systeme bezeichnet, die das Datenbanksystem sowie die zugehörige Anwendung kapseln, und auf die ein Zugriff nur über ein vorgegebenes API (Application Programming Interface) möglich ist.

Diese Kapselung erfolgt beispielsweise, falls das Anwendungssystem Teile der Aufgaben des ihm zugrundeliegenden DBMS (z.B. die Überprüfung von Integritätsbedingungen) übernimmt. Dies kann aus Performancegründen oder zur Wahrung der Austauschbarkeit des benutzten DBMS erfolgen. Prominentes Beispiel dafür, daß Anwendungssysteme nicht nur bei Legacy-Systemen auftreten, ist das SAP/R3 System [SAP], welches seine Funktionalität nur über sogenannte Business APIs (BAPIs) zur Verfügung stellt.

Im Gegensatz zu DBMSen, welche fest vorgegebene Mechanismen zum Zugriff auf die darin enthaltenen Daten zur Verfügung stellen, kann der Aufbau der von Anwendungssystemen zur Verfügung gestellten Schnittstellen sehr unterschiedlich sein. Es muß aus diesem Grund ein System entwickelt werden, welches flexibel genug ist, alle denkbaren Schnittstellen von Anwendungssystemen integrieren zu können. Die Abbildung der zu integrierenden Systeme auf ein globales System soll dabei durch eine Integrationsschicht erfolgen, welche ihrerseits das globale API zur Verfügung stellt. Um eine langsame Migration zu ermöglichen muß es aber dennoch weiterhin möglich sein, daß bereits existierende Clients direkt auf die bisherigen Systeme zugreifen können. Abbildung 1.1 illustriert den Aufbau der zur Integration notwendigen Architektur.

Kapitel 2 beginnt mit einigen grundlegenden Betrachtungen zu Anwendungssystemen, den von ihnen exportierten Funktionen und den möglichen Heterogenitäten. Außerdem erfolgt die Einführung der

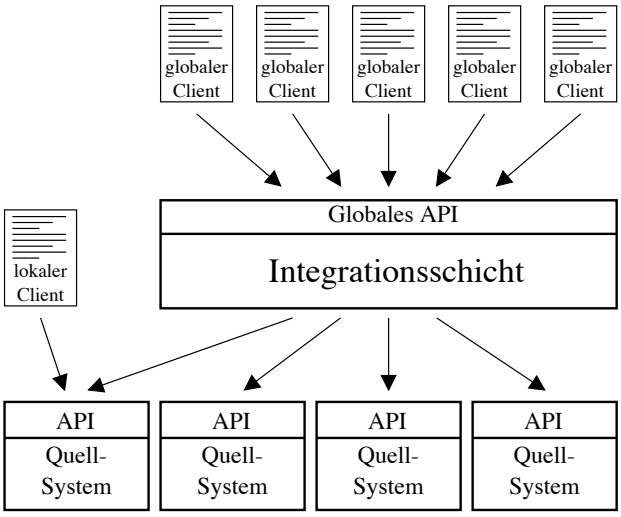


Abbildung 1.1: Integrationsarchitektur.

im Verlauf dieser Arbeit benutzten Beispielsysteme.

In Kapitel 3 erfolgt zunächst eine kurze Einführung in die in [Vogt, 1999] erarbeiteten Grundlagen zur Funktionsintegration durch Parameterabhängigkeiten. Danach werden die notwendigen Erweiterungen im Zusammenhang mit objektorientierten Systemen betrachtet. Anschließend wird die Realisierung von Kontrollstrukturen wie bedingten Anweisungen oder Schleifen beschrieben. Weiterhin erfolgt eine Betrachtung der Fehlerbehandlung und des schreibenden Zugriff. Abschließend wird kurz auf die Verwendung einer Standardbibliothek und auf semantische Attribute eingegangen.

Thema des 4. Kapitels ist die Sprache FIX. Neben einer Erläuterung aller verwendeten Elemente und deren Verwendung anhand von Beispielen werden die in FIX gewählten Lösungsmöglichkeiten der in Kapitel 3 vorgestellten Probleme beschrieben. Weiterhin erfolgt eine Darstellung der Konvertierungsmöglichkeiten von FIX Dokumenten durch XSL Stylesheets.

Kapitel 5 beschäftigt sich mit den Anforderungen an das Ausführungsmodell für FIX Dokumente. Anhand des Workflow Management Systems MQ Workflow von IBM wird die Eignung eines Workflow Management Systems als Ausführungsumgebung untersucht.

Abschließend erfolgt in Kapitel 6 eine Zusammenfassung dieser Arbeit. Zusätzlich sollen mögliche weitere Entwicklungen aufgezeigt werden.

Kapitel 2

Grundlagen

In diesem Abschnitt sollen zunächst die grundlegenden Konzepte von Anwendungssystemen und Funktionen erläutert werden. Anschließend werden die möglichen Heterogenitäten besprochen und das in den folgenden Kapiteln benutzte Beispiel eingeführt.

2.1 Anwendungssysteme

In dieser Arbeit liegt das Hauptaugenmerk auf der Integration sogenannter Anwendungssysteme. Diese genügen dabei folgender Definition.

Definition: *Anwendungssysteme* sind Systeme, die Ihre Funktionalität ausschließlich über ein API (Application Programming Interface) zur Verfügung stellen. Insbesondere wird von Anwendungssystemen keine Datenbankschnittstelle zur Verfügung gestellt.

Anwendungssysteme können ihre Schnittstellen sowohl für lokale als auch entfernte Clients zur Verfügung stellen. Der Zugriff auf entfernte Schnittstellen durch die Clients kann dabei beispielsweise über RPC- (Remote Procedure Call) oder Objektdienste erfolgen.

Das durch die Integration entstehende System besitzt dabei die Charakteristika eines Anwendungssystems. Es wird deshalb zusätzlich zwischen lokalen und globalen Systemen unterschieden.

Definition: Als *lokales System* bzw. *Quellsystem* werden bereits existierende Anwendungssysteme bezeichnet, die im Rahmen der Integration nicht oder nur in sehr geringem Umfang verändert werden können. Auf die von diesem System exportierten Schnittstellen wird sowohl von den existierenden lokalen Clients als auch von dem globalen System (welches sich dabei wie ein lokaler Client verhält) zugegriffen.

Als *globales System* bzw. *Zielsystem* wird das durch die Integration entstehende Anwendungssystem bezeichnet. Der Zugriff auf dieses System erfolgt von globalen Clients. Eine globale Schnittstelle kann (im Rahmen einer weiteren “Integrationsrunde”) selbst wieder als lokales System agieren.

2.2 Funktionen und Attribute

Die von Anwendungssystemen exportierten APIs basieren meist auf einer Menge von aufrufbaren Funktionen. Objektorientierte Anwendungssysteme stellen ihre Funktionalität über Methoden und Attribute serverseitig instantiierbarer Objekte zur Verfügung.

Funktionen in der Mathematik

Ein von einem Anwendungssystem zur Verfügung gestelltes API besteht aus einer Menge von Funktionen. Der Begriff der Funktion kommt ursprünglich aus dem Bereich der Mathematik. Dort ist eine Funktion folgendermaßen definiert [Meyberg, 1993]:

Definition: Seien A und B zwei Mengen. Eine Funktion $f : A^n \supseteq D \mapsto B^m$ bzw. $x \mapsto f(x)$, ist eine Vorschrift, die jedem Vektor $\mathbf{x} \in D$ einen Vektor $\mathbf{f}(\mathbf{x}) \in B^m$ zuordnet. Man bezeichnet dabei D als Definitionsbereich und B^m als Bildbereich.

Funktionen in der Informatik

Innerhalb von Programmiersprachen werden Funktionen eindeutig über ihre Signatur identifiziert. Die Signatur umfaßt dabei den Namen der Funktion und die Datentypen sämtlicher Parameter sowie des optionalen Rückgabewerts. Die Parameter können dabei sowohl Eingabe- oder Ausgabeparameter als auch beides sein. Der Rückgabewert ist ein Spezialfall eines Ausgabeparameters. Die im folgenden Beispiel deklarierte Funktion trägt den Namen `get_color` und erhält den Eingabeparameter `no` vom Typ `long`. Die Funktion liefert einen Rückgabewert vom Typ `string`.

```
string get_color(in long no);
```

Neben den offensichtlichen Gemeinsamkeiten zu dem Funktionsbegriff der Mathematik ergeben sich einige Unterschiede. So ist es möglich, daß Funktionen in Programmiersprachen keine Eingabe- oder Rückgabeparameter besitzen. Des weiteren können die Parameter einer Funktion verschiedene Datentypen besitzen.

Funktionen in Programmiersprachen können sogenannte Seiteneffekte verursachen, d.h. sie haben die Möglichkeit neben der Rückgabe von Werten auch den Zustand des Systems zu verändern. Beispiele hierfür sind der schreibende Zugriff auf eine Datenbank oder die Veränderung einer globalen Variable eines Systems (z.B. auch durch die Initialisierung oder die Anmeldung an einem System). Besitzt eine Funktion keine Rückgabeparameter, kann offensichtlich davon ausgegangen werden, daß sie Seiteneffekte verursachen wird.

Im Gegensatz zu Funktionen im mathematischen Sinn können Funktionen in der Informatik bei mehrfachem Aufruf mit denselben Parametern unterschiedliche Werte zurückgeben. Dies kann entweder durch Seiteneffekte innerhalb dieser oder anderer, zwischenzeitlich ausgeführter Funktionen oder durch Änderungen "von außen"[1] an dem System verursacht worden sein.

[1] Dabei stellt die fortgeschrittene Uhrzeit zwischen zwei Funktionsaufrufen ebenfalls eine Änderung "von außen" dar.

2.3 Heterogenitäten

Die bei der Integration von Funktionen zu überwindenden Heterogenitäten lassen sich in semantische und strukturelle Heterogenitäten unterteilen. Semantische Heterogenität bedeutet, daß ein differierendes Verständnis bzgl. der von verschiedenen Anwendungssystemen angebotenen Funktionalität bzw. der benutzten Funktionen und Parameter besteht. Häufigste Ursache im Bereich der Funktionsintegration ist hierfür eine unzureichende Dokumentation der Anwendungssysteme. Steht keine ausreichende Dokumentation zur Verfügung, sind die Funktionssignaturen der einzige Anhaltspunkt zur Bestimmung der angebotenen Funktionalität. Für eine vollständige Diskussion der semantischen Heterogenität sei auf [Härder, 1999] bzw. [Sauter, 1998] verwiesen.

Von struktureller Heterogenität spricht man bei Unterschieden in der Repräsentation der angebotenen Funktionalität. Die gängigsten Heterogenitäten im Bereich der Funktionsintegration sind dabei:

1. Funktionen der Quell- und Zielsysteme sind in verschiedenen Programmiersprachen implementiert bzw. werden auf getrennten Systemen ausgeführt;
2. fehlende Übereinstimmung der Namen von Funktionen bzw. Parametern;
3. Konflikte zwischen Funktions- oder Parameternamen;
4. keine Übereinstimmung der Datentypen von Parametern bzw. Rückgabewerten;
5. (1:n) bzw. (n:1) Abbildungen von Parametern.

Während die im ersten Punkt beschriebenen Heterogenitäten durch ein Middleware-System wie z.B. CORBA [CORBA] überwunden werden können, müssen die restlichen Heterogenitäten von der zu entwickelnden Integrationsarchitektur behandelt werden. Eine ausführliche Diskussion der im Rahmen der Funktionsintegration auftretenden Heterogenitäten erfolgt in [Vogt, 1999] und [Hergula, 1999].

2.4 Beispiel

Zur Visualisierung der Problematik und zur Erklärung der erarbeiteten Lösungen wird im folgenden immer auf dasselbe Beispiel zurückgegriffen. Das Beispiel umfaßt zwei Werke der imaginären Firma example.com und deren Lieferanten. Da die EDV-Strukturen bei dieser Firma historisch gewachsen sind, werden die in beiden Werken verwalteten Bauteile auf sehr unterschiedliche Weise repräsentiert. In der Datenbank des Lieferanten, die bei der Integration mit einbezogen werden soll, werden die lieferbaren Bauteile wiederum auf eine andere Art und Weise gespeichert. Jede der drei Institutionen betreibt ein Anwendungssystem, das eine Schnittstelle zum Zugriff auf die Teiledaten zur Verfügung stellt. Zu den zu einem bestimmten Bauteil verwalteten Daten gehören

1. eine eindeutige Identifikationsnummer,
2. die Bezeichnung des Bauteils,
3. die Farbe,
4. die Maße des Bauteils und

5. der Beschaffungspreis beim Lieferanten.

Während die Identifikationsnummer und die Bezeichnung in beiden Werken verwaltet werden, ist die Farbe des Bauteils nur im System von Werk 1 gespeichert, die Maße wiederum nur in dem von Werk 2. Die aktuellen Preise der Bauteile liegen nur in der Datenbank des Lieferanten vor. Für Bauteile, die dieser Lieferant nicht im Programm hat, liegen in seinem System auch keine Daten vor.

Jedes der drei Systeme stellt eine ähnliche Schnittstelle zum Zugriff auf die gespeicherten Daten zur Verfügung. Die Funktion `all_parts()` gibt eine Liste sämtlicher Bauteile zurück. Mit den Funktionen `get_part()`, `upd_part()`, `add_part()` und `del_part()` können Einträge ausgelesen, verändert, hinzugefügt oder gelöscht werden.

Das Zielsystem soll über eine ähnliche Schnittstelle den Zugriff auf den Namen, die Farbe und den Preis der Bauteile ermöglichen. Außerdem sollen einzelne zusätzliche Funktionen (wie z.B. die Ausgabe des Preises in verschiedenen Währungen) zur Verfügung gestellt werden. Die kompletten Definitionen der Schnittstellen befinden sich in Anhang E.

Kapitel 3

Integration durch Parameterabhängigkeiten

In diesem Kapitel soll zunächst eine Einführung in die Funktionsintegration durch Parameterabhängigkeiten gegeben werden. Neben den Grundlagen wird die Benutzung von Konstanten, die Referenzierung von Teilen von Parametern, die mehrfache Instantiierung von Funktionen sowie textuelle Beschreibungen innerhalb eines Abhängigkeitsgraphs behandelt.

Danach erfolgt die Beschreibung der notwendigen Erweiterungen im Zusammenhang mit objektorientierten Systemen. Dazu gehört neben der Unterstützung von Attributen vor allem die Verwendung von Objekten als Parameter.

Die anschließende Betrachtung der Kontrollflußsteuerung umfaßt die Behandlung von bedingten Anweisungen und Schleifen. Zusätzlich werden Mechanismen zur redundanten Ausführung vorgestellt.

Das Thema Fehlerbehandlung betrachtet sowohl die Benutzung von Exceptions innerhalb von Abhängigkeitsgraphen als auch die Nachbildung von Exceptions für lokale Systeme, die keine Exceptions unterstützen. Zusätzlich wird auf die Benutzung von Kompensationsaktionen im Fehlerfall eingegangen.

Im Rahmen des schreibenden Zugriffs erfolgt die Betrachtung der Möglichkeiten von Transaktionen in Zielsystemen. Zusätzlich werden auftretende Aktualisierungsprobleme untersucht.

Abschließend wird kurz auf die Verwendung einer Standardbibliothek und auf semantische Attribute eingegangen.

3.1 Grundlagen

3.1.1 Motivation

Eine naheliegende Lösung zur Integration von Anwendungssystemen wäre die Definition einer globalen Schnittstelle und die Ausprogrammierung der darin angebotenen Funktionen unter Benutzung der lokalen Systeme. So könnte beispielsweise die Realisierung der globalen Funktion `get_name()` unter Ausnutzung der lokalen Funktionen `get_part()` und `inttostr()` folgendermaßen aussehen:

```
string get_name(in long no) {
  string partno, result;
  dimension_t dummy;

  partno = inttostr(no);
  werk2.get_part(partno, result, dummy);

  return result;
}
```

Anhand dieses Beispiels werden einige grundlegende Schwächen dieses Ansatzes deutlich:

Komplexität: Durch die Implementierung der globalen Funktion in einer Programmiersprache werden die zur Überwindung der vorhandenen Heterogenitäten notwendigen Mechanismen im Code der Programmiersprache "versteckt". Es ist dadurch (vor allem Dritten) schwer möglich, sich einen Überblick über das System zu verschaffen, ohne zuvor den Code des globalen Systems analysiert zu haben.

Wartung: Ergeben sich Änderungen an den lokalen Systemen (etwa durch den Austausch oder das Hinzufügen eines lokalen Systems), werden dadurch u.U. weitreichende Änderungen an der Implementierung der globalen Funktionen notwendig. Auch eine Übertragung auf andere Programmiersprachen oder Plattformen wird durch die Festlegung auf eine Programmiersprache erschwert. Weitere Schwierigkeiten bzgl. der Wartbarkeit folgen direkt aus der hohen Komplexität.

Dokumentation: Das Problem der mangelnden Dokumentation der lokalen Systeme wird durch diesen Ansatz nicht gelöst. Ebenso kann die Implementierung des globalen Systems wegen Ihrer Komplexität nicht zur Dokumentation der Heterogenitäten genutzt werden. Es wird ein separates System zur Dokumentation benötigt.

Um diese Probleme zu umgehen wurde in [Vogt, 1999] ein anderer Lösungsansatz erarbeitet. Bei der Implementierung in einer Programmiersprache gilt das Hauptaugenmerk den benutzten Funktionen. Legt man den Fokus aber auf die verschiedenen Parameter bzw. deren Werte, so erkennt man, daß sich die Funktionsintegration durch die Abbildung von Parametern darstellen läßt. In folgendem Beispiel soll die globale Funktion `get_name()` unter Verwendung der lokalen Funktion `get_part()` des Anwendungssystems von Werk 1 realisiert werden[1]. Dazu wird der Eingabeparameter `no` der globalen Funktion auf den Eingabeparameter `partno` der Funktion `get_part()` abgebildet. Danach erfolgt die Abbildung des Ausgabeparameters `name` auf den Rückgabewert von `get_name()`. Abbildung 3.1 zeigt die für das Beispiel notwendigen Abbildungen.

Zusätzlich zu den beschriebenen Abbildungen erfolgt beim Aufruf einer lokalen Funktion eine implizite Abbildung der Eingabeparameter auf deren Ausgabeparameter bzw. Rückgabewerte (hier durch einen gepunkteten Pfeil dargestellt). Es zeigt sich, daß die explizite Modellierung einer Ausführungsreihenfolge nicht notwendig ist, da sich diese durch die Parameterabhängigkeiten implizit ergibt.

[1] Abbildung 3.23 auf Seite 38 illustrierte eine mögliche Implementierung von `get_name()` unter Ausnutzung der `get_part()` Funktionen beider Werke.

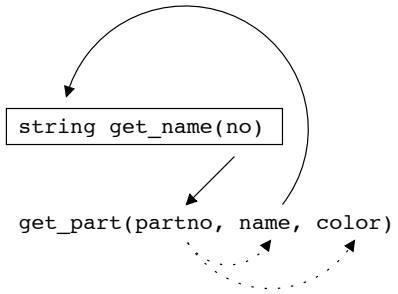


Abbildung 3.1: Abbildung von Parametern.

3.1.2 Hilfsfunktionen

Im vorangegangen Beispiel besaßen die aufeinander abgebildeten Parameter denselben Datentyp. Ist dies nicht der Fall, muß eine Konvertierung der Parameter erfolgen. Operationen wie die Konvertierung zwischen Datentypen oder Stringoperationen, wie z.B. die Konkatenation von Strings, werden durch sogenannte Hilfsfunktionen realisiert. Hilfsfunktionen werden ebenso wie Funktionen der Quellsysteme behandelt und können so in die Abbildungsbeschreibung mit eingebunden werden. Abbildung 3.2 zeigt dies anhand der Modellierung der Funktion `get_color()`.

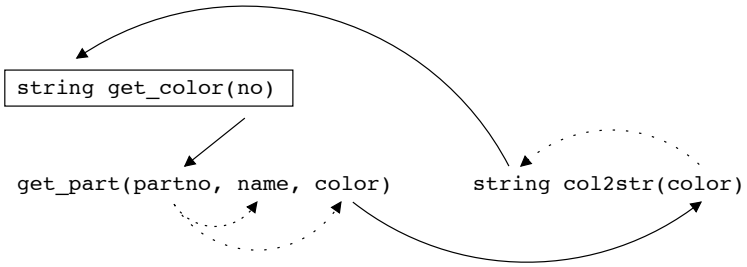


Abbildung 3.2: Benutzung von Hilfsfunktionen.

Hilfsfunktionen müssen dabei in separaten, sogenannten Hilfssystemen, implementiert werden. Hilfssysteme unterscheiden sich, außer dem Namen nach, in ihren Eigenschaften nicht von lokalen Systemen. Innerhalb der Implementierung einer Hilfsfunktion sollten keine Funktionen anderer, im Rahmen der Integration benutzter Quellsysteme aufgerufen werden. Dadurch wird vermieden, daß die zur Integration notwendigen Maßnahmen in der Implementierung einer Quellfunktion “versteckt” werden.

Hilfssysteme können einerseits bereits vor Beginn der Integration vorhanden sein — entweder aus früheren Integrationsphasen vorhandene bzw. im Rahmen einer Standardisierung definierte Standardsysteme (siehe auch Abschnitt 3.7.1) — oder bei der Integration “on the fly” erstellt werden. Es ist weiterhin denkbar, daß Hilfssysteme nicht durch eine Programmiersprache implementiert, sondern ebenfalls durch eine Integration lokaler Systeme erzeugt wurden. Hilfssysteme können also gleichzeitig sowohl Quell- als auch Zielsysteme sein.

3.1.3 Graphentheorie

Betrachtet man beispielsweise Abbildung 3.2, so werden Gemeinsamkeiten zu den Graphen aus der theoretischen Informatik erkennbar [Schöning, 1995].

Definition: Als *Graph* $G = (V,E)$ bezeichnet man eine Menge von *Knoten* V, die durch eine Menge von *Kanten* $E \subseteq V \times V$ verbunden werden. Weisen die Kanten eine Richtung auf, spricht man von einem *gerichteten Graph*. Ein *Weg* in einem Graphen ist eine Folge von Knoten $(v_0, v_1, \ldots, v_k)$ aus V, die durch Kanten verbunden sind, so daß gilt $(v_{i-1}, v_i) \in E$ für $i = (1, \ldots, k)$. Tritt innerhalb eines Wegs kein Knoten mehrfach auf, wird dieser Weg als *einfacher Weg* bezeichnet. Ein einfacher Weg heißt *Zyklus*, falls $v_0 = v_k$. Ein Graph wird als *azyklisch* bezeichnet, falls in ihm kein Zyklus existiert. Die Knoten eines gerichteten, azyklischen Graphen lassen sich so durchnumerieren, daß für jede Kante $(u,v) \in E$ gilt: Nummer von $u <$ Nummer von v. Man nennt eine solche Numerierung (oder Anordnung) der Knoten eine *topologische Sortierung*.

Durch die Parameterabbildungen ergibt sich also ein gerichteter azyklischer Graph von den Eingabeparametern der globalen Funktion über die Parameter der lokalen Funktionen zu den Ausgabeparametern der globalen Funktion. Die Knoten werden dabei durch die Parameter repräsentiert, während die (modellierten oder sich implizit ergebenden) Parameterabbildungen als Kanten angesehen werden können. Im folgenden werden derartige Graphen auch als Abhängigkeitsgraphen bezeichnet.

3.1.4 Ausführungsreihenfolge

Wie man in Abbildung 3.2 erkennen kann, ist die explizite Modellierung der Ausführungsreihenfolge, d.h. die Reihenfolge in der Quell- bzw. Hilfsfunktionen aufgerufen werden, nicht notwendig. Diese ergibt sich implizit durch die Parameterabhängigkeiten. Die Ausführungsreihenfolge kann durch eine topologische Sortierung auf dem Graphen anhand der Reihenfolge der sortierten Knoten (also der Parameter) bestimmt werden.

Betrachtet man die zur Modellierung der globalen Funktion `get_part()` nötigen Abhängigkeitsrelationen[2] (Abbildung 3.3), so zeigt sich, daß zwischen den beiden lokalen Funktionen `get_part()` und `get_price()` keine Abhängigkeiten bestehen und damit die beiden Funktionen parallel ausgeführt werden können. Allgemein ist die parallele Ausführung von lokalen Funktionen immer dann möglich, wenn auf dem Abhängigkeitsgraph kein Weg zwischen ihren Parametern existiert. Somit ist keine explizite Modellierung paralleler Ausführung notwendig. Diese erfolgt, soweit es die Abhängigkeiten und die Ausführungszeiten der einzelnen Funktionen ermöglichen, automatisch.

Sind innerhalb eines Abhängigkeitsgraphen lokale Funktionen von den Seiteneffekten anderer lokaler Funktionen abhängig, ohne von deren Parametern abhängig zu sein, muß diese Tatsache im Abhängigkeitsgraph durch eine explizite Abhängigkeit zwischen diesen beiden Funktionen modelliert werden. Ebenso können Abhängigkeiten zwischen Funktionen dazu benutzt werden, parallele Ausführung zu verhindern (falls etwa ein System parallele Ausführung nicht oder nur unter starken Geschwindigkeitseinbußen erlaubt).

[2] Die Abbildung zeigt eine etwas vereinfachte Version. Die vollständige Modellierung illustriert Abbildung 3.25 auf Seite 39.

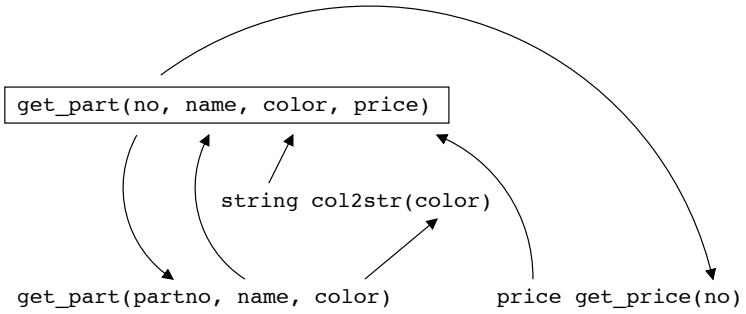


Abbildung 3.3: Parallele Ausführung.

Die Modellierung von Funktionsabhängigkeiten erfolgt dabei analog zu Abhängigkeiten zwischen Parametern. Eine Abhängigkeit `funktion_a()` → `funktion_b()` hat dabei die Bedeutung "`funktion_b()` wird erst aufgerufen, wenn `funktion_a()` korrekt ausgeführt wurde". Abbildung 3.4 zeigt eine Modellierung der globalen Funktion `get_part()` ohne parallele Ausführung.

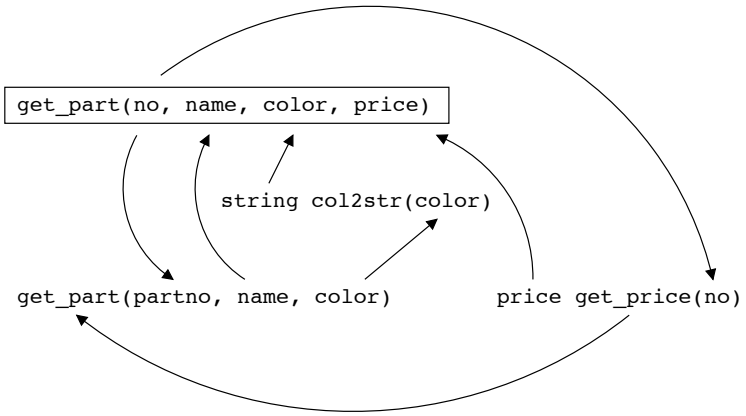


Abbildung 3.4: Vermeidung paralleler Ausführung.

Sollen Funktionen, die nur eine Best-Effort-Semantik garantieren können (z.B. IDL `oneway` Funktionen) in einen Abhängigkeitsgraph integriert werden, ändert sich die Bedeutung der Abhängigkeit `funktion_a()` → `funktion_b()` in "`funktion_b()` wird nach `funktion_a()` aufgerufen". Es kann insbesondere nicht garantiert werden, daß die Ausführung einer Funktion mit Best-Effort-Semantik bereits beendet wurde bzw. fehlerfrei beendet wurde.

Tabelle 3.1 faßt die bisher beschriebenen Abhängigkeitsrelationen und deren Bedeutungen zusammen.

3.1.5 Konstanten

Konstanten werden im Rahmen der Funktionsintegration immer dann benötigt, wenn für Eingabeparameter von Quellfunktionen keine Werte aus den Eingabeparametern der Zielfunktion zur Verfügung stehen. Entweder sind die notwendigen Konstanten bereits im jeweiligen Quellsystem definiert oder es müssen, analog zu den Hilfsfunktionen, sogenannte Hilfskonstanten definiert werden.

Den ersten Fall illustriert die `fseek()` Funktion der C Standard-Bibliothek [C] zum Setzen der Dateiposition. Als dritten Parameter erwartet sie den Ursprung zu dem die neue Position relativ gesetzt wer-

Parameter 1 → Parameter 2	Der Wert von Parameter 1 wird beim Aufruf der zu Parameter 2 gehörenden Funktion als Wert für Parameter 2 eingesetzt.
Funktion 1 → Funktion 2	Funktion 2 wird nur (und nur dann) ausgeführt, nachdem die Ausführung von Funktion 1 fehlerfrei abgeschlossen wurde.
Funktion 1 (Best-Effort) → Funktion 2	Funktion 2 wird nach Funktion 1 aufgerufen.

Tabelle 3.1: Grundlegende Abhängigkeitsrelationen.

den soll. Als Werte kommen die in `stdio.h` (also im Quellsystem) definierten `SEEK_SET`, `SEEK_CUR` oder `SEEK_END` in Frage.

Ein Beispiel für den zweiten Fall tritt bei der Modellierung der globalen Funktion `all_parts()` unter Verwendung der `all_parts()` Funktion des Anwendungssystems von Werk 1 auf. Diese Funktion erwartet als Parameter die Angabe einer Sparte, die von der globalen `all_parts()` Funktion allerdings nicht zur Verfügung gestellt wird. Da das globale System nur zur Integration der Automobil-Sparte dienen soll[3], muß der lokalen Funktion in jedem Fall der Wert `"kfz"` übergeben werden. Abbildung 3.5 enthält die zugehörige Abbildungsbeschreibung.

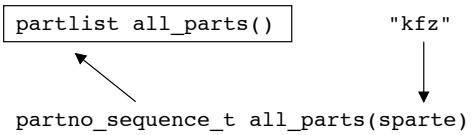


Abbildung 3.5: Konstanten.

Normalerweise erfolgt die Definition von Konstanten innerhalb der Schnittstellenbeschreibung des Anwendungssystems. Somit ist eigentlich keine Modellierung der Werte von Konstanten im Rahmen der Funktionsintegration notwendig. Soll aber, etwa wie im Fall der in dieser Arbeit entwickelten Beschreibungssprache FIX, Funktionsintegration und Schnittstellenbeschreibung vereinigt werden, muß dies auch mit berücksichtigt werden (siehe auch Abschnitt 4.9.6).

Neben der beschriebenen Abhängigkeit Konstante → Parameter ist zusätzlich die Abhängigkeit Funktion → Konstante möglich. Tabelle 3.2 listet die in Verbindung mit Konstanten hinzugekommenen möglichen Abhängigkeitsrelationen und deren Bedeutungen auf.

Konstante → Parameter	Der Wert der Konstanten wird beim Aufruf der zu dem Parameter gehörenden Funktion als dessen Wert eingesetzt.
Funktion → Konstante	Die Funktion muß zuerst erfolgreich ausgeführt worden sein, bevor von der Konstante ausgehende Abhängigkeitsrelationen aufgelöst werden können.

Tabelle 3.2: Konstanten in Abhängigkeitsrelationen.

[3] zugegebenermaßen kein gutes Beispiel für eine umfassende Integration...

3.1.6 Referenzierung von Teilen von Parametern

Neben den in den bisherigen Beispielen benutzten einfachen Datentypen können Parameter auch komplexe Datentypen, etwa Arrays oder Strukturen enthalten. Oft wird innerhalb eines Abhängigkeitsgraphen nur ein bestimmter Wert aus einer Struktur benötigt. Ebenso sind Fälle denkbar, in denen auf ein bestimmtes Element innerhalb eines Arrays zugegriffen werden muß.

Ohne zusätzliche Konstrukte ist der Zugriff auf Komponenten von komplexen Datentypen mit Hilfe von Hilfsfunktionen möglich. Das Beispiel aus Abbildung 3.6 zeigt die Realisierung der globalen Funktion `get_size()` unter Verwendung der `get_part()` Funktion aus Werk 2. Bei dem Parameter `dim` handelt es sich dabei um eine Struktur mit den drei Elementen `x`, `y` und `z`.

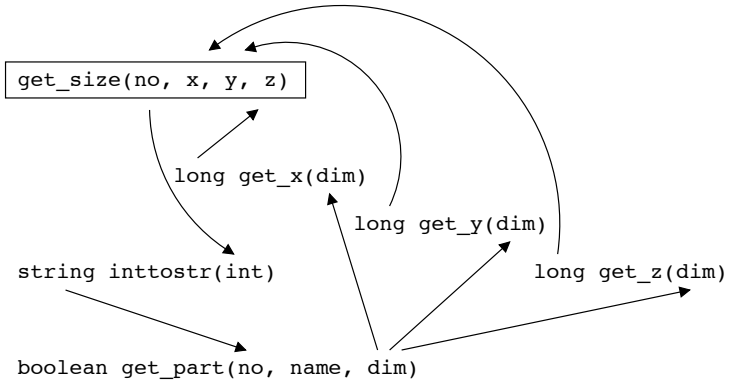


Abbildung 3.6: Verarbeitung von Strukturkomponenten durch Hilfsfunktionen.

Anhand dieses Beispiels werden die Nachteile dieses Vorgehens deutlich. Es wird eine große Anzahl verschiedener, nicht wiederverwendbarer Hilfsfunktionen benötigt. Aus diesem Grund wird ein Konstrukt benötigt, um, wie in Abbildung 3.7 illustriert, direkt auf Komponenten von komplexen Parametern zugreifen zu können. Abschnitt 4.12 beschreibt die innerhalb der Sprache FIX gewählte Realisierung.

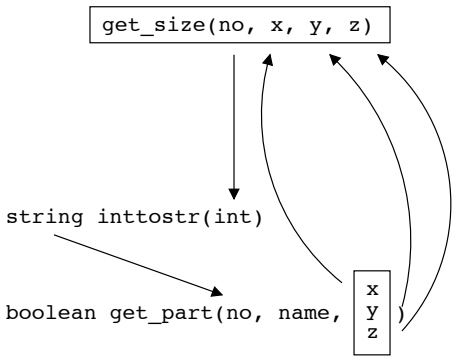


Abbildung 3.7: Direkte Verarbeitung von Strukturkomponenten.

3.1.7 Mehrfache Instantiierung von Funktionen

Wird innerhalb einer Abbildungsbeschreibung eine Funktion mehrfach benötigt (etwa eine Konvertierungsfunktion), müssen die verschiedenen Instanzen als solche unterscheidbar sein. In Abbildung 3.8 wurde dies z.B. durch die angefügten Indizes realisiert.

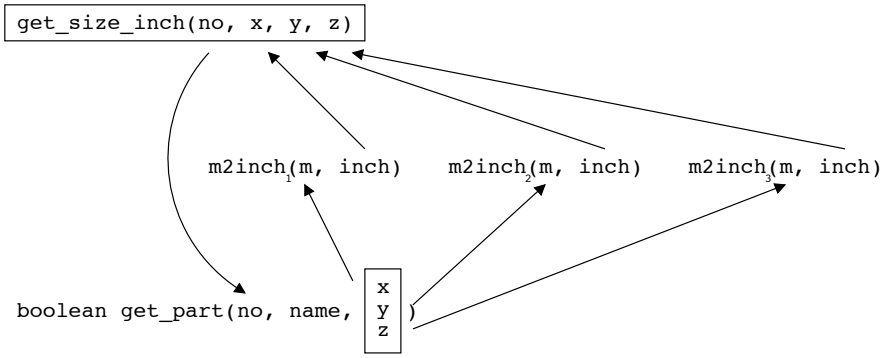


Abbildung 3.8: Mehrfache Instantiierung von Funktionen.

Erfolgt diese Unterscheidung nicht, so ergibt sich das in Abbildung 3.9 dargestellte Diagramm, dessen Sinn sich zwar für den Betrachter sofort erschließt, auf einem Computer aber kaum zu dem gewünschten Ergebnis führen würde.

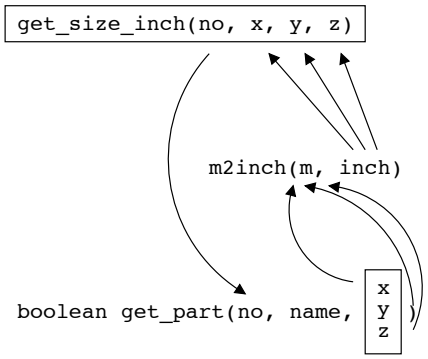


Abbildung 3.9: Ungültige mehrfache Benutzung einer Funktion.

3.1.8 Textuelle Beschreibungen innerhalb eines Abhängigkeitsgraphen

Es sind Problemstellungen denkbar, die nicht (oder nur unter erheblichem Aufwand) durch Abhängigkeiten und Hilfsfunktionen beschreibbar sind. In einem solchen Fall ist es wünschenswert, daß eine Möglichkeit existiert, eine textuelle Beschreibung des Problems in die Abbildungsbeschreibung zu integrieren. Soll dann z.B. aus einer Abbildungsbeschreibung ein lauffähiges Zielsystem erzeugt werden, muß ein Programmierer die textuell beschriebenen Teile “von Hand” ausprogrammieren.

Textuelle Beschreibungen dürfen nicht losgelöst innerhalb einer Abhängigkeitsbeschreibung vorkommen, sondern müssen in den Graph mit eingebunden werden. Damit wird erreicht, daß bei der automatischen Generierung von Zielsystemen die Platzhalter für den vom Programmierer einzufügenden

Code an der richtigen Stelle im erzeugten Code stehen. Es muß nur der wirklich notwendige Code vom Programmierer selbst entwickelt werden, der Rest kann weiterhin aus dem Abhängigkeitsgraph automatisch generiert werden.

Die Einbindung in den Graph wird dadurch erreicht, daß die textuelle Beschreibung von sämtlichen Parametern abhängig ist, die innerhalb des beschriebenen Bereichs verändert werden. Sämtliche nachfolgenden Parameter, die innerhalb des beschriebenen Bereichs verändert bzw. erzeugt werden, sind wiederum von der textuellen Beschreibung abhängig. Sollte z.B. der Wunsch bestehen, die globale Funktion `get_color()` (siehe Abbildung 3.2 auf Seite 21) ohne die Verwendung von Hilfsfunktionen zu realisieren, könnte die Abbildungsbeschreibung aus Abbildung 3.10 benutzt werden[4]. Abbildung 3.11 zeigt, wie der aus dieser Abbildungsbeschreibung generierte Code aussehen könnte.

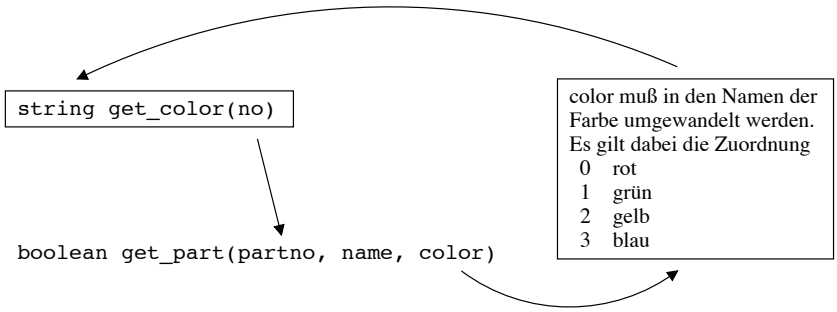


Abbildung 3.10: Textuelle Beschreibung.

```
string get_color(in long no) {

  color_t color;
  string result;

  get_part(no, NULL, color);

  /* Abhängigkeit von color
   *
   * color muß in den Namen der
   * Farbe umgewandelt werden.
   * Es gilt dabei die Zuordnung
   *   0  rot
   *   1  grün
   *   2  gelb
   *   3  blau
   *
   * Abhängigkeit zu result
   */

  return result;
}
```

Abbildung 3.11: Aus textueller Beschreibung generierter Code.

[4]Eine dritte Möglichkeit zur Realisierung von `get_color()` wird in Abschnitt 3.3.1 besprochen.

Tabelle 3.3 illustriert die möglichen Verknüpfungen von textuellen Beschreibungen mit den bisher beschriebenen Parametern, Funktionen und Konstanten. Textuelle Beschreibungen können prinzipiell mit allen Knotentypen verknüpft werden

Parameter → Text	Der Parameter wird innerhalb der textuellen Beschreibung verarbeitet.
Text → Parameter	Innerhalb der textuellen Beschreibung erfolgt eine Zuweisung zu diesem Parameter.
Funktion → Text	Die Funktion muß vor der textuellen Beschreibung ausgeführt werden.
Text → Funktion	Die Funktion darf erst nach der textuellen Beschreibung ausgeführt werden.
Text → Konstante	Die Abarbeitung der textuellen Beschreibung muß beendet sein, bevor von der Konstante ausgehende Abhängigkeitsrelationen aufgelöst werden können.
Konstante → Text	Der Wert der Konstanten wird innerhalb der textuellen Beschreibung benutzt.

Tabelle 3.3: Verknüpfung textueller Beschreibungen.

3.2 Objektorientierung

Schnittstellen von objektorientierten Systemen basieren im Gegensatz zu konventionellen Systemen nicht auf den exportierten Funktionen, statt dessen führen sie den Begriff eines Objekts ein. Ein Objekt dient dabei zur Kapselung von Daten und Code. Objekte stellen ihre Schnittstelle in Form von Methoden und Attributen zur Verfügung. Methoden gleichen dabei den bisher betrachteten Funktionen. Attribute verhalten sich wie Variablen einer Programmiersprache, mit dem Unterschied, daß ein lesender oder schreibender Zugriff nicht nur zur Änderung des Werts der Variablen, sondern auch weitere Aktionen zur Folge haben kann.

Zusätzlich besteht die Möglichkeit, daß mehrere Objekte des selben Typs gleichzeitig instantiiert werden, d.h. es existieren mehrere Versionen der selben Schnittstelle, die u.U. verschiedene Zustände besitzen, und damit unterschiedliche Ergebnisse liefern. Unterstützt das jeweilige System nicht nur die entfernte Instantiierung von Objekten, sondern auch die Verwendung der Objekte innerhalb entfernter oder lokaler Methoden, muß auch innerhalb von Abhängigkeitsgraphen die Verwendung von Objekten als Parameter möglich sein.

3.2.1 Lokale Attribute

Attribute verhalten sich gegenüber dem benutzenden Programm ähnlich wie globale Variablen. Es ist möglich, Werte zu schreiben und wieder auszulesen. Ebenso besteht die Möglichkeit, daß der Wert eines Attributs von einer aufgerufenen Methode verändert wird. Im Gegensatz zu Variablen können aber durch einen Zugriff auf ein Attribut weitere Aktionen ausgelöst werden.

Die Einbindung von Quellsystemattributen erfolgt ähnlich zu der von Parametern oder Konstanten. Ein Attribut kann dabei sowohl Ausgangspunkt (lesender Zugriff) als auch Ziel (schreibender Zugriff) einer Abhängigkeitsrelation sein. Des weiteren besteht die Möglichkeit, daß Attribute ähnlich wie Konstanten in Abhängigkeitsrelationen mit Funktionen auftreten können.

Eine Abhängigkeitsrelation der Form Attribut $\rightarrow$ Parameter ist prinzipiell jederzeit erfüllt (ein Attribut hat im Gegensatz zu einem Parameter immer einen Wert). Ist der Wert eines Attributs abhängig von einem Seiteneffekt einer vorangegangenen Funktion, muß eine zusätzliche Abhängigkeitsrelation Funktion $\rightarrow$ Attribut eingeführt werden, damit die dem Attribut folgenden Abhängigkeitsrelationen erst aufgelöst werden, wenn das Attribut den richtigen Wert besitzt.

Die Abbildung 3.12 zeigt die Modellierung der globalen Funktion `get_price_euro()`. Diese benutzt die Funktion `get_price()` des Lieferantensystems, welche abhängig vom Wert des lokalen Attributs `currency` den Preis in der entsprechenden Währung zurück gibt.

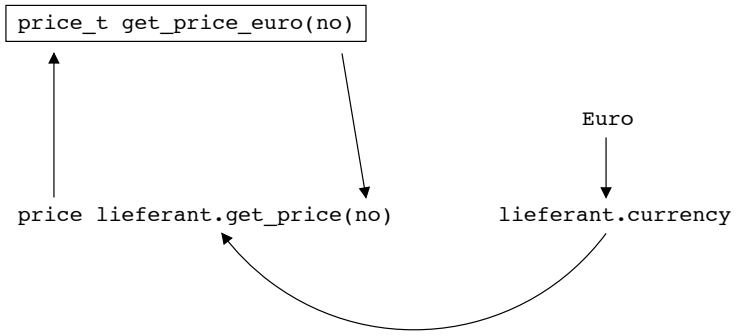


Abbildung 3.12: Lokale Attribute innerhalb eines Abbildungsgraphs.

Tabelle 3.4 zeigt die durch die Einführung von Attributen neu hinzugekommenen Möglichkeiten von Abhängigkeitsrelationen.

3.2.2 Globale Attribute

Zur Realisierung von globalen Attributen ist die Modellierung von Abhängigkeitsgraphen für Attribute analog zu denen für globale Funktionen notwendig. Es muß jeweils ein Abhängigkeitsgraph für den lesenden und schreibenden Zugriff erstellt werden.

Innerhalb objektorientierter Programmiersprachen werden Attribute meist durch zwei (nach außen nicht sichtbare) Methoden implementiert, jeweils eine für den lesenden bzw. schreibenden Zugriff. So erfolgt die Repräsentation des globalen Attributs

```
currency_t currency;
```

aus dem Beispiel durch die beiden Zugriffsfunktionen

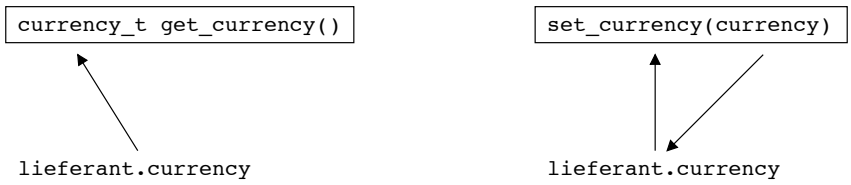
```
currency_t get_currency();
set_currency(in currency_t currency);
```

Parameter → Attribut	Der Wert des Parameters wird in das Attribut geschrieben.
Konstante → Attribut	Der Wert der Konstanten wird in das Attribut geschrieben.
Attribut 1 → Attribut 2	Der Wert des ersten Attributs wird in das zweite Attribut geschrieben.
Attribut → Parameter	Der Wert des Attributs wird beim Aufruf der zu dem Parameter gehörenden Funktion als dessen Wert eingesetzt.
Attribut → Funktion	Die zu dem Attribut hinführenden Abhängigkeitsrelationen müssen erfolgreich aufgelöst worden sein, bevor die Funktion ausgeführt werden kann.
Attribut → Konstante	Die zu dem Attribut hinführenden Abhängigkeitsrelationen müssen erfolgreich aufgelöst worden sein, bevor die von der Konstanten ausgehenden Abhängigkeitsrelationen aufgelöst werden können.
Attribut → Text	Das Attribut wird innerhalb der textuellen Beschreibung verarbeitet.
Text → Attribut	Innerhalb der textuellen Beschreibung erfolgt eine Zuweisung zu diesem Attribut.

Tabelle 3.4: Abhängigkeiten in Verbindung mit Attributen.

Zur Realisierung von Attributen in Zielsystemen ist also die Modellierung dieser beiden Funktionen nötig. Abbildung 3.13 zeigt die mögliche Modellierung des globalen Attributs `currency` durch das gleichnamige lokale Attribut des Lieferantensystems.

Die Abhängigkeitsrelation `currency` → `lieferant.currency` aus Abbildung 3.13b modelliert dabei die eigentliche Wertzuweisung, während `lieferant.currency` → `set_currency` zur Vervollständigung des Abhängigkeitsgraphen dient, damit dieser von einer Ausführungsengine richtig aufgelöst werden kann.

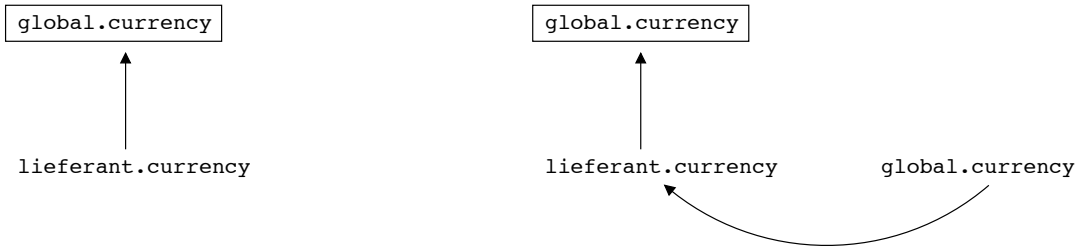


(a) Lesender Zugriff (b) Schreibender Zugriff

Abbildung 3.13: Modellierung von Zielsystemattributen (I).

Soll die Verwendung zusätzlicher Funktionen zum Setzen bzw. Auslesen des globalen Attributs vermieden werden, muß das globale Attribut selbst in den Abhängigkeitsgraph integriert werden. Der lesende Zugriff erfolgt ähnlich zu der Ausführung einer Funktion, das Attribut wird dabei analog zu dem Rückgabewert einer globalen Funktion in den Abhängigkeitsgraph integriert (Abbildung 3.14a).

Der schreibende Zugriff stellt insofern eine Besonderheit dar, als daß das globale Attribut an zwei Stellen innerhalb des Abhängigkeitsgraphs auftaucht (Abbildung 3.14b). Zum einen in der Abhängigkeitsrelation `global.currency` → `lieferant.currency`, hierbei erfolgt die Verwendung von `global.currency` analog zu einem Eingabeparameter einer Zielfunktion. Zum anderen in der umgekehrten Form `lieferant.currency` → `global.currency` stellvertretend für die Zielsystemfunktion selbst, um den Abhängigkeitsgraph zu vervollständigen. Diese zweite Abhängigkeit drückt dabei keine Wertübergabe o.ä. aus, sondern dient allein dazu, dem Abhängigkeitsgraph das richtige Ziel zu geben.



(a) Lesender Zugriff (b) Schreibender Zugriff

Abbildung 3.14: Modellierung von Zielsystemattributen (II).

3.2.3 Objekte als Parameter

Werden Objekte innerhalb eines Abhängigkeitsgraphs nur als Container für Methoden oder Attribute benutzt, werden diese also insbesondere nicht mehrfach instantiiert oder als Parameter übergeben, können objektorientierte Systeme wie bisher beschrieben integriert werden. Es müssen keine neuen Konstrukte eingeführt werden. Ein Beispiel für diese Art des objektorientierten Ansatzes ist die Verwendung von Interfaces in CORBA.

Werden Objekte auch als Parameter benutzt (dazu zählt auch die Erzeugung eines Objekts z.B. durch einen Konstruktor), entsteht das Problem, daß, ähnlich wie in Abschnitt 3.1.6 beschrieben, Abhängigkeitsrelationen auf Teile von Parametern — in diesem Fall Methoden oder Parameter von Methoden — verweisen müssen. Abbildung 3.15 zeigt ein Beispiel, in welchem eine globale Funktion durch zwei lokale Objekte realisiert wird. Den dadurch repräsentierten Code zeigt Abbildung 3.16.

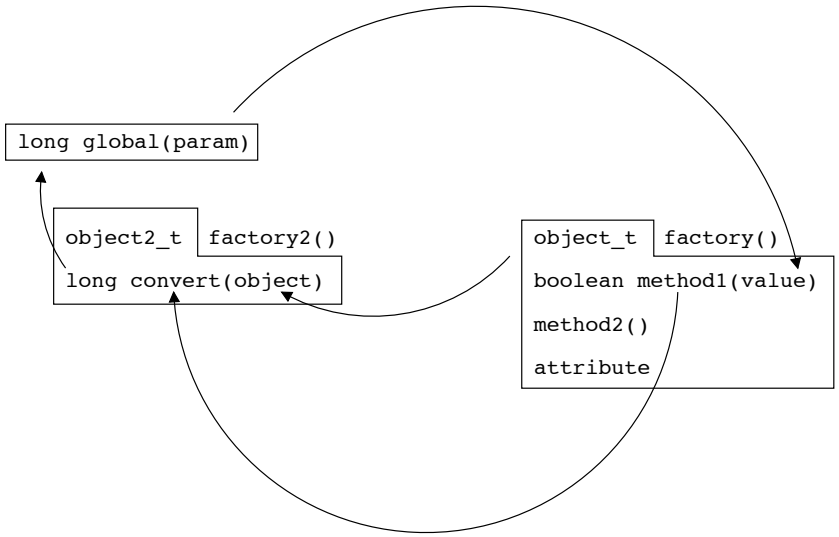


Abbildung 3.15: Integration von Objekten (I).

```
long global(in param_t param) {

  object_t obj1;
  object2_t obj2;
  result long;

  obj1 = factory();
  obj1.method1(param);

  obj2 = factory2();
  result = obj2.convert(obj1);

  return result;
}
```

Abbildung 3.16: Integration von Objekten (II).

Werden innerhalb eines Abhängigkeitsgraphen mehrere Instanzen eines Objekts benutzt, wird dies durch die Verwendung mehrerer Konstruktoren erreicht (die ihrerseits, wie in Abschnitt 3.1.7 beschrieben, unterschieden werden müssen).

Da Konstruktoren im allgemeinen nicht explizit mit einem Rückgabeparameter definiert werden, muß eine neue Abhängigkeitsrelation Konstruktor → Parameter eingeführt werden (siehe Tabelle 3.5). Ansonsten treten bei der Verwendung von Objekten als Parameter keine neuen Typen von Abhängigkeitsrelationen auf, da sie genau wie Parameter mit normalen Datentypen behandelt werden.

Konstruktor → Parameter	Konstruktoren besitzen i.a. keinen Rückgabewert, deshalb muß in Abhängigkeitsrelationen auf den Konstruktor selbst verwiesen werden.

Tabelle 3.5: Abhängigkeitsrelation Konstruktor → Parameter.

3.3 Kontrollflußsteuerung

In den bisher aufgeführten Beispielen ergab sich die Reihenfolge, in der die modellierten Abhängigkeiten aufgelöst werden müssen rein aus der Art und Weise, wie Parameter bzw. Funktionen miteinander durch Abhängigkeitsrelationen verknüpft wurden. Die Aufrufreihenfolge der lokalen Funktionen wurde dadurch von vornherein durch die Modellierung der Abhängigkeiten festgelegt. Mehrfache oder bedingte Ausführung von Teilen des Abhängigkeitsgraphs wurden nicht verwendet.

Programmiersprachen bieten zu diesem Zweck Konstrukte wie Schleifen oder bedingte Anweisungen an. Die Übertragung dieser Konstrukte auf Abhängigkeitsgraphen soll in den nächsten Abschnitten besprochen werden. Des weiteren sollen in diesem Abschnitt weitergehende Aspekte zur parallelen Ausführung betrachtet werden.

3.3.1 Bedingte Ausführung

Bedingte Anweisungen werden innerhalb von Programmiersprachen meist durch `if then` Konstrukte realisiert. Diese bestehen aus einer Bedingung und dem Code, der ausgeführt werden soll, falls diese Bedingung zutrifft. Oft wird weitergehende Funktionalität durch `else` oder `else if` Klauseln, oder durch Konstrukte wie die C `case` Anweisung geboten. All diese erweiterten Konstrukte lassen sich aber auf die einfache Form einer `if then` Anweisung zurückführen (im Fall der `else` Klausel z.B. durch die Verwendung einer zweiten `if then` Anweisung mit negierter Bedingung).

Abbildung 3.17 zeigt die notwendigen Abhängigkeiten zur Modellierung der globalen Funktion `get_color()`. Um im Rahmen der Funktionsintegration kein neues Konstrukt für bedingte Anweisungen zu verwenden, wird sowohl der Bedingungs- als auch der Code-Teil einer bedingten Anweisung als Funktion modelliert. Dabei hängt die “Codefunktion” von der “Bedingungsfunktion” ab. Abbildung 3.18 zeigt eine derartige Modellierung von `get_color()`. Dabei vergleicht die Funktion `isequal()` die beiden übergebenen Werte. Sind die beiden Werte gleich, kehrt sie zurück, andernfalls bricht sie mit einem Fehler ab (siehe auch Abschnitt 3.4).

Die drei Instanzen der Funktion `isequal()` können hierbei parallel ausgeführt werden. Schlagen während der parallelen Ausführung eine oder mehrere Funktionen fehl, wird die Ausführung des

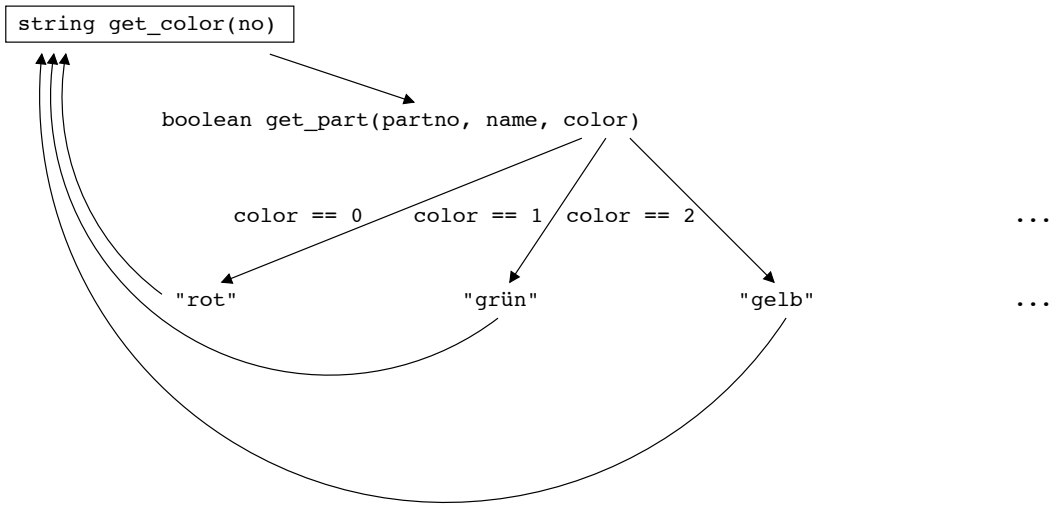


Abbildung 3.17: Bedingte Ausführung (I).

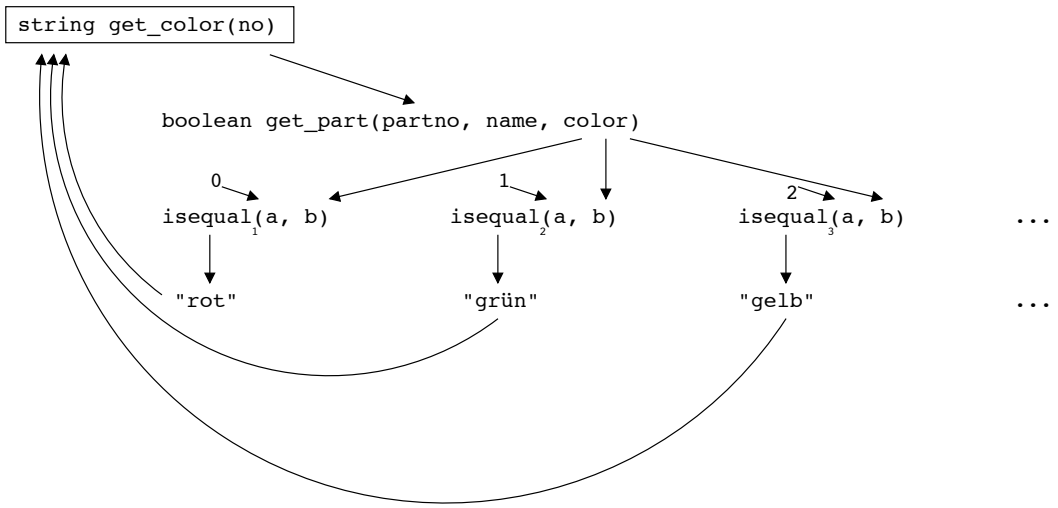


Abbildung 3.18: Bedingte Ausführung (II).

Abhängigkeitsgraphs nicht sofort abgebrochen. Der Abbruch erfolgt in diesem Fall nur, wenn alle der vorhandenen `isequal()` Funktionen fehlschlagen.

Allgemein wird versucht, Abhängigkeiten zu Rückgabewerten, -parametern und der Zielfunktion über jeden möglichen Weg zu erfüllen. Erst wenn dies nicht mehr möglich ist, bricht die Ausführung des Abhängigkeitsgraphs mit einer Fehlermeldung ab.

Als Bedingungsfunktion können prinzipiell sowohl Hilfs- als auch sämtliche Quellfunktionen dienen. Vergleichsfunktionen, wie die im Beispiel verwendete `isequal()`, werden dabei besonders häufig benötigt. Aus diesem Grund sollten solche Funktionen in einer Art Standardbibliothek (siehe Abschnitt 3.7.1) vorhanden sein.

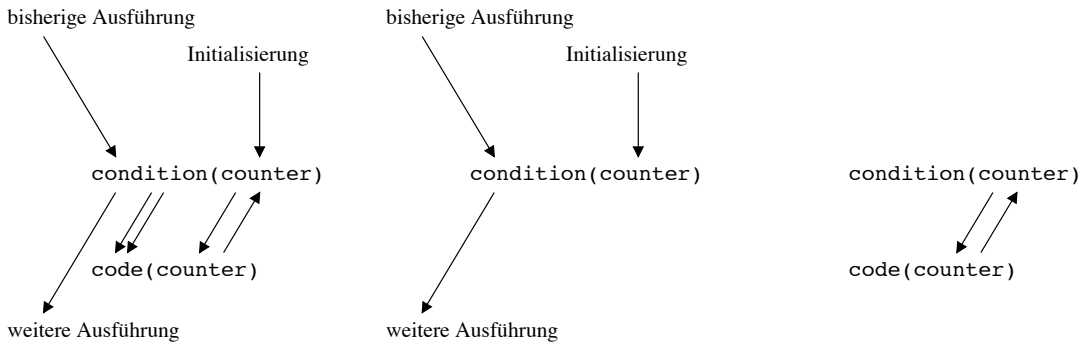
3.3.2 Zyklischer Fall bzw. Schleifen

Schleifenkonstrukte in Programmiersprachen dienen dazu, die Ausführung bestimmter Programmteile zu wiederholen. Ähnlich den bedingten Anweisungen bestehen Schleifen aus einer Bedingung und dem zu wiederholenden Code. Im Gegensatz zur bedingten Anweisung wird der Code aber nicht nur einmal ausgeführt, vielmehr wird nach der Ausführung des Codes die Bedingung erneut ausgewertet und der Code gegebenenfalls erneut ausgeführt. Es existieren eine Reihe verschiedener Varianten von Schleifen in den verschiedenen Programmiersprachen. Neben der am häufigsten anzutreffenden `while` Schleife, bei der die Bedingung vor der ersten Ausführung des Codes ausgewertet wird, existiert die `repeat` Variante, bei der die Auswertung der Bedingung erst nach Ausführung des Codes erfolgt und dadurch der Code auf jeden Fall mindestens einmal abgearbeitet wird. Die `for` Schleife aus Pascal und anderen Programmiersprachen stellt eine Besonderheit dar, da sie bereits Teile der Bedingung und des Codes vorgibt. Prinzipiell kann aber jede Schleife auf die einfache Form einer `while` Schleife zurückgeführt werden.

Die Modellierung von Schleifen innerhalb eines Abhängigkeitsgraphen erfolgt ähnlich zu der Modellierung von bedingten Anweisungen. Die Bedingung wird als Aufruf einer lokalen (oder Hilfs-) Funktion modelliert und durch eine “zyklische Abhängigkeitsrelation” (in den folgenden Beispielen durch einen Doppelpfeil symbolisiert) mit dem Code der Schleife — der aus einem oder mehreren Funktionsaufrufen bestehen kann — verbunden. Den Aufbau einer einfachen Schleife verdeutlicht Abbildung 3.19.

Die zyklische Abhängigkeitsrelation hat dabei große Ähnlichkeit zu den in Abschnitt 3.4.4 behandelten Kompensationsaktionen. Ebenso wie diese ist eine zyklische Abhängigkeit zunächst inaktiv. Führt die Ausführung der Funktion `condition()` aus Abbildung 3.19a zu keiner Exception wird die Ausführung des Abhängigkeitsgraphs in Richtung “weitere Ausführung” fortgesetzt (Abbildung 3.19b); tritt aber eine Exception auf, wird die zyklische Abhängigkeit aktiv und die Funktion, von der sie ausgeht, wird zur neuen, temporären Zielfunktion. (Abbildung 3.19c). Nach der Abarbeitung dieses neuen Graphs erfolgt die erneute Ausführung der Bedingungsfunktion. Eine mögliche Umsetzung dieses Graphs in eine normale Programmiersprache illustriert Abbildung 3.20.

Die Realisierung der Schleife erfolgte in diesem Beispiel durch die beiden spezialisierten Hilfsfunktionen `condition()` und `code()`. Diese Funktionen können an keiner anderen Stelle wiederverwendet werden. Abbildung 3.21a zeigt die Modellierung einer Schleife ohne die Verwendung spezialisierter Hilfsfunktionen. Auch in diesem Fall, wird die zyklische Abhängigkeitsrelation erst beim Auftreten einer Exception aktiv und es ergibt sich der in Abbildung 3.21b dargestellte neue Abhängigkeitsgraph mit der temporären Zielfunktion `eval()`.

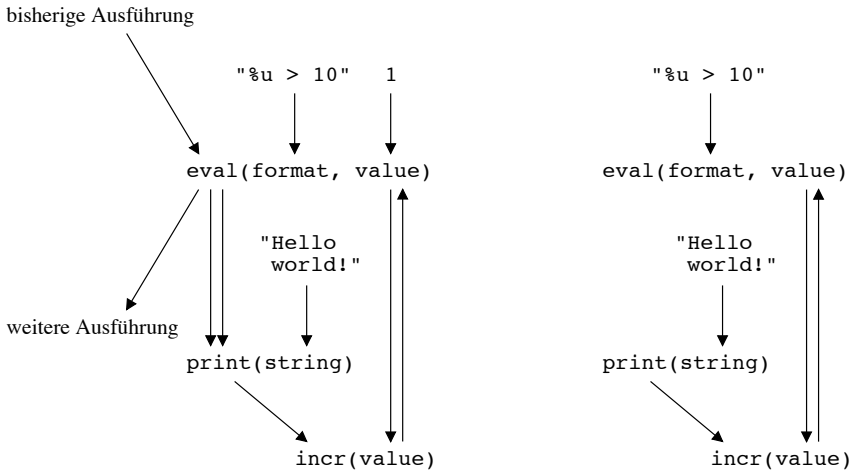


(a) Modellierung (b) keine Exception (c) Exception

Abbildung 3.19: Einfache Schleife.

```
void condition(long &counter) throw (EWRONG) {
  if (counter < 10)
    throw EWRONG;
}

void code(long &counter) {
  printf("Hello world!");
  counter++;
}

void global() {

  int counter=0;

  ...

  while(1) {
    try {
      condition(counter);
    }
    catch (int i) {
      if (i==EWRONG) {
        code(counter);
        continue;
      } else
        throw;
    }
    break;
  }

}
```

Abbildung 3.20: Umsetzung einer zyklischen Abhängigkeit in C++.



(a) Modellierung (b) Exception

Abbildung 3.21: Einfache Schleife ohne spezialisierte lokale Funktionen.

Abbildung 3.21 macht deutlich, daß die Realisierung (auch einfacher) Schleifen durch Abhängigkeitsgraphen relativ aufwendig ist. Aus diesem Grund wurden die zwei in Abbildung 3.22 dargestellten Sonderfälle zyklischer Abhängigkeiten definiert. Es wird dabei eine Liste und ein einfacher Datentyp durch eine zyklische Abhängigkeit verbunden. Dabei müssen die Datentypen der Listenelemente mit dem Datentyp des Parameters der anderen Funktion übereinstimmen.

Abbildung 3.22: Zyklische Sonderfälle.

Der erste Sonderfall hat dabei die Bedeutung, daß `func2()` mit jedem Wert der Liste einmal aufgerufen wird. Im zweiten Fall wird `func1()` sooft ausgeführt, bis eine Exception aufgetreten ist. Sämtliche zurückgegebenen Werte werden in einer Liste zusammengefaßt und `func2()` als Parameter übergeben. Mit Hilfe dieser beiden Konstrukte ist die Realisierung von vielen häufig auftretenden Problemen, wie beispielsweise die Berechnung des Durchschnitts der Werte einer Liste, relativ einfach möglich.

Tabelle 3.6 faßt die im Rahmen des zyklischen Falls möglichen Abhängigkeitsrelationen zusammen.

3.3.3 Redundante Ausführung

Innerhalb eines Abhängigkeitsgraphen können u.U. Werte durch die Ausführung mehrerer verschiedener Funktionen ermittelt werden. So ist die Realisierung der Funktion `get_name()` unter Verwendung der lokalen `get_part()` Funktion sowohl von Werk 1 als auch von Werk 2 möglich. Werden im Rahmen einer globalen Funktion beide Quellfunktionen aufgerufen, kann dadurch eine Kompensation

Funktion 1 ⇒ Funktion 2	Schlägt die Ausführung von Funktion 1 fehl, wird diese Abhängigkeit aktiv und Funktion 1 zur temporären Zielfunktion. Nach der Abarbeitung des Zyklus wird Funktion 1 erneut ausgeführt.
Listenparameter ⇒ Parameter	Funktion des Zielparameters wird mit jedem Element der Liste einmal aufgerufen.
Parameter ⇒ Listenparameter	Ausgangsfunktion wird wiederholt aufgerufen, bis eine Exceptions auftritt. Alle erhaltenen Werte werden als Liste der zweiten Funktion übergeben.

Tabelle 3.6: Zyklische Abhängigkeiten.

des Ausfalls eines der beiden Systeme erfolgen. Außerdem wird die Ausführungsgeschwindigkeit optimiert, falls eines der beiden Quellsysteme im Augenblick der Ausführung überlastet ist. Eine mögliche Modellierung der Funktion `get_name()` illustriert Abbildung 3.23.

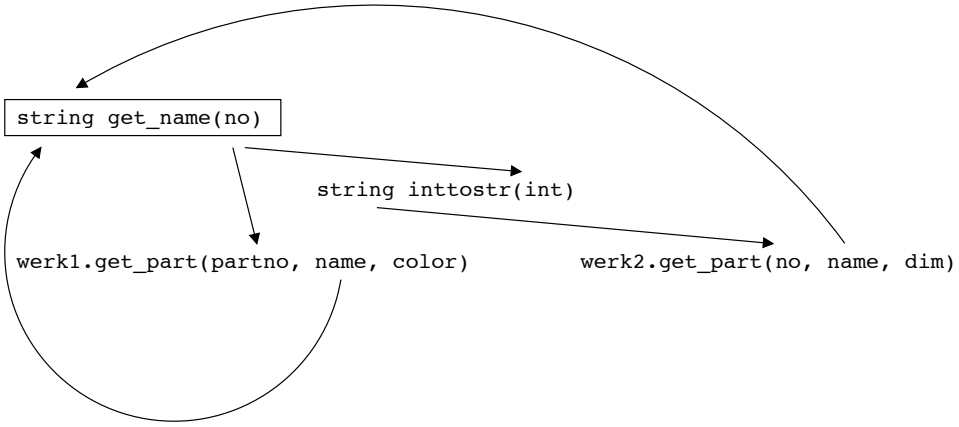


Abbildung 3.23: Redundante Ausführung.

Hier können die beiden lokalen `get_part()` Funktionen gleichzeitig ausgeführt werden. Sobald eine der beiden zurückkehrt, kann der Rückgabewert der globalen Funktion ermittelt werden. Die Ausführung des jeweils anderen Teilgraphs wird (falls möglich) abgebrochen.

Das gleiche Vorgehen ist auch bei einer Funktion, die von mehreren Funktionen abhängig ist, möglich. Im Gegensatz zu den Parametern, besteht aber in diesem Fall auch die Möglichkeit, daß eine Funktion erst aufgerufen werden soll, wenn alle Funktionen, von denen sie abhängig ist, ausgeführt wurden. Abbildung 3.24 illustriert dies am Beispiel der Integration der lokalen `init()` Funktionen. In der Abbildung wird die zwingende Ausführung der beiden lokalen Funktionen durch einen Kreis um die Pfeile symbolisiert.

Eine weitere Möglichkeit besteht darin, Abhängigkeiten als optional zu markieren. Ein solche Abhängigkeit wird erst aktiv, falls jede andere, nicht zwingend erforderliche, zu demselben Ziel hinführende Abhängigkeit nicht erfüllt werden konnte. Abbildung 3.25 zeigt die Benutzung der (durch eine nicht ausgefüllte Pfeilspitze dargestellten) optionalen Ausführung anhand der globalen Funktion

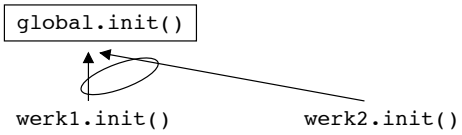


Abbildung 3.24: Doppelte Ausführung.

`get_part()`. Führt der Lieferant ein Teil nicht, dann wird die lokale Funktion `get_price()` mit einer Exception abgebrochen. Nur in diesem Fall tritt die optionale Abhängigkeit `0` $\rightarrow$ `price` in Kraft.

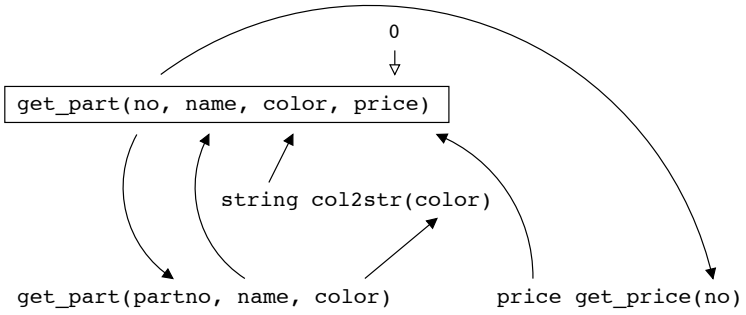


Abbildung 3.25: Optionale Ausführung.

3.4 Fehlerbehandlung

Die Behandlung von Fehlern muß unter mehreren Gesichtspunkten betrachtet werden. Zum einen muß es innerhalb eines Abhängigkeitsgraphen möglich sein, auf einen in einer Quellfunktion auftretenden Fehler zu reagieren oder eine entsprechende Fehlermeldung zurückzugeben. Voraussetzung dazu ist, daß ein Mechanismus zur Fehlererkennung vorhanden ist. Fehler können einerseits direkt durch die aufgerufene Quellfunktion zurückgemeldet werden (etwa durch Exceptions), andererseits können bestimmte Fehler, etwa falsche Daten, nur durch entsprechende bedingte Anweisungen erkannt werden.

3.4.1 Fehlermeldungen

In verschiedenen Programmiersprachen bzw. Systemen werden unterschiedliche Konzepte zur Signalisierung eines Fehlers innerhalb einer Funktion benutzt.

Warnmeldung: Die aufgerufene Funktion kann eine Warnmeldung auf dem Bildschirm ausgeben, ansonsten aber die Ausführung normal fortsetzen. Diese Art der Fehlerrückmeldung macht eigentlich nur als Warnmeldung bei Fehlern, die keinen größeren Einfluß auf die Programmausführung haben, Sinn. Großer Nachteil einer reinen Warnmeldung ist, daß die aufrufende Funktion den Fehler nicht sieht und deshalb nicht darauf reagieren kann.

Abbruch: Tritt ein gravierender Fehler auf, kann die aufgerufene Funktion die Ausführung komplett abbrechen.

Rückgabewert: Der Rückgabewert (oder evt. ein Rückgabeparameter) dient als Indikator, ob innerhalb der Funktion ein Fehler aufgetreten ist. Dabei kann es sich um einen einfachen Wahrheitswert handeln oder z.B. um eine konkrete Fehlernummer. Oft wird die Rückgabe von Daten oder Fehlermeldung innerhalb des Rückgabewerts kombiniert. In diesem Fall dient ein möglicher Wert (meist 0 oder -1) als Fehlerindikator, während in allen anderen Fällen, die gewünschten Daten zurückgegeben werden.

Diese Variante der Fehlermeldung wird meist (aber nicht ausschließlich) von nicht-objektorientierten Systemen gewählt. In einigen Fällen wird zusätzlich eine globale Variable verwendet.

Globale Variablen werden ebenso dazu benutzt, aufgetretene Fehler zu signalisieren. Oft wird in einer globalen Variablen nur der zuletzt *aufgetretene* Fehler gespeichert. Um also sicher feststellen zu können, ob beim Aufruf der letzten Funktion ein Fehler aufgetreten ist, muß die globale Variable vorher zurückgesetzt worden sein.

Die in C benutzte Variable `errno` ist ein bekanntes Beispiel für die Verwendung einer globalen Variable zur Signalisierung eines Fehlers.

Funktionen: Als Variante einer globalen Variable wird die aktuelle Fehlernummer als Rückgabewert einer Funktion zurückgeliefert. Ebenso wie unter Verwendung von globalen Variablen wird u.U. nur der zuletzt aufgetretene Fehler zurückgemeldet. Eine Rücksetzung des Zustandes erfolgt dabei durch den Aufruf einer weiteren Funktion.

Als Beispiel für die Fehlermeldung durch separate Funktionen können die im Zusammenhang mit dem Dateizugriff in C benutzten Funktionen `ferror()` und `clearerr()` dienen.

Exceptions stellen eine weitere (häufig in objektorientierten Sprachen zu findende) Möglichkeit zur Signalisierung von Fehlern dar. Exceptions können an beliebiger Stelle ausgelöst ("geworfen") werden. Programmblöcke, in denen Exceptions auftreten können und besonders behandelt werden sollen, werden besonders markiert (in C++ etwa als sogenannter `try` Block). Tritt in solch einem Block eine Exception auf, springt die Programmausführung zur Behandlung dieser Exception (dem `catch` Block in C++). In diesem Block kann auf die aufgetretene Exception reagiert werden. Dem `catch` Block können dabei neben der Art der Exception auch weitere Daten übergeben werden. Für eine ausführlichere Diskussion von Exceptions sei auf [C++] verwiesen.

Aufgrund der beschriebenen Vorteile soll zur Fehlermeldung innerhalb eines Graphs auf Exceptions zurückgegriffen werden. Um eine bessere Zusammenarbeit garantieren zu können, wird von den angesprochenen Quellsystemen ebenfalls die Verwendung von Exceptions erwartet. Ist dies nicht der Fall, müssen Exceptions wie in Abschnitt 3.4.3 beschrieben nachgebildet werden.

3.4.2 Werfen von Exceptions

Tritt bei der Abarbeitung einer lokalen Funktion eine Exception auf und wurden innerhalb des Abhängigkeitsgraphen keine weiteren Vorkehrungen getroffen, wird die Ausführung des Abhängigkeitsgraphen abgebrochen. Die aufgetretene Exception wird an den Aufrufer der Zielfunktion weitergegeben. Tritt eine Exception auf, während mehrere Teile eines Abhängigkeitsgraphen parallel ausgeführt werden, wird zunächst versucht, die erforderlichen Abhängigkeiten über andere Wege zu erfüllen. Erst wenn dies nicht mehr möglich ist, wird die Ausführung des Abhängigkeitsgraphen abgebrochen und

die aufgetretene Exception wird an den Aufrufer zurückgemeldet. Sollten während der parallelen Ausführung mehrere Exceptions aufgetreten sein, wird die letzte aufgetretene Exception zurückgemeldet.

Soll nicht die ursprünglich innerhalb einer Quellfunktion aufgetretene Exception an den Aufrufer weitergegeben werden, muß die gewünschte "neue" Exception mit dem Aufruf der Quellfunktion innerhalb des Abhängigkeitsgraphen verknüpft werden. Ebenso kann eine Exception aufgrund einer Vergleichsoperation oder ähnlichem ausgelöst werden, in dem die gewünschte Exception mit der Vergleichsoperation, also einem Funktionsaufruf, verknüpft wird. Sollen etwa bei der Ausführung von `get_color()` nur die Farben rot oder gelb zurückgegeben, ansonsten aber die Exception `EWRONGCOL` ausgelöst werden, so muß der Abhängigkeitsgraph wie in Abbildung 3.26 modelliert werden. Die Verknüpfung des Funktionsaufrufs mit der Exception wird hierbei in der Abbildung durch eine gestrichelte Linie ohne Pfeilspitze symbolisiert.

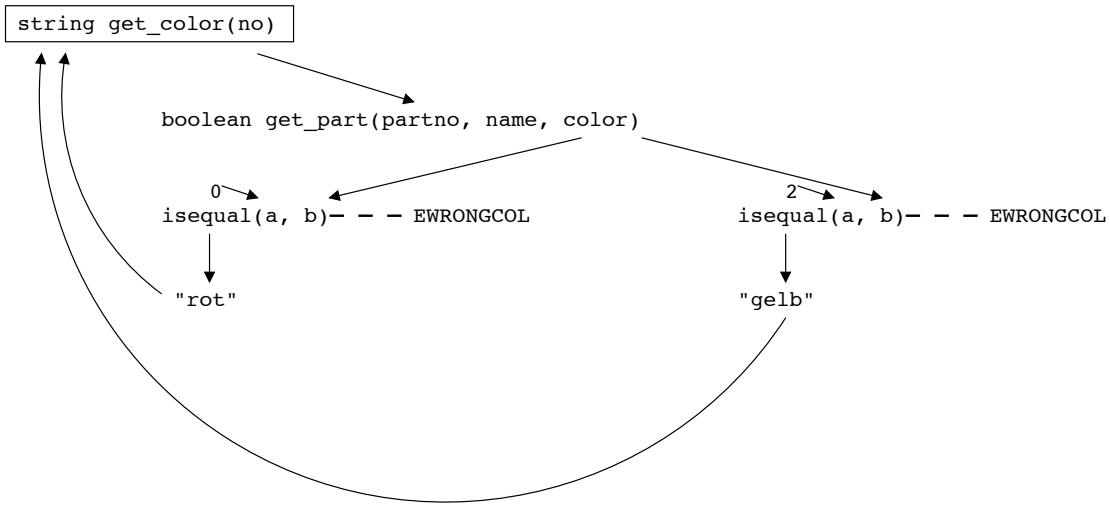


Abbildung 3.26: Auslösen von Exceptions.

3.4.3 Nachbildung von Exceptions

Wie schon in Abschnitt 3.4.1 beschrieben, verwenden nicht alle Systeme Exceptions zur Meldung von Fehlern. Um solche Systeme ebenfalls integrieren zu können, müssen die von dem System verwendeten Methoden zur Signalisierung von Fehlern auf Exceptions abgebildet werden. Da es sich dabei nicht immer um eine einfache Abbildung Fehlercode → Exception handelt, bietet es sich an, diese Abbildung ebenfalls durch einen Abhängigkeitsgraphen darzustellen.

Dabei wird jeder Quellfunktion, für die Exceptions nachgebildet werden sollen, ein solcher Abhängigkeitsgraph, also eine Art Wrapper, zugeordnet. Das System aus Werk 2 des Beispiels verwendet den Rückgabewert einer Funktion zur Signalisierung von Fehlern. Ein Wert von `FALSE`, `NULL` oder `0` bedeutet dabei, daß ein Fehler aufgetreten ist. Wird ein anderer Wert zurückgegeben, wurde die Funktion erfolgreich ausgeführt. Abbildung 3.27 zeigt einen Abhängigkeitsgraphen zur Nachbildung von Exceptions für die Funktion `del_part()`. Existiert ein solcher Graph für eine Quellfunktion, wird anstelle des simplen Aufrufs der Quellfunktion dieser Graph ausgeführt.

Die Funktion `istrue()` (siehe auch Abschnitt 4.13) löst eine Exception aus, wenn ihr Parameter den

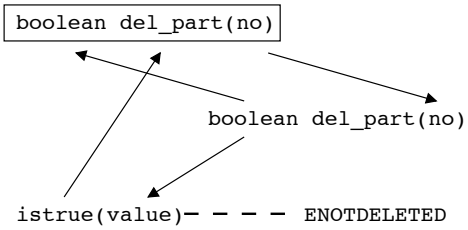


Abbildung 3.27: Nachbildung von Exceptions (I).

Wert `FALSE` besitzt. Infolgedessen wird dann innerhalb des Graphen die der Funktion `istrue()` zugeordnete Exception `ENOTDELETED` ausgelöst. Gibt `del_part()` hingegen `TRUE` zurück, kehrt auch `istrue()` ohne Exception zurück. Damit wird auch die Funktionsabhängigkeit `istrue()` $\rightarrow$ `del_part()` erfüllt und somit alle Abhängigkeiten innerhalb dieses Graphen. Innerhalb eines Fehlergraphen bedeutet dies, daß in der zugehörigen Quellfunktion kein Fehler aufgetreten ist.

Zur Modellierung eines solchen Abhängigkeitsgraphen wird ähnlich zur Modellierung des schreibenden Zugriffs auf ein globales Attribut (Abbildung 3.14, Seite 31) die Quellfunktion, für die der Abhängigkeitsgraph erstellt wird, zweimal innerhalb des Graphen benutzt.

Es sind Fälle denkbar, bei denen innerhalb eines solchen Wrappers nicht nur die Rückgabewerte der Quellfunktion ausgewertet werden, sondern u.U. auf weitere Funktionen oder Attribute des Quellsystems zugegriffen werden muß. Der Beispielgraph aus Abbildung 3.28 für die ANSI C Funktion `fread()` gibt je nach Zustand des Streams die Exceptions `EEOF` oder `EERROR` zurück und verwendet dazu die C Funktionen `clearerr()`, `feof()` und `ferror()` und zusätzlich die Funktion `iszero()` aus der Standardbibliothek. Dieses Beispiel zeigt auch, daß der eigentliche Aufruf der Quellfunktion innerhalb des Graphen nicht unbedingt an erster Stelle erfolgen muß.

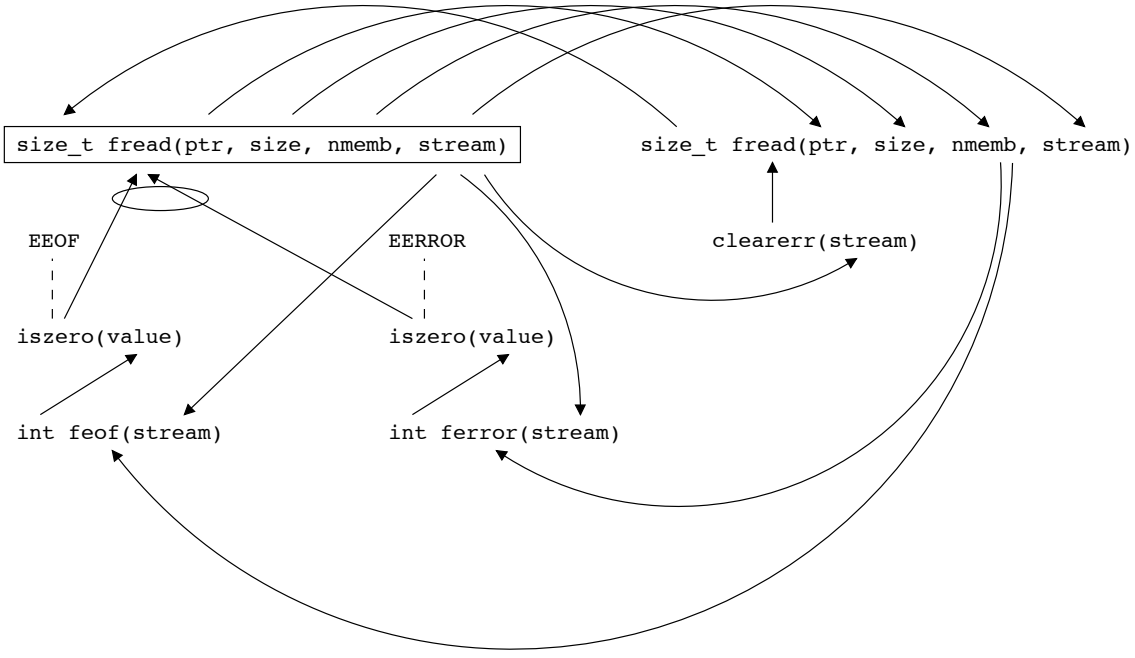


Abbildung 3.28: Nachbildung von Exceptions (II).

3.4.4 Kompensationen

Werden zur Modellierung einer globalen Funktion lokale Funktionen verwendet, welche Nebeneffekte verursachen, reichen die im Abschnitt 3.4.2 beschriebenen Möglichkeiten zur Behandlung von Fehlern nicht aus. Im Fehlerfall müssen vor Beendigung der globalen Funktionen eventuell die an den lokalen Systemen vorgenommenen Zustandsänderungen wieder rückgängig gemacht werden. Welche bzw. wieviele dieser Kompensationsaktionen ausgeführt werden müssen, hängt von der Stelle ab, an der die normale Ausführung durch den Fehler unterbrochen wurde.

Die Realisierung von klassischen Transaktionen ist durch diese Kompensationsaktionen nicht möglich. Möglichkeiten zur Implementierung von globalen Transaktionen werden in Abschnitt 3.5 beschrieben.

Um Kompensationen in Abhängigkeit der bereits ausgeführten lokalen Funktionen aktivieren zu können, müssen diese in den Abhängigkeitsgraphen integriert werden. Innerhalb des Graphen werden Kompensationen ebenfalls durch Ihre Parameter repräsentiert und stellen dadurch Knoten dar. Mit den Knoten der regulären lokalen Funktionsaufrufe werden sie mittels spezieller (Fehler-)Kanten (in den folgenden Abbildungen immer durch gestrichelte Pfeile symbolisiert) verknüpft. Diese Fehlerkanten sind im Rahmen der normalen Ausführung der globalen Funktion inaktiv. Im Fall eines Fehlers werden alle Fehlerkanten aktiv, die von bereits ausgeführten Funktionen sowie der fehlgeschlagenen Funktion selbst ausgehen. Das Laufzeitsystem versucht nun wiederum alle noch bestehenden Abhängigkeiten aufzulösen.

Abbildung 3.29 zeigt die Modellierung der globalen Funktion `add_part()` zunächst beschränkt auf die Verwendung der Funktion `add_part()` von Werk 2 mit der zugehörigen Kompensation. Die in der Darstellung gestrichelte Abhängigkeit `werk2.add_part()` → `werk2.del_part()` bleibt während der normalen Ausführung des Abhängigkeitsgraphs inaktiv. Schlägt die Ausführung von `werk2.add_part()` fehl, kann die Abhängigkeit `partno` → `str` nicht mehr erfüllt werden. Statt dessen wird die Abhängigkeit `werk2.add_part()` → `werk2.del_part()` aktiv und die Ausführung kann über `werk2.del_part()` → `0` und `0` → `long` abgeschlossen werden.

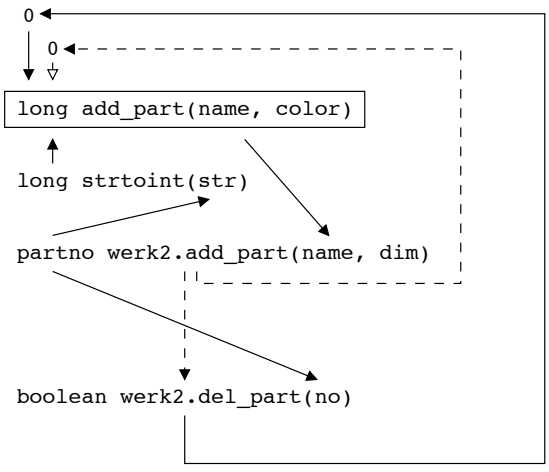


Abbildung 3.29: Kompensation einer Funktion.

Die Abbildung 3.30 zeigt die vollständige Modellierung von `add_part()`. Hier existieren für die beiden `add_part()` Funktion zwei verschiedene Kompensationen. Diese werden unabhängig von-

einander aufgerufen. Tritt z.B. bereits bei der Ausführung von `werk2.add_part()` oder `strtoint()` ein Fehler auf, wird nur die Kompensation `werk2.del_part()` ausgeführt.

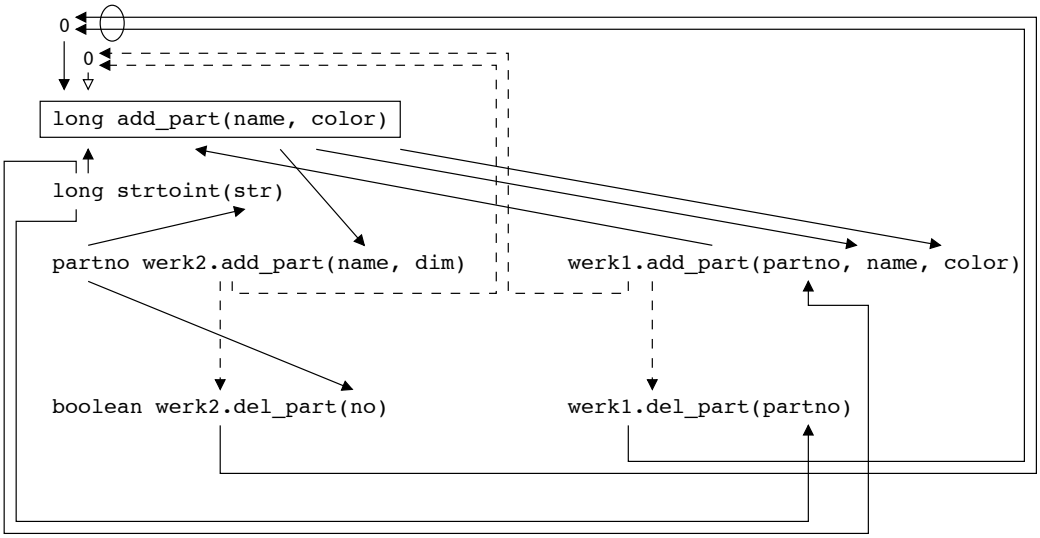


Abbildung 3.30: Kompensation zweier Funktionen.

3.5 Transaktionen

Die Unterstützung von Transaktionen ist eine wesentliche Grundlage für die Realisierung des schreibenden Zugriffs. In diesem Abschnitt sollen die Möglichkeiten und notwendigen Vorraussetzungen zur Realisierung von Zielsystemtransaktionen erläutert werden.

3.5.1 Einführung

Unter einer Transaktion versteht man die Zusammenfassung mehrerer Operationen, die sich wie eine atomare Operation verhält. Eine Transaktion besitzt dabei die folgenden (auch als ACID-Eigenschaften bezeichneten) Eigenschaften [Gray, 1993]:

Atomarität (Atomicity) Die Zustandsänderungen einer Transaktion sind atomar. Es werden entweder alle oder keine der Operationen einer Transaktion ausgeführt (Alles-Oder-Nichts-Prinzip).

Konsistenz (Consistency) Eine Transaktion überführt ein System von einem konsistenten Zustand in einen anderen konsistenten Zustand. Insbesondere verletzt der neue Zustand keine vorgegebenen Integritätsbedingungen.

Isolation (Isolation) Eine noch nicht abgeschlossene Transaktion gibt keine Zwischenergebnisse an andere Transaktionen weiter.

Dauerhaftigkeit (Durability) Wenn eine Transaktion erfolgreich beendet wurde, sind deren Ergebnisse dauerhaft, auch bei nachfolgenden Systemfehlern.

Man spricht hierbei auch von einer flachen Transaktion (flat transaction). Die Realisierung von flachen Transaktionen erfolgt dabei mittels der drei Grundprimitiven `begin()`, `commit()` und `abort()`:

`begin()` markiert den Beginn einer Transaktion. Die Resultate der zwischen `begin()` und dem folgenden `commit()` ausgeführten Operationen werden erst zum Zeitpunkt des `commit()` nach außen sichtbar.

`commit()` macht die erfolgten Operationen nach außen hin sichtbar. Schlägt der Aufruf von `commit()` fehl, werden keine Zustandsänderungen vorgenommen.

`abort()` beendet die Transaktion, ohne Änderungen nach außen sichtbar werden zu lassen.

```
begin();

if (!operation_a())
  abort();
if (!operation_b())
  abort();
/* ... */

commit();
```

Abbildung 3.31: Ablauf einer flachen Transaktion.

3.5.2 Möglichkeiten für Zielsystemtransaktionen

Die beschriebenen flachen Transaktionen behandeln nur den Fall, daß ein System an einer Transaktion beteiligt ist. Während der Ausführung einer Zielfunktion sind aber immer das Zielsystem und ein oder mehrere Quellsysteme beteiligt. Es muß also ein weitergehender Mechanismus gefunden werden.

Geschachtelte Transaktionen

Geschachtelte Transaktionen (nested transactions) bestehen aus einer Haupttransaktion (top-level transaction) und mehreren Subtransaktionen. Sowohl Haupt- als auch Subtransaktionen erfüllen die ACID-Eigenschaften mit der Ausnahme, daß ein `commit()` einer Subtransaktion erst effektiv wird, wenn die Haupttransaktion ebenfalls durch ein `commit()`abgeschlossen wurde. Außerdem veranlaßt ein `abort()` der Haupt- oder einer Subtransaktion den sofortigen Rollback der gesamten geschachtelten Transaktion. Die Subtransaktionen erfüllen also die A, C und I Eigenschaften, nicht aber die D Eigenschaft. Eine weiterführende Behandlung von geschachtelten Transaktionen erfolgt z.B. in [Gray, 1993, Seite 195ff].

Geschachtelte Transaktionen passen sehr gut zu dem verwendeten Modell der Integration. Die globale Funktion bzw. deren Ausführung durch die Abarbeitung des Abhängigkeitsgraphen stellt die Haupttransaktion dar, während die lokalen Funktionsaufrufe den Subtransaktionen entsprechen. Es existieren bisher allerdings sehr wenig Implementierungen von geschachtelten Transaktionen. Aus diesem Grund ist deren Verwendung im Rahmen der Funktionsintegration nicht sinnvoll.

Verteilte Transaktionen

Erfolgt die Ausführung der Operationen einer flachen Transaktion auf mehreren, verteilten Systemen, spricht man von einer verteilten Transaktion. Die beteiligten Systeme werden dabei als Ressourcenmanager bezeichnet. Koordiniert wird die verteilte Transaktion durch einen Transaktionsmanager, welcher auch die Funktionen `begin()`, `commit()` und `abort()` zur Verfügung stellt. Wird eine Funktion eines Ressourcenmanagers im Rahmen einer verteilten Transaktion aufgerufen, registriert sich dieser Ressourcenmanager beim Transaktionsmanager. Soll bei einer verteilten Transaktion ein Commit erfolgen, beginnt das sogenannte Zwei-Phasen Commit-Protokoll (2PC):

Phase 1 (Voting Phase)

1. Der Transaktionsmanager sendet eine "Prepare" Nachricht an die beteiligten Ressourcenmanager
2. Ein Ressourcenmanager sendet eine positive Antwort ("ready-to-commit") zurück, wenn er bereit ist, die Transaktion abzuschließen. Falls der Ressourcenmanager einen Fehler erkannt hat — falls also die Beendigung der Transaktion das System in einem inkonsistenten Zustand zurücklassen würde — sendet er eine negative Antwort zurück.

Phase 2 (Completion Phase)

3. Sind alle eingegangen Antworten positiv, wird die Transaktion abgeschlossen. Der Transaktionsmanager schickt eine "Commit" Nachricht an alle beteiligten Ressourcenmanager. Ist mindestens eine Antwort negativ, sendet der Transaktionsmanager eine "Abort" Nachricht an alle beteiligten Ressourcenmanager.
4. Die Systeme führen entsprechend ein "Commit" oder ein "Abort" aus.

Die Realisierung von globalen Transaktionen ist unter Verwendung von verteilten Transaktionen möglich. Mit der von der Open Group entwickelten XA Spezifikation [XA] steht ein Standard für verteilte Transaktionen zur Verfügung, welcher von vielen Implementierungen unterstützt wird.

3.5.3 Voraussetzungen für Zielsystemtransaktionen

Bedingt durch die Heterogenität der zu integrierenden Systeme sind auch deren Fähigkeiten bezüglich Transaktionen sehr unterschiedlich. Diese Fähigkeiten entscheiden darüber, ob eine Realisierung von Zielsystemtransaktionen (also verteilten Transaktionen) überhaupt möglich ist und wenn ja, welcher Aufwand dazu betrieben werden muß. Bezüglich der Fähigkeiten der Quellsysteme kann folgende Einteilung unternommen werden:

Quellsystem unterstützt keine Transaktionen

Ein Quellsystem, welches keinerlei transaktionale Fähigkeiten bietet, kann nicht zur Integration innerhalb eines transaktionsbasierten Zielsystems verwendet werden. Falls eine Änderung des Systems

auf Quelltextebene nicht möglich ist, bleibt nur die Alternative, einen Wrapper als einen Aufsatz auf Basis von Kompensationen für dieses System zu erstellen. Durch Kompensationen kann für solche Quellsysteme allerdings keine vollständige Transaktionssemantik für die globalen Systeme verwirklicht werden, da die Zwischenergebnisse der Ausführung bereits vor der Beendigung der Transaktion nach außen, d.h. für andere, gleichzeitig laufende Prozesse sichtbar werden. Dadurch wird die Isolationseigenschaft der ACID-Kriterien verletzt.

Quellsystem unterstützt keine Transaktionen, aber Quellfunktionen sind in sich transaktional

Ein solches Quellsystem stellt keine Transaktionen mittels der Funktionen `begin()`, `commit()` und `abort()` zur Verfügung. Statt dessen erfolgt die Ausführung der einzelnen Quellfunktionen transaktional, also nach dem Alles-Oder-Nichts-Prinzip. Trotz dieser Eigenschaft ist eine Einbindung solcher Systeme in verteilte Transaktionen nicht möglich, da für diese Systeme kein `abort()` möglich ist. Es bieten sich prinzipiell die gleichen Lösungsmöglichkeiten wie für Systeme, die keine Transaktionen unterstützen, an.

Quellsystem unterstützt Locking

Unterstützt ein Quellsystem eine Form von Locking, d.h. die Möglichkeit für einzelne Prozesse auf bestimmte (im einfachsten Fall alle) Ressourcen des Systems exklusiv zugreifen zu können, ist es möglich, darauf aufbauend mittels Kompensationen eine komplette Transaktionssemantik nachzubilden.

Quellsystem unterstützt flache Transaktionen

Soll nur ein lokales System integriert werden, so kann dies auf Basis von flachen Transaktion erfolgen. Für die Integration mehrerer lokaler Systeme ist dies aber nicht ausreichend. Da es nicht möglich ist, aufbauend auf den `commit()` bzw. `abort()` Primitiven einen Ressourcenmanager zu erstellen, der das 2PC unterstützt, ist die Einbindung von Systemen, welche nur flache Transaktionen unterstützen, in eine verteilte Transaktion ohne Änderungen an deren Quelltext nicht möglich. Da diese Systeme aber bereits flache Transaktionen unterstützen, sollten die notwendigen Änderungen am Quelltext relativ gering ausfallen.

Weitere Möglichkeiten der Einbindung bestehen über die Nachbildung von Transaktionen per Locking oder über die bereits beschriebenen Kompensationen mit den ebenfalls bereits beschriebenen Nachteilen.

Quellsystem unterstützt verteilte Transaktionen

Unterstützen alle beteiligten Quellsysteme verteilte Transaktionen, ist die Realisierung von Zielsystemtransaktionen ohne Einschränkungen möglich. Das entstehende Zielsystem muß dabei ebenfalls als Ressourcenmanager innerhalb der verteilten Transaktion agieren können.

3.5.4 Realisierung

Um innerhalb eines Zielsystems Transaktionen zur Verfügung stellen zu können, müssen die zu integrierenden Quellsysteme ebenso wie das Zielsystem selbst innerhalb einer verteilten Transaktion zusammengefaßt werden können. Das bedeutet, daß alle beteiligten Quellsysteme als Ressourcenmanager agieren und das 2PC unterstützen müssen.

Beim Aufruf einer Zielfunktion registriert sich der zu dem Zielsystem gehörende Ressourcenmanager bei dem globalen Transaktionsmanager. Innerhalb der Ausführung der Zielfunktion registrieren sich die Ressourcenmanager sämtlicher benutzter Quellsysteme ebenfalls beim Transaktionsmanager. Kommt es zum Commit, nehmen alle beteiligten Quellsysteme und das Zielsystem am 2PC teil.

Das Zielsystem muß sich ebenfalls beim Transaktionsmanager als Ressourcenmanager registrieren, damit es im Falle eines Rollbacks durch einen anderen Ressourcenmanager (also ein Quellsystem) sofort eine Benachrichtigung darüber erhält und die Abarbeitung des Abhängigkeitsgraphen abbrechen kann.

Die Modellierung der Abhängigkeitsgraphen für ein Zielsystem, welches Transaktionen unterstützt, unterscheidet sich nicht von der bisher beschriebenen Modellierung für Zielsysteme ohne Transaktionen. Die Benutzung eines transaktionsunterstützenden Zielsystems durch Client-Programme unterscheidet sich ebenfalls nicht von der normalen Benutzung eines Systems innerhalb einer verteilten Transaktion. Soll das Vorhandensein verteilter Transaktionen vor den Clients verborgen werden, so muß das Zielsystem selbst die Funktionen `begin()`, `commit()` und `abort()` zur Verfügung stellen. Diese Funktionen müssen dann entsprechend mit dem Transaktionsmanager — welcher sich seinerseits wie ein Quellsystem innerhalb der zugehörigen Abhängigkeitsgraphen verhält — interagieren.

3.6 Aktualisierungsprobleme

DBMSe bieten sogenannte Views zur Definition von Sichten auf bestimmte Tabellen. Während die Abbildung der originalen Tabellen auf die Sicht ohne Probleme möglich ist, bereitet die umgekehrte Richtung u.U. Schwierigkeiten, falls Änderungsoperationen auf der Sicht nicht eindeutig Änderungsoperationen auf den zugrundeliegenden Tabellen zugeordnet werden können. Man spricht dabei von einem Aktualisierungsproblem in Sichten (view-update problem).

View-update Probleme treten innerhalb relationaler DBMSe z.B. bei der Projektion von Schlüsselattributen oder bei der Projektion des Verbundattributs bei einer Join-Operation auf. Außerdem können view-update Probleme auftreten, falls Attribute in der Sicht durch Berechnung erzeugt wurden. Im Rahmen der Funktionsintegration können ähnliche Probleme auftreten. Im Gegensatz zu den DBMSen, die die Abbildung von Sichten auf die zugrundeliegenden Tabellen automatisch aufgrund allgemeiner Regeln vornehmen müssen (und aus diesem Grund u.U. Schreiboperationen auf bestimmte Sichten nicht erlauben), können Aktualisierungsprobleme innerhalb der Funktionsintegration durch spezielle Abhängigkeitsgraphen für jeden Fall einzeln gelöst werden.

Fehlende Attribute

Werden innerhalb der globalen Funktionen nicht alle Argumente der integrierten lokalen Funktionen verwendet, kommt es bei der Aktualisierung bestehender Datensätze zu dem Problem, daß nicht alle

notwendigen Daten von den globalen Funktionen zur Verfügung gestellt werden. Es müssen also durch eine vorangestellte Leseoperation die restlichen Daten aus dem bisherigen Datensatz ausgelesen werden. Abbildung 3.32 zeigt dies anhand der globalen Funktion `upd_part()`.

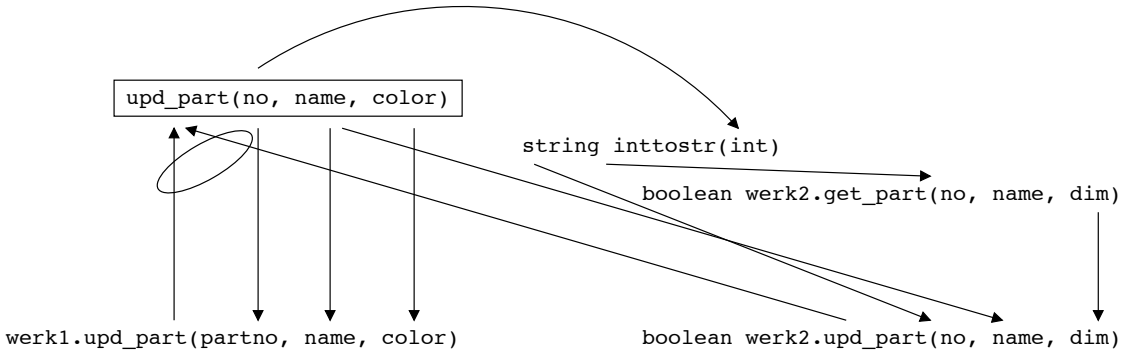


Abbildung 3.32: Fehlende Werte.

Informationsverlust

Ein Informationsverlust tritt auf, wenn keine bijektive Abbildung zwischen den entsprechenden Parametern der Quell- und Zielfunktionen besteht. Dies kann z.B. auftreten, falls zur Darstellung eines Werts in der Quellfunktion ein Fließkommadatentyp verwendet wird, während der Wert innerhalb der Zielfunktion durch eine Integerdatentyp dargestellt wird. Außerdem können Aktualisierungsprobleme durch Informationsverlust bei berechneten Werten innerhalb der Zielfunktion auftreten.

Abbildung 3.33 illustriert das Problem anhand der globalen Funktion `set_price_euro()`. Diese verwendet als Datentyp für den Preis einen Integer, während die Funktion `set_price()` des Lieferantensystems einen Festkommadatentyp benutzt. Innerhalb eines Abhängigkeitsgraphen für `set_price_euro()` muß also zunächst geprüft werden, ob sich der neue Wert von dem gerundeten bisherigen Wert unterscheidet. Nur dann wird der Wert durch die Funktion `set_price()` in dem lokalen System aktualisiert. So wird vermieden, daß u.U. ein Festkommawert unnötigerweise durch dessen gerundete Variante ersetzt wird.

3.7 Verschiedenes

In diesem Abschnitt sollen weitergehende Aspekte bzgl. einer Standardbibliothek und der Verwendung von semantischen Attributen besprochen werden.

3.7.1 Standardbibliothek

Einige Typen von Funktionen werden zur Funktionsintegration recht häufig benutzt. Es handelt sich dabei, neben den oft benötigten Funktionen zur Konvertierung zwischen den verschiedenen Datentypen, auch um die für bedingte Anweisungen bzw. Schleifen benötigten Bedingungsfunktionen wie beispielsweise `isequal()`.

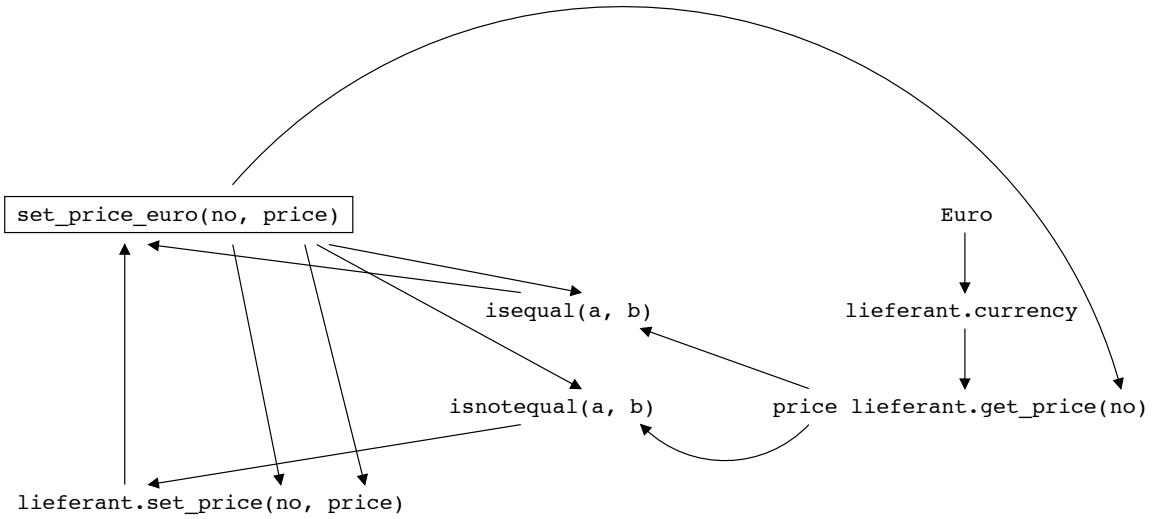


Abbildung 3.33: Informationsverlust.

Da diese Funktionen zumeist nicht Teil eines lokalen Systems sind, müssen sie durch ein Hilfssystem bereitgestellt werden. Um zu vermeiden, daß diese Funktionen bei jeder Integration neu und höchstwahrscheinlich inkompatibel erstellt werden, sollte eine Art Standardbibliothek definiert werden. Dies verringert den Aufwand der Integration und verbessert die Portabilität und Austauschbarkeit der erstellten Graphen. Folgende Kategorien von Funktionen sollten in einer Standardbibliothek vorhanden sein[5]:

Bedingungsfunktionen: Bedingungsfunktionen werden sehr häufig benötigt, da sie sowohl zur Realisierung von bedingten Anweisungen als auch von Schleifen benötigt werden. Sie müssen das in Abschnitt 3.3.1 beschriebene Verhalten aufweisen. Neben Funktionen in der Art des bereits in den Beispielen benutzen `isequal()` sollte eine allgemein Bedingungsfunktion enthalten sein, welcher die gewünschte Bedingung als Argument übergeben werden kann.

Konvertierungsfunktionen: Da das verwendete Modell keinen Mechanismus zur Konvertierung von Datentypen vorsieht, sondern Hilfsfunktionen dazu benutzt werden, sollten innerhalb einer Standardbibliothek grundlegende Konvertierungsfunktionen wie z.B. `strtoint()` definiert sein.

Strings / Listen: Im Rahmen der Funktionsintegration werden Operationen wie beispielsweise die Konkatenation von Strings sehr häufig benötigt. Ebenfalls ist oft ein Zugriff auf Arrays oder Listen erforderlich. Aus diesem Grund sollte in der Standardbibliothek ein vollständiger Satz an Funktionen zur Bearbeitung von Strings und Listen vorhanden sein.

Gängige Berechnungen: Die Aufnahme von Funktionen in der Art von `incr(value)` trägt ebenfalls zu einer Verringerung der Komplexität der entstehenden Abhängigkeitsgraphen und zur besseren Portabilität bei.

[5] Abschnitt 4.13 enthält eine ausführlichere Beschreibung der hier erwähnten Funktionen.

3.7.2 Semantische Attribute

Semantische Attribute dienen dazu, sowohl lokale als auch globale Funktionen oder Attribute so zu kategorisieren, daß ein graphisches Werkzeug Abhängigkeitsgraphen effizient darstellen und den Benutzer bei der Integration unterstützen kann.

In der bisherigen Beschreibung der Funktionsintegration durch Abhängigkeitsgraphen wurde bewußt darauf geachtet, möglichst wenig verschiedene Konstrukte zu verwenden. So wurden Konvertierungen durch zusätzliche Hilfsfunktionen realisiert. Ebenso wurden die Bedingungen von bedingten Anweisungen und Schleifen als Funktionen realisiert.

Dadurch ist es möglich, daß eine entsprechende Beschreibungssprache relativ einfach gehalten werden kann. Dies führt aber u.U. auch zu sehr unübersichtlichen Abhängigkeitsgraphen mit einer großen Anzahl von Konvertierungs- bzw. Bedingungsfunktionen im Vergleich zu den verwendeten Quellsystemfunktionen. Werden Konvertierungs- und Bedingungsfunktionen als solche markiert, kann ein graphisches Werkzeug diese entsprechend darstellen oder — im Fall von Konvertierungsfunktionen — sie in bestimmten Ansichten ausblenden.

Abbildung 3.34 illustriert eine mögliche Art der Darstellung von Konvertierungsfunktionen anhand des Beispiels von Abbildung 3.8 auf Seite 26. Ein Beispiel für die mögliche Darstellung von Bedingungsfunktionen enthält die Abbildung 3.17 auf Seite 34.

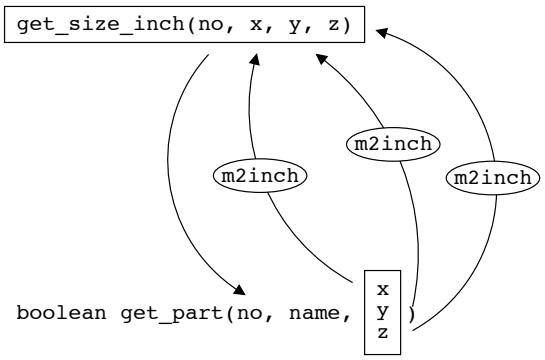


Abbildung 3.34: Darstellung von Konvertierungsfunktionen.

Eine weitere Möglichkeit der Kategorisierung bietet sich durch die Vergabe von semantischen Attributen, welche den Zweck einer Funktion genauer beschreiben. Beispiele dafür sind `read`, `add`, `update` oder `delete`. Sind die Funktionen der zu integrierenden Quellsysteme derart gekennzeichnet, können Werkzeuge den Benutzer bei der Erstellung der Abhängigkeitsgraphen unterstützen, indem diese zu der behandelten globalen Funktion "passende" lokale Funktionen heraussucht. Passend bedeutet in diesem Zusammenhang, daß die Datentypen oder Namen von Parametern der lokalen Funktionen ganz oder teilweise mit denen der globalen Funktion übereinstimmen, und daß die semantischen Attribute der Funktionen übereinstimmen. Soll beispielsweise ein Abhängigkeitsgraph für die globale Funktion

```
void get_part(in long no, out string name, out string color,
    out price_t price);
```

(gekennzeichnet durch das semantische Attribut `read`) erstellt werden, so könnte ein graphisches Werkzeug die "passenden" Funktionen

```
void get_part(in partno_t partno, out string name, out color_t color);
price get_price(in long no);
```

des Systems von Werk 1 bzw. des Lieferanten heraussuchen. Zusätzlich würde auch die Konvertierungsfunktion

```
string col2str(in color_t color);
```

angezeigt werden.

Die Kategorisierung durch semantische Attribute ist nicht auf die hier vorgeschlagenen Werte `read`, `add`, `update` und `delete` beschränkt, sondern beliebig erweiterbar. Semantische Attribute können nur als Hilfestellung für den Benutzer bei der Erstellung eines Abhängigkeitsgraphs dienen. Eine vollständige Automatisierung durch semantische Attribute ist dabei nicht möglich.

Kapitel 4

Die Beschreibungssprache

In diesem Kapitel wird die Beschreibungssprache FIX (Function Integration in XML) vorgestellt, eine Sprache zur Modellierung der in Kapitel 3 behandelten Konzepte. Neben einer ausführlichen Syntaxbeschreibung enthält dieses Kapitel Beispiele zur Lösung der im vorangegangenen Kapitel beschriebenen Probleme.

4.1 Anforderungen

In Kapitel 3 erfolgte die Beschreibung der Abhängigkeiten durch Graphen. Dabei wurden die Funktionen durch ihre Signatur dargestellt, Abhängigkeiten wurden durch Pfeile symbolisiert. Spezialfälle, wie z.B. der zyklische Fall, wurden durch weitere Symbole — in diesem Fall der Doppelpfeil — dargestellt. Diese visuelle Darstellung ist als formale und vor allem Computer-lesbare Beschreibung nicht geeignet. Hierfür bietet sich eine (Beschreibungs-) Sprache an.

Neben der Beschreibung der Abhängigkeiten muß mit der gewählten Beschreibungssprache die Definition der von den Systemen zur Verfügung gestellten Schnittstellen möglich sein. Des weiteren soll die Sprache auch zur Dokumentation der beschriebenen Systeme benutzt werden können.

Die Schnittstellenbeschreibung enthält neben der Definition der zur Verfügung gestellten Funktionen zusätzlich die Definition der verwendeten Datentypen und Konstanten. Stellt ein System mehrere Schnittstellen zur Verfügung, oder sind die zur Verfügung gestellten Schnittstellen auf eine bestimmte Art und Weise (z.B. hierarchisch) strukturiert, müssen diese Strukturen ebenfalls in der Schnittstellenbeschreibung wiedergegeben werden. Benutzt ein zu beschreibendes System einen objektorientierten Ansatz, müssen dessen zusätzlichen Merkmale (etwa Vererbungsbeziehungen der exportierten Schnittstellen, etc.) ebenfalls wiedergegeben werden können.

Es existieren mehrere Sprachen zur Schnittstellenbeschreibung. Von der im Rahmen des Distributed Computing Environment [DCE] entwickelten Interface Definition Language (IDL), wurden inzwischen weitere Varianten, wie z.B. Microsoft COM IDL sowie die von Corba verwendete OMG IDL [CORBA] entwickelt. IDL benutzt eine an C++ angelehnte Syntax zur Beschreibung der Schnittstellen. Eine weitere Sprache zur Definition von Schnittstellen ist die an C angelehnte eXternal Data Representation Sprache [XDR], welche im Zusammenhang mit ONC RPC [ONCRPC] benutzt wird. Wegen der geplanten Verwendung von CORBA wurde versucht, den Interfacebeschreibungsteil der Sprache in Umfang und Mächtigkeit möglichst nahe an OMG IDL anzulehnen.

Die Systemdokumentation umfaßt mehrere Teilaspekte. Zum einen müssen die zur Benutzung der exportierten Schnittstellen notwendigen technischen Daten definiert werden. Dazu gehören neben dem Namen bzw. der Netzwerkadresse des Systems das verwendete Kommunikationsprotokoll wie etwa ONC RPC oder IIOP und u.U. weitere protokollspezifische Details, z.B. die Port Nummer bei einem auf TCP/IP basierenden Protokoll. Des weiteren sollen die im Schnittstellenbeschreibungsteil vorgenommenen Funktions- bzw. Typdefinitionen dokumentiert werden können. Ebenso soll es möglich sein, Anmerkungen zu den Abhängigkeitsbeschreibungen hinzuzufügen.

4.2 Die Sprache FIX

Als Grundlage der Beschreibungssprache wurde XML [XML] gewählt. XML bietet vor allem den Vorteil, daß, bedingt durch die Standardisierung und weite Verbreitung, eine große Anzahl an Software zur Verarbeitung von XML Dokumenten existiert. Neben den für viele Systeme bzw. Programmiersprachen vorhandenen Parsern existieren z.B. Editoren oder Tools zur Umwandlung von XML Dokumenten in andere Formate (siehe Abschnitt 4.14).

FIX ist eine Anwendung des XML 1.0 Standards. Dies bedeutet insbesondere, daß jedes FIX Dokument wohlgeformt sein muß. FIX Dokumente sollten gültig gegenüber der in Anhang A aufgeführten DTD[1], bzw. bei Verwendung eines auf Schemas [SCHEMA1] basierenden Parsers schemagültig gegenüber dem in Anhang B aufgeführten XML Schemas sein.

Es wird vorgeschlagen, für FIX Dokumente einen Media-Type bei der Internet Assigned Numbers Authority (IANA) zu reservieren. In Übereinstimmung mit [Murata, 2000] wird dafür der Name `application/fix+xml` vorgeschlagen.

4.3 Namensräume

FIX definiert den Namensraum [NAMESPACE] `http://dcx.com/fix`. Dieser dient u.a. dazu, die in FIX definierten Werte des `semantic` Attributs (Abschnitt 4.11) von denen anderer Namensräume zu unterscheiden. Außerdem wird dadurch beispielsweise die schemagültige Einbettung von FIX Dokumenten in andere XML Dokumente ermöglicht. Weiterhin werden in FIX Dokumenten Attribute aus dem XLink Namensraum verwendet. Um die Validierung von FIX Dokumenten gegenüber der verwendeten DTD zu ermöglichen, muß die Deklaration der Namensräume immer in der in Abschnitt 4.8 dargestellten Weise erfolgen.

Sollen Elemente anderer Namensräume innerhalb der `<desc>` oder `<expr>` Elemente oder zusätzliche Werte für das `semantic` Attribut verwendet werden (siehe auch Abschnitt 4.11), ist die Deklaration zusätzlicher Namensräume notwendig. Während dies im ersten Fall innerhalb des ersten Elements des zusätzlich verwendeten Namensraums erfolgen sollte, damit keine Probleme im Zusammenhang mit der Gültigkeit des Dokuments auftreten können, ist dies im zweiten Fall nicht auf eine Art und Weise möglich, welche die Gültigkeit gegenüber dem hier verwendeten DTD erhält. Aus diesem Grund muß das FIX Dokument eine interne DTD Teilmenge mit einer zusätzlichen `ATTLIST` Deklaration enthalten (siehe Abbildung 4.1).

[1] Mit der Ausnahme zusätzlicher Namensraumdeklarationen für `semantic` Attribute (siehe Abschnitt 4.3).

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE system SYSTEM "http://dcx.com/fix/fix.dtd" [
  <!ATTLIST system
            xmlns:myns CDATA #FIXED "http://example.com/myns">
]>

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:myns="http://example.com/myns">

  ...

  <op semantic="myns:special">
    ...
  </op>
```

Abbildung 4.1: Definition zusätzlicher Namensräume.

4.4 XLink

FIX verwendet die XML Linking Language (XLink) [XLINK] zur Verknüpfung von verschiedenen Ressourcen. XLink definiert Attribute, mit denen Verknüpfungen zwischen XML Dokumenten realisiert werden können. Durch die Verwendung eines eigenen Namensraumes können die durch XLink spezifizierten Attribute beliebigen Elementen zugewiesen werden.

XLink unterscheidet zwischen zwei Arten von Verknüpfungen. Simple Links dienen der Verknüpfung einer lokalen und einer entfernten Ressource. Die Verknüpfungsrichtung verläuft dabei immer von der lokalen zu der entfernten Ressource. Simple Links ähneln damit dem aus HTML [HTML] bekannten `<a>` Element. In dem XML Fragment in Abbildung 4.2 wird z.B. die lokale Ressource “Beispiel” mit dem Dokument `verknuepft.html` verknüpft.

```
<a xmlns:xlink="http://www.w3.org/1999/xlink"
 xlink:type="simple" xlink:href="verknuepft.html">Beispiel</a>
```

Abbildung 4.2: XLink Simple Link.

Mit Extended Links ist die Verknüpfung von mehreren, sowohl entfernten als auch lokalen Ressourcen möglich. Somit können auch sogenannte out-of-line Links, also Verknüpfungen, an denen nur entfernte Ressourcen beteiligt sind, realisiert werden. Extended Links erlauben weiterhin die Angabe, in welcher Richtung die Verknüpfung erfolgen soll. Die Abbildung 4.3 zeigt einen Extended Link, der drei Ressourcen (eine davon lokal) miteinander verknüpft, wobei nur eine Verknüpfung von der lokalen Ressource bzw. `a.html` in Richtung `b.html` erfolgt, und nicht umgekehrt.

Zum Verweis auf entfernte Ressourcen benutzt XLink die auf der XML Path Language (XPath) [XPATH] basierende XML Pointer Language (XPointer) [XPOINTER]. XPointer ermöglichen es, nicht nur komplette XML Dokumente zu referenzieren, sondern auf einzelne Teile eines XML Dokuments (bis hin zu einzelnen Zeichenfolgen) zu verweisen. So zeigt das Element `<see>` aus Abbil-

```
<ext xmlns:xlink="http://www.w3.org/1999/xlink xlink:type="extended">
  <remote xlink:type="locator" xlink:role="a" xlink:href="a.html"/>
  <remote xlink:type="locator" xlink:role="b" xlink:href="b.html"/>
  <local xlink:type="resource" xlink:role="c">
    lokal
  </local>
  <link xlink:type="arc" xlink:from="c" xlink:to="b"/>
  <link xlink:type="arc" xlink:from="a" xlink:to="b"/>
</ext>
```

Abbildung 4.3: XLink Extended Link.

dung 4.4 auf das erste <para> Element, dessen Vater-Element die ID intro trägt. Neben der normalen XPointer Syntax verwendet FIX zur Referenzierung von Teilen von Parametern (z.B. Elemente einer Struktur) eine erweiterte Syntax (siehe Abschnitt 4.12).

```
<see xmlns:xlink="http://www.w3.org/1999/xlink"
 xlink:type="simple"
 xlink:href="http://example.com/document.xml#xpointer(id("intro")/child::
para[1]"/>
```

Abbildung 4.4: XPointer.

Innerhalb von FIX werden XLink Simple Links beispielsweise innerhalb der <type> und <raises> Elemente benutzt, um Datentypen bzw. Exceptions zu referenzieren, welche an anderer Stelle definiert wurden. Simple Links werden ebenfalls zur Modellierung von Vererbungsbeziehungen benutzt. Die Abhängigkeitsgraphen werden in FIX durch XLink Extended Links beschrieben.

4.5 Parameter Entities

Parameter Entities sind Konstrukte zum Textersatz innerhalb von DTDs ähnlich den #define Anweisungen des C Präprozessors. Der in Abbildung 4.5 dargestellte Ausschnitt aus einer DTD definiert ein Parameter Entity:

```
<!ENTITY  % id
          "id ID #IMPLIED">
```

Abbildung 4.5: Parameter Entity.

Nach dieser Definition wird jedes Vorkommen des Ausdrucks %id; innerhalb der DTD durch den Ausdruck id ID #IMPLIED ersetzt.

In der verwendeten DTD wurde häufig von Parameter Entities Gebrauch gemacht. Um Wiederholungen zu vermeiden, und die DTD kleiner und dadurch übersichtlicher zu gestalten, wurden oft

verwendete Element-Inhalte bzw. Attribute durch Parameter Entities ersetzt. Des weiteren wurden Datentypen und — in Anlehnung an die IDL Definition — eine Reihe von Typklassen als Parameter Entities definiert.

DTDs bieten nur sehr eingeschränkte Möglichkeiten zur Definition von Datentypen. Aus diesem Grund wurden innerhalb der hier benutzten DTD einige Entities zur Beschreibung der verschiedenen Datentypen definiert. Um eine einfache Umstellung auf XML Schemas zu ermöglichen, wurden, soweit möglich, die Datentypen aus [SCHEMA2] benutzt (Abbildung 4.6).

```
<!ENTITY  % string              "CDATA">
<!ENTITY  % boolean             "(true | 1 | false | 0)">
<!ENTITY  % uri                 "CDATA">
<!ENTITY  % language            "NMTOKEN">
<!ENTITY  % Name                "NMTOKEN">
<!ENTITY  % QName               "NMTOKEN">
<!ENTITY  % NCName              "NMTOKEN">
<!ENTITY  % integer             "CDATA">
<!ENTITY  % non-negative-integer "CDATA">
<!ENTITY  % positive-integer    "CDATA">
<!ENTITY  % contenttype         "CDATA">
<!ENTITY  % semantic            "NMTOKEN">
<!ENTITY  % coords              "CDATA">
```

Abbildung 4.6: Parameter Entities für Datentypen.

Des weiteren wurden die in Abbildung 4.7 dargestellten Parameter Entities zur Beschreibung bestimmter Typklassen definiert. Diese wurden analog zu den angegebenen Produktionen der IDL Grammatik [CORBA, Abschnitt 3.4] definiert.

4.6 Aufbau

Neben den an verschiedenen Stellen benutzten allgemeinen Elementen sind FIX Dokumente in drei Hauptteile gegliedert. Nach der Systembeschreibung, zu der z.B. die Adresse und das verwendete Kommunikationssystem, aber auch weitergehende Dokumentation gehört, folgt die Definition der exportierten Schnittstellen. Diese orientiert sich sehr stark an den Fähigkeiten von OMG IDL, es wird deshalb des öfteren auf die entsprechenden Stellen innerhalb von [CORBA] verwiesen. An die Definition der Schnittstellen schließt sich optional die Abhängigkeitsbeschreibung an.

In den folgenden Abschnitten erfolgt die Beschreibung der einzelnen Elemente durch einen Auszug aus der DTD und einer Beschreibung der Verwendung des Elements und seiner Attribute. Obwohl die Verwendung von Schemas eine genauere Beschreibung der verwendeten Elemente erlaubt, wurde auf DTD Ausschnitte zurückgegriffen, da sie eine kompaktere und übersichtlichere Darstellung erlauben.

Anhand von Beispielen wird die Verwendung der beschriebenen Elemente verdeutlicht. Die Beispiele orientieren sich zumeist an den in Kapitel 3 benutzen Abhängigkeitsgraphen. Anhang E enthält die kompletten FIX Dokumente zu den verwendeten Beispielen. In den Beispielen werden die Werte des `id` Attributs zumeist gleich dem Namen des zugehörigen Elements gewählt, dies ist aber nicht zwingend notwendig (siehe dazu auch Abschnitt 4.9.1). Einige der Beispiele enthalten zum besseren Verständnis eine Gegenüberstellung von entsprechenden Konstrukten in FIX und IDL.

```
<!-- IDL Grammatik Produktion 46 -->
<!ENTITY  % base                "(float | int | char | bool | octet |
                                 any | object | valuebase)">

<!-- IDL Grammatik Produktion 47 -->
<!ENTITY  % template            "(sequence | string | fixed)">

<!-- IDL Grammatik Produktion 45 -->
<!ENTITY  % simple              "(%base; | %template; | type)">

<!-- IDL Grammatik Produktion 48 -->
<!ENTITY  % constr              "(struct | union | enum)">

<!-- IDL Grammatik Produktion 44 -->
<!ENTITY  % datatype            "(%simple; | %constr;)">

<!-- IDL Grammatik Produktion 95 -->
<!ENTITY  % param               "(%base; | string | type)">

<!-- IDL Grammatik Produktion 42 -->
<!ENTITY  % type_dcl            "(typedef | struct | union | enum |
                                 native)">

<!-- IDL Grammatik Produktion 2 -->
<!ENTITY  % definition          "(%type_dcl; | const | except |
                                 interface | module | valuetype)">

<!-- IDL Grammatik Produktion 9 -->
<!ENTITY  % export              "(%type_dcl; | const | except | attr |
                                 op)">
```

Abbildung 4.7: Parameter Entities für Typklassen.

Nach der Beschreibung der in FIX verwendeten Elemente folgen Abschnitte über semantische Attribute, die erweiterte FIXPointer Syntax sowie die Beschreibung von Konvertierungsmöglichkeiten in andere Formate durch XSL Stylesheets.

4.7 Allgemeine Elemente

Die Elemente `<title>` und `<desc>` werden an mehreren Stellen einer FIX Datei verwendet. Sie werden dazu benutzt, das umgebende Vaterelement näher zu beschreiben. Innerhalb des `<desc>` Elements können durch das `<a>` Element Verknüpfungen zu anderen Dokumenten erfolgen.

`<title>`

Das `<title>` Element dient zur Festlegung des Namens des Vaterelements. Er wird zur Darstellung des jeweiligen Objekts am Bildschirm benutzt. Außerdem wird der Inhalt des `<title>` Elements an einigen Stellen bei der Konvertierung in eine IDL Datei benutzt. Aus diesem Grund gelten für die benutzten Namen die IDL Regeln zu “Scoped Names” [CORBA, Abschnitt 3.2.3 und 3.15].

Innerhalb des `<system>` Elements kann der Name frei gewählt werden, es muß sich *nicht* um den innerhalb des `<address>` Elements angegebenen Netzwerknamen handeln.

```
<!ELEMENT title                 (#PCDATA)>
<!ATTLIST title
          xlink:type            (title) #FIXED "title">
```

`xlink:type` ist festgesetzt auf den Wert `title`, um dieses Element als XLink Title Element zu markieren. Dies ist notwendig, damit XLink kompatible Software die innerhalb von `<graph>`, `<node>` oder `<dep>` Elementen auftretenden `<title>` Elemente als Linktitel interpretieren kann. Kommt das `<title>` Element an anderen Stellen vor, wird dieses Attribut ignoriert.

`<desc>`

Das `<desc>` Element wird zur Dokumentation des umgebenden Vaterelements benutzt. Neben reinem Text kann es Verknüpfungen mit anderen Dokumenten (mittels `<a>` Elementen) beinhalten. Es ist immer das Auftreten mehrerer `<desc>` Elemente hintereinander möglich. Dies kann zur mehrsprachigen Dokumentation mittels verschiedener `xml:lang` Attribute benutzt werden.

Zusätzlich ist es möglich, daß der innerhalb des `<desc>` Elements auftretende Text durch Elemente aus anderen Namensräumen ausgezeichnet wird (siehe Abbildung 4.8). Dies ist zwar durch das Schema aber nicht durch die DTD darstellbar[2].

```
<!ELEMENT desc                  (#PCDATA | a)*>
<!ATTLIST desc
          xml:lang              %language; #IMPLIED>
```

`xml:lang` dient zur Identifikation der innerhalb des `<desc>` Elements verwendeten Sprache (siehe [XML, Abschnitt 2.12]).

[2] Ein Elementinhalt von `ANY` erlaubt nur in der DTD tatsächlich definierte Elemente.

<a>

Das <a> Element dient der Verknüpfung mit anderen Ressourcen. Die Verwendung erfolgt dabei analog zu [HTML]. Das <a> Element wurde als XLink Simple Link realisiert.

```
<!ELEMENT a                    (#PCDATA)>
<!ATTLIST a
          xlink:type           (simple) #FIXED "simple"
          xlink:href           %uri; #REQUIRED>
```

`xlink:type` ist festgesetzt auf den Wert `simple`, um dieses Element als Simple Link zu markieren.

`xlink:href` verweist bei XLink Simple Links bzw. Locator Elementen innerhalb von Extended Links auf die zu verknüpfende Ressource.

```
<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>Werk 1</title>
  <desc xml:lang="de">
    Schnittstelle von <a xlink:href="http://werk1.example.com">Werk 1</a>
  </desc>
  <desc xml:lang="en">
    Interface of <a xlink:href="http://werk1.example.com">plant 1</a>
  </desc>

  ...

  <typedef id="partno_t">
    <title>partno_t</title>
    <desc>
      Bauteile werden über eine
      <html:b xmlns:html="http://www.w3.org/1999/xhtml">eindeutige</html:b>
      Teilenummer referenziert
    </desc>
    <int/>
  </typedef>

  ...

</system>
```

Abbildung 4.8: Verwendung der allgemeinen Elemente.

4.8 Systembeschreibung

Die Beschreibung eines Systems besteht aus dessen Namen, einer optionalen Beschreibung, der Adresse und dem Typ des verwendeten Kommunikationsprotokolls. <system> ist das Wurzelele-

ment einer Systembeschreibung. Das <address> Element wird zur Spezifikation der Netzwerkadresse benutzt. Das verwendete Kommunikationssystem wird durch das darauffolgende Element, z.B. <corba>, <dcom> oder <rmi>, festgelegt.

<system>

<system> stellt das Wurzelelement einer Systembeschreibung dar. Dabei kann es sowohl zur Beschreibung von lokalen als auch globalen Systemen dienen. Neben den Elementen zur Dokumentation und zur Beschreibung von Netzwerkadresse und Kommunikationsprotokoll enthält es sämtliche Elemente zur Beschreibung der exportierten Schnittstellen.

Um weitestgehende Kompatibilität mit validierenden und nicht-validierenden Parsern zu erreichen, sollten die drei Namensraumdefinitionen in jedem FIX Dokument innerhalb des <system> Elements mit denselben Präfixen enthalten sein (siehe Abbildung 4.9). Dies garantiert sowohl eine Gültigkeit gegenüber der verwendeten DTD als auch eine Interoperabilität mit nicht-validierenden Parsern. Zusätzliche Namensraumdefinitionen können, wie in Abschnitt 4.3 beschrieben, integriert werden.

```
<!ELEMENT system              (title, desc*, address?,
                              (dcom | iiop | local | oncrpc | rmi),
                              (%definition;)*)>
<!ATTLIST system
          xmlns               %uri; #FIXED "http://dcx.com/fix"
          xmlns:fix           %uri; #FIXED "http://dcx.com/fix"
          xmlns:xlink         %uri; #FIXED "http://www.w3.org/1999/xlink">
```

`xmlns` setzt den Default Namensraum auf den FIX Namensraum.

`xmlns:fix` ordnet dem fix: Präfix ebenfalls den FIX Namensraum zu.

`xmlns:xlink` ordnet dem xlink: Präfix den XLink Namensraum zu.

<address>

Durch das <address> Element wird die Netzwerkadresse des Systems spezifiziert. Hierbei kann sowohl ein Hostname als auch eine IP-Adresse angegeben werden.

```
<!ELEMENT address             (#PCDATA)>
```

<dcom>

Das System stellt seine Schnittstelle per DCOM zur Verfügung.

```
<!ELEMENT dcom                EMPTY>
<!ATTLIST dcom
          uuid                %string; #REQUIRED
          version             %float; #REQUIRED>
```

`uuid` Universal Unique Identifier des Interfaces.

`version` Version des Interfaces.

<iiop>

Als Kommunikationsprotokoll wird das Internet Inter Orb Protocol (IIOP) [CORBA, Abschnitt 15.7] benutzt. Da IIOP keinen Standardport spezifiziert, muß dieser immer mit angegeben werden.

```
<!ELEMENT iiop                EMPTY>
<!ATTLIST iiop
          port                %positive-integer; #REQUIRED>
```

`port` TCP/IP Port des IIOP Servers.

<local>

Ein `<local>` Element markiert das System als lokal auf diesem Rechner vorhandenes System[3]. Dies kann z.B. für ein Hilfssystem benutzt werden, das nur für diese Integration benötigt wird und daher auf demselben Rechner, auf dem sich auch das Zielsystem befindet, ausgeführt werden soll. Des weiteren besteht die Möglichkeit, daß sämtliche Systeme lokal auf einem Rechner vorhanden sind.

```
<!ELEMENT local               EMPTY>
<!ATTLIST local
          filename            %string; #REQUIRED>
```

`filename` gibt den Pfad der Bibliothek an.

<oncrpc>

Das System stellt seine Schnittstelle per ONC RPC (früher Sun RPC) [ONCRPC] zur Verfügung.

```
<!ELEMENT oncrpc              EMPTY>
<!ATTLIST oncrpc
          program             %string; #REQUIRED
          version             %non-negative-integer; #REQUIRED
          transport           (tcp | udp) "udp">
```

`program` gibt die Programmnummer des RPC Services an. Dem Attribut muß ein hexadezimal kodierter Wert zugewiesen werden.

`version` gibt die Version des RPC Services an.

`transport` spezifiziert das zu verwendende Transportprotokoll. Mögliche Werte sind dabei `udp` (Default) oder `tcp`.

[3] Beispielsweise als Dynamic Link Library (DLL) unter Windows bzw. Shared Object (SO) unter Unix.

<rmi>

Das System stellt seine Schnittstelle per Java RMI [RMI] zur Verfügung.

```
<!ELEMENT rmi                   EMPTY>
<!ATTLIST rmi
          uri                   %uri; #REQUIRED>
```

uri	Die zum Erhalten des Objekt Handles notwendige URI.

```
<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Werk 2</title>
  <address>werk2.example.com</address>
  <iiop port="683"/>
  ...
</system>
```

Abbildung 4.9: Systembeschreibung.

4.9 Schnittstellendefinition

Innerhalb der Schnittstellendefinition werden sowohl die verwendeten Datentypen als auch die exportierten Funktionen bzw. Attribute definiert. Dieser Teil der Sprache wurde in Art und Umfang stark an OMG IDL angelehnt.

4.9.1 Definition neuer Datentypen

FIX unterstützt alle aus OMG IDL bekannten einfachen und strukturierten Datentypen. Die Definition von neuen Datentypen erfolgt durch das <typedef> Element. Sie enthält, neben dem Namen und der Beschreibung des Datentyps, die Spezifikation eines einfachen oder komplexen Datentyps sowie einen oder mehrere Array Deklaratoren.

Neu definierte Datentypen können wiederum bei der Definition weiterer Datentypen verwendet werden. Außerdem können sie bei der Definition von Operationen oder Attributen als Datentypen benutzt werden. Die Referenzierung neu definierter Datentypen erfolgt hierbei durch das <type> Element.

<typedef>

<typedef> entspricht damit weitestgehend dem IDL Schlüsselwort typedef, mit der Ausnahme, daß Konstrukte in der Art von

```
typedef int stunden, minuten, sekunden;
```

nicht möglich sind. Diese müssen statt dessen in einzelne <typedef> Konstrukte aufgelöst werden.

Abbildung 4.10 verdeutlicht die Verwendung des <typedef> Elements.

```
<!ELEMENT typedef              (title, desc*, %datatype;, array*)>
<!ATTLIST typedef
          id                   ID #IMPLIED>
```

id — Eine gültige XML ID. Die ID stellt eine Möglichkeit zur Referenzierung durch andere Elemente (z.B. <type>) dar. In den im folgenden verwendeten Beispielen werden zumeist die Werte der ID Attribute gleich dem Namen des zugehörigen Elements (Datentyp, Operation, etc.) gewählt. Dies ist nicht zwingend erforderlich, sondern soll nur dem besseren Verständnis der Beispiele dienen. Für ID Attribute bestehen keine über [XML, Abschnitt 3.3.1] hinausgehende Anforderungen.

```
<typedef id="partno_t">                  // Bauteile werden über eine
                                         // eindeutige Teilenummer
  <title>partno_t</title>                // referenziert
  <desc>                                 typedef long partno_t;
    Bauteile werden über eine
    <html:b xmlns:html="http://www.w3.org/1999/xhtml">eindeutige</html:b>
    Teilenummer referenziert
  </desc>
  <int/>
</typedef>
```

Abbildung 4.10: Definition eines neuen Datentyps.

<array>

Mit dem <array> ist es möglich, Arrays zu definieren. Die Definition von mehrdimensionalen Arrays ist durch das mehrfache Vorkommen von <array> Elementen möglich (siehe auch Abbildung 4.11).

```
<!ELEMENT array                EMPTY>
<!ATTLIST array
          count                %positive-integer; #REQUIRED>
```

count — Das count Attribut muß bei einem <array> Element angegeben werden, bei den Elementen <sequence> bzw. <string> ist die Angabe optional. count wird dazu benutzt, eine Obergrenze der in dem jeweiligen Element enthaltenen Elemente vorzugeben. Wird keine Obergrenze angegeben, ist die Länge des Strings bzw. der Sequence unbeschränkt.

<type>

Das <type> Element dient zur Referenzierung bereits durch <typedef>, <struct>, <union> bzw. <enum> definierter Typen.

```
<!ELEMENT type                  EMPTY>
<!ATTLIST type
          xlink:type            (simple) #FIXED "simple"
          xlink:href            %uri; #REQUIRED>
```

xlink:type siehe Abschnitt 8.

xlink:href siehe Abschnitt 8.

```
<typedef>                                   typedef partno_t array_t[20][10];
  <title>array_t</title>
  <type xlink:href="#partno_t"/>
  <array count="20"/>
  <array count="10"/>
</typedef>
```

Abbildung 4.11: Definition eines Arrays.

4.9.2 Einfache Datentypen

Zur Spezifikation von einfachen Datentypen stehen die folgenden Elemente <float>, <int>, <char>, <bool>, <octet>, <any>, <object> und <valuebase> zur Verfügung.

<float>

<float> kennzeichnet Fließkommadatentypen. Es entspricht (durch die entsprechende Wahl des type Attributs) den IDL Typen float, double und long double.

```
<!ELEMENT float                 EMPTY>
<!ATTLIST float
          type                  (float | double | long_double) "double">
```

type wird zur Auswahl des gewünschten Fließkommadatentyps benutzt. Mögliche Werte sind float, double bzw. long_double.

<int>

<int> kennzeichnet einen Integerdatentypen. Es entspricht (durch die entsprechende Wahl des type Attributs) den IDL Typen short, long oder long long bzw. deren vorzeichenlosen (unsigned) Varianten.

```
<!ELEMENT int                 EMPTY>
<!ATTLIST int
          type                (short | long | long_long) "long"
          unsigned            %boolean; "false">
```

type — wird zur Auswahl des gewünschten Integerdatentyps benutzt. Mögliche Werte sind short, long bzw. long_long.

unsigned — Wenn true, handelt es sich um einen vorzeichenlosen Datentyp.

<char>

Das <char> Element wird zur Vereinbarung von Character Typen benutzt. Es faßt die IDL Typen char und wchar zusammen.

```
<!ELEMENT char                EMPTY>
<!ATTLIST char
          wide                %boolean; "false">
```

wide — Das wide Attribut kann in <char> und <string> Elementen verwendet werden. Es dient zur Vereinbarung von IDL wide char bzw. wide string Typen. Ist das wide Attribut auf true gesetzt, wird die wide Variante benutzt.

<bool>

Das <bool> Element dient zur Vereinbarung boolescher Datentypen und entspricht dem IDL boolean Typ.

```
<!ELEMENT bool                EMPTY>
```

<octet>

<octet> entspricht dem gleichnamigen IDL Schlüsselwort. Ein Octet ist ein 8-bit Byte, welches bei einem Transport zu einem anderen System in keiner Form konvertiert wird.

```
<!ELEMENT octet               EMPTY>
```

<any>

<any> entspricht dem gleichnamigen IDL Schlüsselwort. Parameter dieses Datentyps kann jeder Datentyp übergeben werden.

```
<!ELEMENT any                  EMPTY>
```

<object>

<object> entspricht dem IDL Schlüsselwort Object.

```
<!ELEMENT object               EMPTY>
```

<valuebase>

<valuebase> entspricht dem IDL Schlüsselwort ValueBase.

```
<!ELEMENT valuebase            EMPTY>
```

4.9.3 Vorlagentypen

FIX stellt ebenso wie IDL drei sogenannte Vorlagentypen (Template Types) zur Verfügung. Diese können zur Definition von String-, Listen- und Festkommadatentypen benutzt werden.

<sequence>

<sequence> beschreibt ein ein-dimensionales Array variabler Länge. Neben dem Typ der Elemente kann deren Obergrenze angegeben werden. Abbildung 4.12 zeigt die Definition einer Sequenz.

```
<!ELEMENT sequence             (%simple;)>
<!ATTLIST sequence
          count                %positive-integer; #IMPLIED>
```

count siehe Abschnitt 4.9.1.

<string>

<string> dient zur Vereinbarung von Zeichenketten und entspricht dabei einer Sequence vom Typ <char>. Das Element <string> faßt die IDL Typen string und wstring zusammen. In Abbildung 4.13 erfolgt die Definition eines Strings mit der maximalen Länge von sechs Zeichen.

```
<!ELEMENT string               EMPTY>
<!ATTLIST string
          count                %positive-integer; #IMPLIED
          wide                 %boolean; "false">
```

count siehe Abschnitt 4.9.1.

wide siehe Abschnitt 4.9.2.

```
<typedef id="partlist">                    typedef sequence<long> partlist;
  <title>partlist</title>
  <sequence>
  <int/>
  </sequence>
</typedef>
```

Abbildung 4.12: Definition einer Sequenz.

```
<typedef id="partno">                      typedef string<6> partno;
  <title>partno</title>
  <string count="6"/>
</typedef>
```

Abbildung 4.13: Definition eines Strings.

<fixed>

<fixed> beschreibt einen Festkommatyp analog zum IDL fixed Typ. Tritt ein <fixed> Element innerhalb eines <const> Elements auf, dürfen die Attribute digits bzw. scale nicht angegeben werden, ansonsten müssen beide Attribute angegeben werden! Abbildung 4.14 illustriert die Definition eines Festkommadatentyps.

```
<!ELEMENT fixed                EMPTY>
<!ATTLIST fixed
          digits               %positive-integer; #IMPLIED
          scale                %positive-integer; #IMPLIED>
```

digits — spezifiziert die Gesamtzahl der Ziffern.

scale — gibt die Anzahl der Nachkommastellen an.

```
<typedef id="price_t">                     typedef fixed<7,2> price_t;
  <title>price_t</title>
  <fixed digits="7" scale="2"/>
</typedef>
```

Abbildung 4.14: Definition eines Festkommatyps.

4.9.4 Strukturierte Datentypen

Zur Beschreibung von strukturierten Datentypen stehen die Elemente <struct>, <union> und <enum> zur Verfügung. Im Gegensatz zu den einfachen bzw. Vorlagentypen kann die Definition neuer Datentypen auf Basis von komplexen Datentypen auch ohne umgebendes <typedef> Element erfolgen. Es ist in FIX nicht erlaubt, rekursive Definitionen strukturierter Datentypen vorzunehmen.

<struct>

Strukturen werden durch das <struct> Element beschrieben. Sie bestehen aus einem oder mehreren (durch das <member> Element beschriebenen) Komponenten. Diese Komponenten können wiederum einfache oder komplexe Datentypen sein.

```
<!ELEMENT struct              (title, desc*, member+)>
<!ATTLIST struct
          id                  ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

<member>

<member> dient zur Definition der Elemente von Strukturen oder Exceptions (Abschnitt 4.9.7). Die Beschreibung des Elementtyps erfolgt dabei analog zum <typedef> Element. Ein innerhalb eines <member> Elements vorkommendes <type> Element darf dabei nicht die übergeordnete Struktur referenzieren.

Das Beispiel in Abbildung 4.15 enthält die Definition einer Struktur dimension_t bestehend aus den drei Elementen x, y und z.

```
<!ELEMENT member              (title, desc*, %datatype;, array*)>
<!ATTLIST member
          id                  ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

<union>

<union> dient zusammen mit den <case> und <default> Elementen zur Definition von Unionen. Sie besitzen denselben Aufbau wie IDL Unionen, stellen also eine Mischung aus den in C verwendeten Unionen und einer switch Anweisung dar. Das <union> Element enthält neben dem Datentyp des Auswahlelements ein oder mehrere <case> Elemente und ein optionales <default> Element zur Definition der möglichen Alternativen der Union. Abbildung 4.16 veranschaulicht die Definition einer Union anhand eines Beispiels.

```
<!ELEMENT union               (title, desc*;, (int | char | bool | enum |
                              type), case+, default?)>
<!ATTLIST union
          id                  ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

```
<typedef id="dimension_t">                typedef struct dimension {
  <title>dimension_t</title>                float x;
  <struct id="dimension">                   float y;
    <title>dimension</title>                float z;
    <member id="dimension.x">             } dimension_t;
      <title>x</title>
      <float type="float"/>
    </member>
    <member id="dimension.y">
      <title>y</title>
      <float type="float"/>
    </member>
    <member id="dimension.z">
      <title>z</title>
      <float type="float"/>
    </member>
  </struct>
</typedef>
```

Abbildung 4.15: Definition einer Struktur.

<case>

Das <case> Element dient zur Definition einer Alternative der Union. Neben dem Datentyp muß mit dem <expr> Element (siehe Abschnitt 4.9.6) ein Ausdruck zur Auswahl dieser Alternative angegeben werden.

```
<!ELEMENT case                (title, desc*, %datatype;, array*, expr)>
<!ATTLIST case
          id                  ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

<default>

Mit dem <default> Element wird die Defaultalternative einer Union definiert. Das <default> Element entspricht einem <case>, welches kein <expr> Element beinhaltet.

```
<!ELEMENT default             (title, desc*, %datatype;, array*)>
<!ATTLIST default
          id                  ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

```
<union>                                    union example switch (long) {
  <title>example</title>                     case 1 : long integer;
  <int/>                                     case 2 : double floatingpoint;
  <case>                                     default : char zeichen;
    <title>integer</title>                 }
    <int/>
    <expr type="text/idl">1</expr>
  </case>
  <case>
    <title>floatingpoint</title>
    <float/>
    <expr type="text/idl">2</expr>
  </case>
  <default>
    <title>zeichen</title>
    <char/>
  </default>
</union>
```

Abbildung 4.16: Definition einer Union.

<enum>

<enum> leitet die Vereinbarung eines Aufzählungstyps (Enumeration) ein. Neben den Beschreibungselementen enthält es mindestens ein <enumerator> Element. Die <enum> und <enumerator> Elemente entsprechen dem enum Konstrukt der IDL (siehe auch Abbildung 4.17).

```
<!ELEMENT enum                 (title, desc*, enumerator+)>
<!ATTLIST enum
          id                   ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

<enumerator>

Mit dem <enumerator> Element werden die möglichen Werte beschrieben, die ein Aufzählungstyp annehmen kann. Bei der Konvertierung (z.B. nach IDL) wird der Inhalt des <title> Elements als Wert benutzt.

```
<!ELEMENT enumerator           (title, desc*)>
<!ATTLIST enumerator
          id                   ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

```
<typedef id="color_t">
  <title>color_t</title>
  <desc>
    mögliche Farben
  </desc>
  <enum id="color">
    <title>color</title>
    <enumerator id="color.red">
      <title>red</title>
    </enumerator>
    <enumerator id="color.green">
      <title>green</title>
    </enumerator>
    <enumerator id="color.yellow">
      <title>yellow</title>
    </enumerator>
    <enumerator id="color.blue">
      <title>blue</title>
    </enumerator>
  </enum>
</typedef>
```

```
typedef enum color {red, green,
    yellow, blue} color_t;
```

Abbildung 4.17: Definition einer Aufzählung.

4.9.5 <native>

Das <native> Element entspricht dem IDL Schlüsselwort native.

```
<!ELEMENT native              (title, desc*)>
<!ATTLIST native
          id                  ID #IMPLIED>
          %id;>
```

id siehe Abschnitt 4.9.1.

4.9.6 Konstanten

Die Definition von Konstanten erfolgt ähnlich zu der Definition von Datentypen. Sie besteht aus dem Namen der Konstanten, deren Datentyp und einer optionalen Wertzuweisung. Zum Zweck der Modellierung der Abhängigkeiten wird der Wert einer Konstante nicht benötigt. Soll aus einem FIX Dokument aber beispielsweise ein IDL Dokument erzeugt werden, ist eine Konvertierung von Konstantendefinitionen ohne Wertzuweisung nicht möglich (siehe Abschnitt 4.14).

Wird innerhalb einer Konstantendefinition der Typ durch ein <type> Element spezifiziert, darf dieses nur auf Typdefinitionen verweisen, deren Basistyp <long>, <char>, <bool>, <float>, <string>, <octet> oder <enum> ist. Wird innerhalb eines <const> Elements ein <fixed> Element benutzt, darf dieses kein digits oder scale Attribut enthalten.

<const>

<const> dient zur Definition von Konstanten und entspricht dabei dem IDL const Konstrukt[4]. Neben der Angabe des Namens und eines Typs kann dabei optional mit dem <expr> Element der Wert der Konstanten festgelegt werden.

```
<!ELEMENT const          (title, desc*, (int | char | bool | float |
                         string | fixed | type | octet), expr?)>
<!ATTLIST const
          id             ID #IMPLIED>
```

id siehe Abschnitt 4.9.1.

<expr>

Mit Hilfe des <expr> Elements ist die Modellierung von konstanten Ausdrücken für die Definition von Konstanten bzw. Unions möglich. Die Modellierung kann dabei in verschiedenen Formaten erfolgen. Denkbar sind unter anderem:

- IDL

  Die Beschreibung des Konstantenwerts erfolgt mit der normalen IDL Notation direkt innerhalb des <expr> Elements. Um den Inhalt des <expr> Elements als IDL zu kennzeichnen, muß das type="text/x-idl" Attribut angegeben werden.

- MathML [MATHML]

  MathML ist eine ebenfalls XML-basierte Sprache, die ähnlich LaTeX zur Darstellung mathematischer Ausdrücke entwickelt wurde. Im Gegensatz zu LaTeX erlaubt MathML auch die semantische Repräsentation von Ausdrücken. MathML ist eine Recommendation des World Wide Web Consortium, für die bereits Parser existieren. Die Modellierung der Wertzuweisung durch ein MathML Objekt erfolgt durch die Einbettung des MathML Wurzelelements <math> innerhalb des <expr> Elements. Zusätzlich wird die Verwendung von MathML durch das Attribut type="text/mathml" gekennzeichnet. Abbildung 4.19 illustriert die Definition einer numerischen Konstante.

- OpenMath [OPENMATH]

  OpenMath ist ein in der Entwicklung befindlicher Standard zur semantischen Repräsentation mathematischer Ausdrücke. Analog zu MathML wird ein OpenMath Objekt durch das OpenMath Wurzelelement <omobj> und das type="text/x-openmath" Attribut gekennzeichnet.

- Text

  Zur Darstellung von String Konstanten ist keine weitere Sprache notwendig. Dies wird durch das Attribut type="text/plain" gekennzeichnet. In diesem Fall hängt die Behandlung des Inhalts des <expr> Elements auch von dem xml:space Attribut ab. Das Beispiel in Abbildung 4.18 enthält die Definition einer Stringkonstante.

[4] Im Gegensatz zu [CORBA, Abschnitt 3.4, Produktion 28] (aber in Übereinstimmung mit [CORBA, Abschnitt 3.9.2]) sind auch Aufzählungstypen als Konstanten möglich.

- FIX

  Konstanten für Enumeration können durch ein <type> Element innerhalb des <expr> Elements identifiziert werden, welches auf das entsprechende <enumerator> Element verweist. In diesem Fall muß das type Attribut des <expr> Elements auf den Wert "application/fix+xml" gesetzt werden.

```
<!ELEMENT expr                ANY>
<!ATTLIST expr
          type                %contenttype; #REQUIRED
          xml:space           (default | preserve) "default">
```

`type` enthält den MIME-Typ [MIME] des innerhalb des <expr> Elements verwendeten Formats. Mögliche Werte sind u.a. `text/x-idl`, `text/mathml`, `text/x-openmath`, `text/plain` oder `application/fix+xml`.

`xml:space` Speziell für die Kodierung `text/plain` kann mit dem `xml:space` Attribut [XML, Abschnitt 2.10] das Verhalten des XML-Parsers bzgl. Leerzeichen beeinflußt werden.

```
<const id="name">                        const string name = "Werk 1";
  <title>name</title>
  <string/>
  <expr type="text/plain" xml:space="preserve">Werk 1</expr>
</const>
```

Abbildung 4.18: Definition einer Stringkonstante.

```
<const>                                  const long version = 42;
  <title>version</title>
  <int/>
  <expr type="text/mathml">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <cn>42</cn>
    </math>
  </expr>
</const>
```

Abbildung 4.19: Definition einer numerischen Konstante.

4.9.7 Operationen und Attribute

Zur Definition von Operationen (also Funktionen bzw. Methoden) wird das <op> Element benutzt. Es beinhaltet die Definition der Parameter und des Rückgabewerts, der Exceptions, die während der Ausführung auftreten können, und die der Funktion zugeordneten Abhängigkeitsgraphen. Die Definition

der Parameter und des Rückgabewerts einer Funktion erfolgt durch <param> Elemente bzw. durch das <return> Element. Tritt kein <return> Element auf, besitzt die beschriebene Operation keinen Rückgabewert.

Eine Exception wird durch das <except> Element definiert. Innerhalb eines <op> Elements werden die möglichen Exceptions durch ein oder mehrere <raises> Elemente definiert.

Attribute werden durch das <attr> Element definiert. Eine Attributdefinition beinhaltet neben dem Namen und einer optionalen Beschreibung dessen Datentyp und die ihm zugeordnete Abbildungsgraphen.

<op>

Das <op> Element dient zur Definition einer Funktion bzw. Methode. Diese Definition erfolgt durch den Namen der Funktion, deren Signatur (durch die Elemente <param> bzw. <return>), der optionalen Definition möglicher Exceptions durch <raises> Elemente und durch die optionale Angabe eines Kontexts (siehe Abbildung 4.20). Zusätzlich kann ein <op> Element ein oder mehrere <graph> Elemente enthalten (siehe Abschnitt 4.10.1).

```
<!ELEMENT op                  (title, desc*, param*, return?, raises*,
                              context*, effect*, graph*)>
<!ATTLIST op
          id                  ID #IMPLIED
          oneway              %boolean; "false"
          semantic            %semantic; "undefined">
```

`id` siehe Abschnitt 4.9.1.

`oneway` entspricht dem IDL `oneway` Attribut, es gelten die in [CORBA, Abschnitt 3.12.1] beschriebenen Ausführungssemantiken. Ist `oneway` auf `true` gesetzt, darf das <op> Element keine <return> oder <raises> Elemente enthalten, ebenso keine <param> Elemente, deren `type` Attribut auf `in` bzw. `inout` gesetzt ist.

`semantic` Das `semantic` Attribut dient zur Angabe von semantischen Hinweisen für die graphische Oberfläche (siehe Abschnitt 3.7.2). Die möglichen Werte behandelt Abschnitt 4.11.

<param>

<param> Elemente beschreiben die einzelnen Parameter einer Funktion. Neben der Angabe des Namens und des Datentyps beschreibt das `type` Attribut, ob es sich um einen Eingabe- oder Ausgabeparameter handelt.

```
<!ELEMENT param               (title, desc*, %param;)>
<!ATTLIST param
          id                  ID #IMPLIED
          type                (in | out | inout) "in">
```

`id` siehe Abschnitt 4.9.1.

`type` gibt die Art des Parameters an. Mögliche Werte sind:

`in`	Eingabeparameter;
`out`	Ausgabeparameter;
`inout`	Eingabe- und Ausgabeparameter.

<return>

Das <return> Element dient zur Beschreibung des Rückgabeparameters einer Operation. Es entspricht dabei einem <param> Element mit gesetztem type="out" Attribut und fehlendem <title> Element. Enthält ein <op> Element kein <return> Element, entspricht dies der Deklaration einer Operation als void in IDL.

```
<!ELEMENT return              (desc*, %param;)>
<!ATTLIST return
          id                  ID #IMPLIED>
```

`id` siehe Abschnitt 4.9.1.

```
<op id="get_part">
  <title>get_part</title>
  <param id="get_part.no">
    <title>no</title>
    <type xlink:href="#partno"/>
  </param>
  <param id="get_part.name" type="out">
    <title>name</title>
    <string/>
  </param>
  <param id="get_part.dim" type="out">
    <title>dim</title>
    <type xlink:href="#dimension_t"/>
  </param>
  <return id="get_part.return">
    <bool/>
  </return>
</op>
```

```
boolean get_part(in partno no,
    out string name,
    out dimension_t dim);
```

Abbildung 4.20: Definition einer Operation.

<raises>

Die innerhalb eines <op> Elements vorkommenden <raises> Elemente referenzieren durch ihr xlink:href Attribut die Exceptions, die während der Ausführung der Funktion auftreten können.

```
<!ELEMENT raises                EMPTY>
<!ATTLIST raises
          xlink:type            (simple) #FIXED "simple"
          xlink:href            %uri; #REQUIRED>
```

`xlink:type` siehe Abschnitt 8.

`xlink:href` siehe Abschnitt 8.

<except>

Neue Exceptions werden durch das <except> Element definiert. Neben dem Namen kann eine Exception, ähnlich einer Struktur, mehrere, mittels des <member> Elements definierte Komponenten enthalten (Abbildung 4.21).

```
<!ELEMENT except                (title, desc*, member*)>
<!ATTLIST except
          id                    ID #IMPLIED>
```

`id` siehe Abschnitt 4.9.1.

```
<except id="example_e">
  <title>example_e</title>
  <member id="internal_error">
    <title>internal_error</title>
    <int/>
  </member>
</except>

<op id="get_part" semantic="read">
  <title>get_part</title>
  <param id="get_part.partno">
    <title>partno</title>
    <type xlink:href="#partno_t"/>
  </param>
  <param id="get_part.name" type="out">
    <title>name</title>
    <string/>
  </param>
  <param id="get_part.color" type="out">
    <title>color</title>
    <type xlink:href="#color_t"/>
  </param>
  <raises xlink:href="#example_e"/>
</op>
```

Abbildung 4.21: Definition einer Operation und deren Exceptions.

<context>

Der Inhalt des <context> Elements wird der aufgerufenen Funktion als Kontext [CORBA, Abschnitt 3.12.4] übergeben.

```
<!ELEMENT context              (#PCDATA)>
```

<attr>

<attr> dient zur Definition der Attribute eines Interfaces. Neben Name und Typ des Attributs können in einem Zielsystem zwei <graph> Elemente zur Beschreibung des schreibenden bzw. lesenden Zugriffs enthalten sein (siehe Abbildung 4.22).

```
<!ELEMENT attr                 (title, desc*, %param;, effect*, graph*)>
<!ATTLIST attr
          id                   ID #IMPLIED
          readonly             %boolean; "false">
```

id	siehe Abschnitt 4.9.1.
readonly	Ist das readonly Attribut auf true gesetzt, ist es nicht möglich, schreibend auf das vereinbarte Attribut zuzugreifen. Das readonly Attribut entspricht dem IDL readonly Schlüsselwort.

```
<attr id="num_parts" readonly="true">   readonly attribute long num_parts;
  <title>num_parts</title>
  <int/>
</attr>
```

Abbildung 4.22: Definition eines Attributs.

4.9.8 Module und Interfaces

Die zur Verfügung gestellten Schnittstellen eines Systems können zur Strukturierung in verschiedene Teile unterteilt werden. Dazu stehen die Elemente <module> und <interface> zur Verfügung.

Einzelne, durch <interface> Elemente definierte Schnittstellen können untereinander in Vererbungsbeziehungen stehen. Eine Schnittstelle, die von einer anderen Schnittstelle erbt, implementiert zusätzlich zu ihren eigenen Operationen und Attributen die der beerbten Schnittstelle. Innerhalb eines <interface> Elements können zu beerbende Schnittstellen durch ein oder mehrere <inherits> Elemente referenziert werden.

Abbildung 4.23 illustriert die Verwendung der hier beschriebenen Elemente.

<module>

Das <module> Element dient zur Unterteilung eines Systems in verschiedene Module. Bis auf die Elemente zur Beschreibung der Netzwerkadresse und des Kommunikationsprotokolls kann es die gleichen Definitionen enthalten wie das <system> Element. So können <module> Elemente auch geschachtelt werden. Im Gegensatz zum <inteface> Element können innerhalb eines <module> Elements keine Operationen oder Attribute definiert werden.

Das <module> Element entspricht im wesentlichen [CORBA, Abschnitt 3.4, Produktion 3].

```
<!ELEMENT module              (title, desc*, (%definition;)+)>
```

<interface>

Innerhalb eines <interface> Elements ist ebenfalls die Definition von Datentypen, Konstanten und Exceptions möglich. Zusätzlich können Operationen und Attribute definiert werden. Ein <interface> Element kann keine weiteren <module> oder <interface> Elemente beinhalten.

```
<!ELEMENT interface           (title, desc*, inherits*, (%export;)*)>
<!ATTLIST interface
          id                  ID #IMPLIED
          abstract            %boolean; "false">
```

`id` siehe Abschnitt 4.9.1.

`abstract` wird zur Kennzeichnung von abstrakten Interfaces benutzt und entspricht dem `abstract` Schlüsselwort von IDL.

<inherits>

Das <inherits> Element dient zur Beschreibung von Vererbungsbeziehungen zwischen Interfaces oder Value Types (siehe Abschnitt 4.9.9). Es kann innerhalb von <interface> oder <valuetype> Elementen mehrfach auftreten und verweist jeweils auf ein weiteres <interface> bzw. <valuetype> Element, von dem das aktuelle Element erbt. Es gelten die Vererbungsregeln aus [CORBA, Abschnitt 3.7.5 bzw. 3.8.5].

```
<!ELEMENT inherits            EMPTY>
<!ATTLIST inherits
          xlink:type          (simple) #FIXED "simple"
          xlink:href          %uri; #REQUIRED>
```

`xlink:type` siehe Abschnitt 8.

`xlink:href` siehe Abschnitt 8.

```
<module>
  <title>Module</title>
  <interface id="a">
    <title>a</title>
    ...
  </interface>

  <interface>
    <title>b</title>
    <inherits xlink:href="#a"/>
  </interface>
</module>
```

Abbildung 4.23: Gliederung von Schnittstellen.

4.9.9 Value Types

Value Types entsprechen den gleichnamigen CORBA Konstrukten. Durch sie können Objekte mit deren Methoden und Attributen definiert werden. Im Gegensatz zu Interfaces können Value Types auch als Parameter an andere Methoden übergeben werden.

`<valuetype>`

Das `<valuetype>` Element dient zur Definition eines Value Types. Zur Beschreibung der Vererbungsbeziehungen kann es `<inherits>` bzw. `<supports>` Elemente beinhalten. Dabei dienen `<inherits>` Elemente zur Referenzierung anderer Value Types, während `<supports>` Elemente Interfaces beschreiben, welche der Value Type implementiert.

Es gibt normale Value Types und sogenannte Boxed Value Types. Während erstere neben sämtlichen innerhalb des `<interface>` Elements erlaubten Elementen zusätzlich noch `<statemember>` oder `<factory>` Elemente beinhalten können, dürfen Boxed Value Types nur ein einzelnes `<boxed>` Element beinhalten.

```
<!ELEMENT valuetype            (title, desc*, inherits*, supports*,
                                ((%export; | statemember | factory)* | boxed))>
<!ATTLIST valuetype
          id                   ID #IMPLIED
          custom               %boolean; "false"
          abstract             %boolean; "false"
          truncatable          %boolean; "false">
```

`id` siehe Abschnitt 4.9.1.

`custom` gibt an, ob für diesen Value Type ein spezielles Marshalling verwendet werden soll.

`abstract` siehe Abschnitt 4.9.8.

`truncatable` entspricht dem CORBA `truncatable` Schlüssel Wort (siehe auch [CORBA, Abschnitt 3.8.1.3]).

<supports>

Das <supports> Element dient ähnlich dem <inherits> Element zur Beschreibung von Vererbungsbeziehungen. Im Gegensatz zu diesem kann es nur innerhalb eines <valuetype> Elements vorkommen. Es dient zur Referenzierung von <interface> Elementen, welche durch den Value Type implementiert werden.

```
<!ELEMENT supports              EMPTY>
<!ATTLIST supports
          xlink:type            (simple) #FIXED "simple"
          xlink:href            %uri; #REQUIRED>
```

xlink:type siehe Abschnitt 8.

xlink:href siehe Abschnitt 8.

<boxed>

Innerhalb des <valuetype> Elements darf das <boxed> Element nur einmal auftreten. Das Element selbst darf nur einen (einfachen oder komplexen) Datentyp beinhalten.

```
<!ELEMENT boxed                 (%datatype;)>
```

<statemember>

Ein Statemember definiert ein Element des Zustands, welches bei der Übergabe des Value Types als Parameter mit übergeben und bei Bedarf auch über das Netz übertragen wird. Statemember können entweder privat oder öffentlich sein. Auf private Statemembers kann nur der Code der Implementierung des Value Types zugreifen. Die Verwendung von <statemember> Elementen erfolgt ähnlich zu der von <attr> Elementen.

```
<!ELEMENT statemember           (title, desc*, %datatype;, array*, graph*)>
<!ATTLIST statemember
          id                    ID #IMPLIED
          type                  (public | private) "public">
```

id siehe Abschnitt 4.9.1.

type dient zur Unterscheidung zwischen privaten und öffentlichen Statemembern.

<factory>

Factories werden durch das <factory> Element definiert. Die Verwendung des <factory> Elements erfolgt ähnlich zu der des <op> Elements. Die innerhalb des <factory> Elements enthaltenen <param> Elemente müssen Eingabeparameter sein (type="in"), ebenso darf kein Rückgabewert definiert werden.

```
<!ELEMENT factory              (title, desc*, param*, graph)>
<!ATTLIST factory
          id                   ID #IMPLIED
          semantic             %semantic; "undefined">
```

`id` siehe Abschnitt 4.9.1.

`semantic` siehe Abschnitt 4.9.7.

4.10 Abhängigkeitsbeschreibung

4.10.1 Grundlagen

Die Beschreibung der Abhängigkeiten erfolgt in FIX mittels eines XLink Extended Links, repräsentiert durch das <graph> Element. Das <graph> Element kann sowohl in <op> als auch in <attr> Elementen auftreten. Innerhalb eines <op> Elements dient es zur Modellierung einer globalen Funktion oder eines Quellsystem-Wrappers, innerhalb eines <graph> Elements wird es zur Beschreibung des lesenden bzw. schreibenden Zugriff auf ein globales Attribut benutzt.

Innerhalb des <graph> Elements erfolgt die Beschreibung der Knoten des Abhängigkeitsgraphen durch <node> Elemente, die Kanten werden durch <dep> Elemente repräsentiert.

<graph>

Das <graph> Element leitet einen Abhängigkeitsgraphen ein. Den Typ des Abhängigkeitsgraphen bestimmt dabei das `xlink:role` Attribut. <graph> Elemente können direkt innerhalb von <op>, <attr>, <factory> oder <statemember> Elementen oder aber als externe Graphen (Abschnitt 4.10.13) innerhalb eines <mapping> Elements auftreten. <graph> Elemente, die innerhalb eines Zielsystems auftreten (bzw. mit Elementen eines Zielsystems verknüpft sind), dienen zur Modellierung eines Abhängigkeitsgraphen für globale Funktionen bzw. Attribute. Innerhalb von Quellsystemen können <graph> Elemente zur Modellierung von Wrappern für Quellfunktionen (siehe Abschnitt 4.10.12) benutzt werden.

```
<!ELEMENT graph                (desc*, ((node | todo | dep | done)+ | ext))>
<!ATTLIST graph
          id                   ID #IMPLIED
          xlink:type           (extended) #FIXED "extended"
          xlink:role           (exec | read | write |
                               xlink:external-linkset) "exec"
          type                 (implementation | wrapper) "implementation">
```

`id` siehe Abschnitt 4.9.1.

`xlink:type` ist festgesetzt auf den Wert `extended`, um dieses Element als Extended Link zu markieren.

`xlink:role` dient zur Kennzeichnung des Typs des Abhängigkeitsgraphen (Tabelle 4.1).

`type` unterscheidet reguläre Abhängigkeitsgraphen von Wrappern für Quellsystemfunktionen (siehe Abschnitt 4.10.12).

Wert	mögliche Vaterelemente	Erläuterung
exec	<op>, <factory>	Ausführung einer Operation
read	<attr>, <statemember>	lesender Zugriff auf Attribute
write	<attr>, <statemember>	schreibender Zugriff auf Attribute
xlink:external-linkset	<op>, <factory>, <attr>, <statemember>	Verweis auf externen Graph

Tabelle 4.1: Mögliche Werte des xlink:role Attributs.

<node>

<node> Elemente sind die Locator Elemente des durch <graph> erzeugten Extended Links. Sie dienen zur Referenzierung der externen Ressourcen. Referenziert werden können dabei Parameter, Konstanten, Operationen, Attribute und Exceptions. <node> Elemente repräsentieren die Knoten des Abhängigkeitsgraphen. Die möglichen Elemente, auf die ein <node> Element verweisen kann, sind: <param>, <return>, <op>, <factory>, <effect>, <attr>, <statemember> und <const>. Außerdem kann ein <node> Element auch auf ein <enumerator> Element verweisen, um dessen Wert als Konstante zu benutzen. So wird dies beispielsweise bei der Modellierung der globalen Funktion get_price_euro() (siehe Anhang E.5) benutzt.

```
<!ELEMENT node                  (title, desc*)>
<!ATTLIST node
          xlink:type            (locator) #FIXED "locator"
          xlink:href            %uri; #REQUIRED
          xlink:role            %Name; #REQUIRED
          instance              %Name; #IMPLIED
          semantic              %semantic; "undefined"
          compensation          %boolean; "true"
          coords                %coords; #IMPLIED>
```

xlink:type — ist festgesetzt auf den Wert locator, um dieses Element als Locator Element eines Extended Links zu markieren.

xlink:href — siehe Abschnitt 8. Innerhalb des <node> Elements kann neben der normalen XPointer Syntax auch die erweiterte FIXPointer Syntax (siehe Abschnitt 4.12) benutzt werden.

xlink:role — wird zur Referenzierung innerhalb von <dep> Elementen benutzt. Aus diesem Grund muß jedem <node> Element ein innerhalb des umgebenden <graph> Elements eindeutiges xlink:role Attribut zugeordnet werden.

instance — dient zur Unterscheidung mehrerer Instanzen einer Funktion innerhalb eines Abhängigkeitsgraphen (siehe auch Abschnitt 4.10.3).

semantic — <node> Elemente können ebenfalls ein semantic Attribut enthalten. Dies kann erfolgen, falls innerhalb des Quellsystems keine Semantik für diese Funktion angegeben wurde, oder um eine angegebene Semantik zu überschreiben, falls die Funktion innerhalb dieses Graphen zu einem anderen Zweck als dem ursprünglich vorgesehenen gebraucht wird. Siehe auch Abschnitt 4.9.7 und 4.11.

`compensation` gibt an, ob eine innerhalb der Definition der Operation angegebene Defaultkompensation ausgeführt werden soll (Default) oder nicht.

`coords` dient zur Spezifikation einer durch Kommata getrennten Liste von x,y Koordinatenpaaren. In den Elementen <node> und <todo> wird nur das erste Koordinatenpaar verwendet. Die Werte dienen hierbei zur Festlegung der Bildschirmposition des jeweiligen Elements. Innerhalb eines <dep> Elements werden die Koordinatenpaare als Punkte eines zu zeichnenden Linienzuges interpretiert.

<dep>

<dep> Elemente dienen als Arc Elemente zur Verknüpfung der <node> oder <todo> Elemente untereinander. Mit <dep> Elementen werden Abhängigkeitsrelationen, Kompensationsabhängigkeiten und Verknüpfungen von Funktionen und Exceptions realisiert. Eine Aufstellung aller möglichen Abhängigkeitsrelationen beinhaltet Anhang F.

Wird durch ein <dep> Element eine Kopieraktion modelliert (also z.B. durch eine Abhängigkeit der Art Parameter → Parameter), so werden außer den innerhalb von CORBA definierten Datentypkonvertierungen (wie z.B. `int` → `long`) keinerlei Konvertierungen durchgeführt. Dies muß durch die Verwendung einer entsprechenden Hilfsfunktion erfolgen.

Die Abbildung 4.24 zeigt die Modellierung der globalen Funktion `get_name()` mit dem in Abbildung 3.1 auf Seite 21 dargestellten Abhängigkeitsgraphen.

```
<!ELEMENT dep                   (%descr;)>
<!ATTLIST dep
          xlink:type            (arc) #FIXED "arc"
          xlink:from            %Name; #REQUIRED
          xlink:to              %Name; #REQUIRED
          xlink:role            (implementation | compensation | exception)
                                "implementation"
          invocation            (single | cyclic) "single"
          completion            (required | sufficient | optional) #IMPLIED
          priority              %integer; "0"
          coords                %coords; #IMPLIED>
```

`xlink:type` ist festgesetzt auf den Wert `arc`, um dieses Element als Arc Element eines Extended Links zu markieren.

`xlink:from` dient zur Spezifikation des Ausgangspunkts einer Abhängigkeitsrelation. Es wird dabei das <node> Element referenziert, dessen `xlink:role` Attribut mit dem Wert von `xlink:from` übereinstimmt.

`xlink:to` dient analog zu `xlink:from` zur Spezifikation des Ziels einer Abhängigkeitsrelation.

`xlink:role` spezifiziert den Typ der Abhängigkeitsrelation. Mögliche Werte sind:

- `implementation` kennzeichnet eine normale Abhängigkeitsrelation.

	`compensation`	markiert eine Kompensationsabhängigkeit innerhalb des Abhängigkeitsgraphen (siehe Abschnitt 4.10.11). Bei der Ausführung wird nur bei einem auftretenden Fehler versucht, diese Abhängigkeitsrelation zu erfüllen.
	`exception`	dient zur Verknüpfung einer Funktion mit einer Exception. Schlägt die Ausführung dieser Funktion fehl (und können die bestehenden Abhängigkeiten auf keinem anderen Weg aufgelöst werden), wird die verknüpfte Exception ausgelöst (Abschnitt 4.10.10).
`invocation`	Ist das `invocation` Attribut auf `cyclic` gesetzt, erfolgt eine zyklische Ausführung wie in Abschnitt 4.10.8 beschrieben.	
`completion`	gibt den Typ der Abhängigkeit in Bezug auf die redundante Ausführung an. Mögliche Werte sind `required`, `sufficient` oder `optional`. Die Bedeutung der Werte ist in Abschnitt 4.10.9 beschrieben.	
`priority`	bestimmt die Priorität einer als `completion="optional"` markierten Abhängigkeitsrelation, dabei stehen größere Werte für eine höhere Priorität. Handelt es sich nicht um eine optionale Abhängigkeitsrelation, hat der Wert keine Bedeutung.	
`coords`	siehe Abschnitt 4.10.1	

```
<op id="get_name" semantic="read">
  <title>get_name</title>
  <param id="get_name.no">
    <title>no</title>
    <int/>
  </param>
  <return id="get_name.return">
    <string/>
  </return>
  <graph>
    <node xlink:href="#get_name.no" xlink:role="get_name.no"/>
    <node xlink:href="#get_name.return" xlink:role="get_name.return"/>
    <node xlink:href="werk1.xml#get_part.partno"
     xlink:role="get_part.partno"/>
    <node xlink:href="werk1.xml#get_part.name"
     xlink:role="get_part.name"/>

    <dep xlink:from="get_name.no" xlink:to="get_part.partno"/>
    <dep xlink:from="get_part.name" xlink:to="get_name.return"/>
  </graph>
</op>
```

Abbildung 4.24: Abhängigkeitsgraph.

4.10.2 Abhängigkeiten zwischen Funktionen

Die in Abschnitt 3.1.4 eingeführte Modellierung der Ausführungsreihenfolge durch Abhängigkeiten zwischen Funktionen wurde innerhalb von FIX dahingehend erweitert, daß explizit Seiteneffekte einzelner Funktionen modelliert werden können. Andere Funktionen können in Abhängigkeit von diesen Seiteneffekten stehen. Seiteneffekte werden innerhalb von <op> oder <attr> Elementen durch <effect> Elemente modelliert.

Zur Modellierung der Ausführungsreihenfolge kann das xlink:from Attribut von <node> Elementen auf ein <effect> Element und das xlink:to Element auf ein <op> Element verweisen. Als Sonderfall besteht die Möglichkeit, das xlink:from Attribut auf ein <node> Element verweisen zu lassen, welches auf ein <op> Element zeigt. Dies entspricht dem Verweis auf ein in dem <op> Element enthaltenes <effect> Element ohne Inhalt. Dieser Sonderfall ist auf <attr> Elemente nicht erweiterbar, da in diesem Fall nicht klar ist, ob sich die Abhängigkeit aus einem lesenden oder schreibenden Zugriff ergibt.

<effect>

Das <effect> Element beschreibt einen auftretenden Seiteneffekt. Neben dem Namen des Seiteneffekts kann eine Beschreibung innerhalb ein oder mehrerer <desc> Elemente angegeben werden.

```
<!ELEMENT effect              (title, desc*)>
<!ATTLIST effect
          id                  ID #IMPLIED
          type                (exec | read | write) "exec">
```

id — siehe Abschnitt 4.9.1.

type — gibt an, bei welcher Aktion der Nebeneffekt auftritt. Steht das <effect> Element innerhalb eines <op> Elements, ist nur der Defaultwert exec zulässig. Innerhalb eines <attr> Elements wird durch das type Attribut angegeben, ob der Seiteneffekt bei lesendem oder schreibendem Zugriff auftritt.

Die Definition von Seiteneffekten illustriert Abbildung 4.25 anhand der lokalen Funktion init() von Werk 1. Abbildung 4.26 zeigt die Modellierung von Abhängigkeiten von Seiteneffekten anhand der globalen init() Funktion. Es wurde dabei der in Abbildung 3.24 auf Seite 39 dargestellte Abhängigkeitsgraph modelliert.

4.10.3 Mehrfache Instantiierung von Funktionen

Die in Abschnitt 3.1.7 beschriebene mehrfache Instantiierung von Funktionen erfolgt in FIX durch mehrere <node> Elemente, die das gleiche Ziel referenzieren, aber verschiedene instance Attribute besitzen. Solche Elemente müssen ebenfalls unterschiedliche xlink:role Attribute besitzen, da diese nicht doppelt vorkommen dürfen. <node> Elemente, deren instance Attribute den gleichen Wert besitzen, gehören zur selben Instanz der Funktion. Abbildung 4.27 zeigt einen Ausschnitt der Realisierung der Abbildung 3.8 von Seite 26.

```
<op id="init">
  <title>init</title>
  <raises xlink:href="#example_e"/>
  <effect id="neustart">
    <title>Neustart</title>
    <desc xml:lang="de">
      Das System führt einen Neustart durch
    </desc>
  </effect>
</op>
```

Abbildung 4.25: Definition von Seiteneffekten.

```
<op id="init">
  <title>init</title>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#init" xlink:role="global.init"/>
    <node xlink:href="werk1.xml#neustart" xlink:role="werk1.neustart"/>
    <node xlink:href="werk2.xml#reboot" xlink:role="werk2.reboot"/>

    <dep xlink:from="werk1.neustart" xlink:to="global.init"
     completion="required"/>
    <dep xlink:from="werk2.reboot" xlink:to="global.init"
     completion="required"/>
  </graph>
</op>
```

Abbildung 4.26: Modellierung von Abhängigkeiten zwischen Funktionen.

```
<node xlink:href="#m2inch.m" xlink:role="m2inch.mx" instance="x"/>
<node xlink:href="#m2inch.inch" xlink:role="m2inch.inchx" instance="x"/>
<node xlink:href="#m2inch.m" xlink:role="m2inch.my" instance="y"/>
<node xlink:href="#m2inch.inch" xlink:role="m2inch.inchy" instance="y"/>
<node xlink:href="#m2inch.m" xlink:role="m2inch.mz" instance="z"/>
<node xlink:href="#m2inch.inch" xlink:role="m2inch.inchz" instance="z"/>
```

Abbildung 4.27: Mehrfache Instantiierung von Funktionen.

Der Wert `fix:target` für das `instance` Attribut ist reserviert für die Benutzung im Rahmen der Modellierung von globalen Attributen (Abschnitt 4.10.5) und Quellfunktions-Wrappern (Abschnitt 4.10.12). Ansonsten kann die Wahl des Werts des `instance` Attributs frei erfolgen, er wird an keiner anderen Stelle verwendet.

4.10.4 Textuelle Beschreibung

Die in Abschnitt 3.1.8 beschriebene textuelle Notation von Aktionen wird in FIX durch das `<todo>` Element realisiert.

`<todo>`

`<todo>` ist eine lokale Ressource des durch `<graph>` repräsentierten Extended Links. So kann es durch `<dep>` Elemente mit anderen `<node>` oder `<todo>` Elementen verknüpft werden.

Bei einer eventuellen Konvertierung des FIX Dokuments in eine Programmiersprache wird der Inhalt dieser Elemente in Kommentaren an den entsprechenden Stellen des Quellcodes (vorgegeben durch die Abhängigkeiten zu anderen `<node>` oder `<todo>` Elementen) gesetzt.

```
<!ELEMENT todo                (title, desc*)>
<!ATTLIST todo
          xlink:type          (resource) #FIXED "resource"
          xlink:role          %Name; #REQUIRED
          coords              %coords; #IMPLIED>
```

`xlink:type` ist festgesetzt auf den Wert `resource`, um dieses Element als eine lokale Ressource des Extended Links zu markieren.

`xlink:role` siehe Abschnitt 4.10.1.

`coords` siehe Abschnitt 4.10.1.

Abbildung 4.28 zeigt die Modellierung des in Abbildung 3.10 auf Seite 27 dargestellten Abhängigkeitsgraphen der globalen Funktion `get_color()`.

4.10.5 Globale Attribute

Die Modellierung von globalen Attributen erfolgt mit FIX analog zu dem Beispiel aus Abbildung 3.14 (Seite 31). Innerhalb der Definition eines globalen Attributs durch das `<attr>` Element müssen zwei `<graph>` Elemente vorhanden sein, jeweils eins für den lesenden und den schreibenden Zugriff. Das in der Abbildung 3.14b zweite (umrandete) Auftreten des globalen Attributs wird in FIX durch das Attribut `instance="fix:target"` gekennzeichnet.

Die Modellierung des Beispiels von Abbildung 3.14 in FIX illustriert Abbildung 4.29.

```
<graph>
  <node xlink:href="#get_color.no" xlink:role="get_color.no"/>
  <node xlink:href="#get_color.return" xlink:role="get_color.return"/>
  <node xlink:href="werk1.xml#get_part.partno"
   xlink:role="get_part.partno"/>
  <node xlink:href="werk1.xml#get_part.color"
   xlink:role="get_part.color"/>
  <todo xlink:role="color2string">
    <title>color2string</title>
    <desc xml:lang="de">
      color muß in den Namen der
      Farbe umgewandelt werden.
      Es gilt dabei dei Zuordnung
       0   rot
       1   grün
       2   gelb
       3   blau
    </desc>
  </todo>

  <dep xlink:from="get_color.no" xlink:to="get_part.partno"/>
  <dep xlink:from="get_part.color" xlink:to="color2string"/>
  <dep xlink:from="color2string" xlink:to="get_color.return"/>
</graph>
```

Abbildung 4.28: Textuelle Beschreibung.

```
<attr id="currency">
  <title>currency</title>
  <type xlink:href="#currency_t"/>
  <graph xlink:role="read">
    <node xlink:href="#currency" xlink:role="currency"/>
    <node xlink:href="lieferant.xml#currency"
     xlink:role="lieferant.currency"/>
    <dep xlink:from="lieferant.currency" xlink:to="currency"/>
  </graph>
  <graph xlink:role="write">
    <node xlink:href="#currency" xlink:role="currency"
     instance="fix:target"/>
    <node xlink:href="#currency" xlink:role="currencysource"/>
    <node xlink:href="lieferant.xml#currency"
     xlink:role="lieferant.currency"/>
    <dep xlink:from="currencysource" xlink:to="lieferant.currency"/>
    <dep xlink:from="lieferant.currency" xlink:to="currency"/>
  </graph>
</attr>
```

Abbildung 4.29: Globale Attribute.

4.10.6 Objekte als Parameter

Die in Abschnitt 3.2.3 beschriebene Referenzierung von Methoden innerhalb von instantiierten Objekten wird in FIX durch spezielle FIXPointer-Achsen ermöglicht (siehe auch Abschnitt 4.12). Mit Hilfe der `member` Achse, die auch zur Referenzierung von Komponenten einer Struktur dient, kann innerhalb eines FIXPointers auf Attribute, Methoden und Statemembers eines Objekts verwiesen werden. Die `parameter` Achse kann zusätzlich zur Referenzierung eines Parameters der durch die vorangegangene Achse spezifizierten Methode benutzt werden. Ein XPointer verweist auf ein bestimmtest Objekt, in dem er auf die Stelle verweist, an der das Objekt erzeugt wurde. Dies kann entweder der Rückgabewert einer Funktion sein oder aber eine Factory. Der XPointer

```
id("factory")/member::method1
```

verweist auf die Methode `method1()` des von `factory()` erzeugten Objekts, während

```
id("factory")/member::method1/parameter::value
```

auf deren Parameter `value` verweist.

Abbildung 4.30 zeigt die Modellierung des Beispiels aus Abbildung 3.15 in FIX.

```
<graph>
  <node xlink:href="#global.param" xlink:role="global.param"/>
  <node xlink:href="#global.return" xlink:role="global.return"/>

  <node xlink:href="local.xml#factory" xlink:role="factory"/>
  <node xlink:href="local.xml#fixpointer(id("factory")/member::method1)"
   xlink:role="factory.method1"/>
  <node xlink:href="local.xml#fixpointer(id("factory")/member::method1/
   parameter::method1.value)" xlink:role="factory.method1.value"/>

  <node xlink:href="local.xml#factory2" xlink:role="factory2"/>
  <node xlink:href="local.xml#fixpointer(id("factory2")/member::convert)"
   xlink:role="factory2.convert"/>
  <node xlink:href="local.xml#fixpointer(id("factory2")/member::convert/
   parameter::convert.object)" xlink:role="factory2.convert.object"/>
  <node xlink:href="local.xml#fixpointer(id("factory2")/member::convert/
   parameter::convert.return)" xlink:role="factory2.convert.return"/>

  <dep xlink:from="global.param" xlink:to="factory.method1.value"/>
  <dep xlink:from="factory" xlink:to="factory2.convert.object"/>
  <dep xlink:from="factory.method1" xlink:to="factory2.convert"/>
  <dep xlink:from="factory.method1.return" xlink:to="global.return"/>
</graph>
```

Abbildung 4.30: Integration von Objekten.

4.10.7 Bedingte Ausführung

Gemäß der vorgeschlagenen Modellierung von Bedingungen als Abhängigkeitsrelation einer Codefunktion von einer Bedingungsfunktion (siehe Abschnitt 3.3.2), sind innerhalb von FIX ebenfalls keine weitergehenden Sprachkonstrukte zur Modellierung von Bedingungen notwendig. Zur Realisierung häufig vorkommender Bedingungen wurden in der Standardbibliothek Funktionen wie z.B. `eval()`, `evalseq()`, `istrue()` oder `isfalse()` definiert (siehe Abschnitt 4.13).

4.10.8 Schleifen

Schleifen werden in FIX durch zwei Funktionen realisiert, die Bedingungsfunktion und die Codefunktion, welche durch ein `<dep>` Element mit gesetztem `invocation="cyclic"` Attribut verknüpft sind. Das `invocation="cyclic"` Attribut entspricht dabei dem Doppelpfeil der Abbildungen in Abschnitt 3.3.2.

Ein solches `<dep>` Element verbindet dabei im allgemeinen Fall zwei `<node>` Elemente, die jeweils auf Funktionen verweisen. Um die beschriebenen Sonderfälle zu realisieren ist es auch möglich, daß solche `<dep>` Elemente einen Array oder eine Sequenz mit einem Parameter oder umgekehrt verbinden. Im Zusammenhang damit werden oft die in der Standardbibliothek vorhandenen Funktionen zur Listenverarbeitung benötigt.

4.10.9 Steuerung paralleler Ausführung

Für den Fall, daß mehrere `<dep>` Elemente auf denselben Zielknoten verweisen, ist die Auflösung einer dieser Abhängigkeiten ausreichend. Ist die Auflösung aller zu einem Knoten führenden Abhängigkeiten notwendig, müssen diese als solche gekennzeichnet werden. Zu diesem Zweck stehen die Attribute `completion` und `priority` des `<dep>` Elements zur Verfügung. Die möglichen Werte für `completion` haben dabei folgende Bedeutung:

`sufficient` — Wird die durch dieses `<dep>` Element definierte Abhängigkeitsrelation bei der Ausführung erfüllt, muß keine weitere, zu dem unter `xlink:to` spezifizierten Knoten führende Abhängigkeitsrelation erfüllt werden. Die Ausführung mehrerer `sufficient` (oder `required`) Abhängigkeitsrelationen sollte — soweit möglich — parallel erfolgen. `sufficient` ist der Default-Wert.

`required` — Die Abhängigkeitsrelation muß bei der Ausführung erfüllt werden. Kann sie nicht erfüllt werden, bricht die Ausführung des Graphen ab.

`optional` — Es wird erst versucht, diese Abhängigkeitsrelation zu erfüllen, wenn alle als `sufficient` markierten, zum selben Zielknoten führenden Abhängigkeitsrelationen nicht erfüllt werden konnten. `optional` Abhängigkeitsrelationen werden in der Reihenfolge ihrer Priorität ausgeführt.

Grundsätzlich gilt: Es müssen alle der als `required` markierten Abhängigkeiten und mindestens eine der als `sufficient` oder `optional` markierten Abhängigkeiten erfüllt werden.

Sind mehrere Abhängigkeitsrelationen als `optional` markiert, wird deren `priority` Attribut ausgewertet. Es wird versucht, die optionalen Abhängigkeitsrelationen in der Reihenfolge ihrer Priorität (je

größer der Wert des `priority` Attributs, desto höher die Priorität) nacheinander aufzulösen. Sobald eine Funktion erfolgreich ausgeführt werden konnte, wird nicht mehr versucht, weitere als `optional` markierte Abhängigkeitsrelationen aufzulösen.

Als Beispiel dient die globale Funktion `get_part()`. Zu deren Realisierung (siehe auch Abbildung 3.25 auf Seite 39) muß auf die Funktion `get_price()` des Lieferantensystems zurückgegriffen werden. Liefert diese keinen Wert zurück (etwa weil das referenzierte Bauteil nicht von diesem Lieferanten bezogen werden kann), soll `get_part()` den Betrag 0 als Preis zurückgeben. Dies kann durch das in Abbildung 4.31 dargestellt FIX Fragment realisiert werden.

```
<graph>
  <node xlink:href="#get_part.no" xlink:role="global.no"/>
  <node xlink:href="#get_part.name" xlink:role="global.name"/>
  <node xlink:href="#get_part.color" xlink:role="global.color"/>
  <node xlink:href="#get_part.price" xlink:role="global.price"/>
  <node xlink:href="werk1.xml#get_part.partno"
   xlink:role="werk1.partno"/>
  <node xlink:href="werk1.xml#get_part.name"
   xlink:role="werk1.name"/>
  <node xlink:href="werk1.xml#get_part.color"
   xlink:role="werk1.color"/>
  <node xlink:href="helper.xml#col2str.color"
   xlink:role="col2str.color"/>
  <node xlink:href="helper.xml#col2str.return"
   xlink:role="col2str.return"/>
  <node xlink:href="lieferant.xml#get_price.no"
   xlink:role="lieferant.no"/>
  <node xlink:href="lieferant.xml#get_price.return"
   xlink:role="lieferant.price"/>
  <node xlink:href="helper.xml#null" xlink:role="null"/>

  <dep xlink:from="global.no" xlink:to="werk1.partno"/>
  <dep xlink:from="global.no" xlink:to="lieferant.no"/>
  <dep xlink:from="werk1.name" xlink:to="global.name"/>
  <dep xlink:from="werk1.color" xlink:to="col2str.color"/>
  <dep xlink:from="col2str.return" xlink:to="global.color"/>
  <dep xlink:from="lieferant.price" xlink:to="global.price"/>
  <dep xlink:from="null" xlink:to="global.price"
   completion="optional"/>
</graph>
```

Abbildung 4.31: Redundante Ausführung.

4.10.10 Werfen von Exceptions

Die in Abschnitt 3.4.2 beschriebene Zuordnung von Exceptions zu Quellfunktionsaufrufen erfolgt in FIX ebenfalls durch ein <dep> Element. Um diese Verknüpfung von normalen Abhängigkeitsrelationen unterscheiden zu können, wird es durch ein `xlink:role="exception"` Attribut gekennzeichnet.

4.10.11 Kompensationen

Die Realisierung von Kompensationen innerhalb eines Abhängigkeitsgraphen erfolgt in FIX wie in Abschnitt 3.4.4 beschrieben. Die sogenannten Fehlerabhängigkeitsrelationen werden durch <dep> Elemente mit einem `xlink:role="compensation"` Attribut modelliert. Die FIX Modellierung von Abbildung 3.30 (Seite 44) illustriert Abbildung 4.32.

```
<!-- "normale" Abhängigkeiten -->

<dep xlink:from="add_part.name" xlink:to="werk2.add_part.name"/>
<dep xlink:from="add_part.name" xlink:to="werk1.add_part.partno"/>
<dep xlink:from="add_part.color" xlink:to="werk1.add_part.color"/>
<dep xlink:from="werk2.add_part.return" xlink:to="strtoint.str"/>
<dep xlink:from="strtoint.return" xlink:to="add_part.return"/>
<dep xlink:from="strtoint.return" xlink:to="werk1.add_part.partno"/>
<dep xlink:from="werk1.add_part" xlink:to="add_part"/>

<!-- Fehlerabhängigkeitsrelationen -->

<dep xlink:from="werk2.add_part" xlink:to="werk2.del_part"
 xlink:role="compensation"/>
<dep xlink:from="werk2.add_part" xlink:to="null2"
 xlink:role="compensation"/>
<dep xlink:from="werk2.add_part.return"
 xlink:to="werk2.del_part.no"/>
<dep xlink:from="werk2.del_part" xlink:to="null1"
 completion="required"/>

<dep xlink:from="werk1.add_part" xlink:to="werk1.del_part"
 xlink:role="compensation"/>
<dep xlink:from="werk1.add_part" xlink:to="null2"
 xlink:role="compensation"/>
<dep xlink:from="strtoint.return"
 xlink:to="werk1.del_part.partno"/>
<dep xlink:from="werk1.del_part" xlink:to="null"
 completion="required"/>

<dep xlink:from="null1" xlink:to="add_part.return"/>
<dep xlink:from="null2" xlink:to="add_part.return"
 completion="optional"/>
```

Abbildung 4.32: Kompensationen in FIX.

4.10.12 Wrapper für Quellfunktionen

Die in Abschnitt 3.4.3 beschriebenen Abhängigkeitsgraphen für Quellfunktionen zur Nachbildung von Exceptions werden in FIX durch <graph> Elemente mit gesetztem `type="wrapper"` Attribut realisiert. Abbildung 4.33 zeigt die Modellierung der Nachbildung von Exceptions für die Funktion `del_part()` von Werk 2 aus Abbildung 3.27.

Auf dieselbe Weise ist innerhalb FIX die Realisierung von Default-Kompensationen möglich. Default-Kompensationen sind Kompensationen, die einer bestimmten Quellfunktion zugeordnet werden. Tritt

```
<graph type="wrapper">
  <node xlink:href="#del_part" xlink:role="del_part.target"
   instance="fix:target"/>
  <node xlink:href="#del_part.no" xlink:role="del_part.no.target"
   instance="fix:target"/>
  <node xlink:href="#del_part.return"
   xlink:role="del_part.return.target" instance="fix:target"/>
  <node xlink:href="#del_part.no" xlink:role="del_part.no"/>
  <node xlink:href="#del_part.return" xlink:role="del_part.return"/>
  <node xlink:href="stdlib.xml#istrue" xlink:role="istrue"/>
  <node xlink:href="stdlib.xml#istrue.boolean"
   xlink:role="istrue.boolean"/>
  <node xlink:href="global.xml#ENOTDELETED" xlink:role="ENOTDELETED"/>

  <dep xlink:from="del_part.no.target" xlink:to="del_part.no"/>
  <dep xlink:from="del_part.return" xlink:to="del_part.return.target"/>
  <dep xlink:from="del_part.return" xlink:to="istrue.boolean"/>
  <dep xlink:from="istrue" xlink:to="del_part"/>
  <dep xlink:from="istrue" xlink:to="ENOTDELETED"
   xlink:role="exception"/>
</graph>
```

Abbildung 4.33: Nachbildung von Exceptions.

während der Ausführung eines Abhängigkeitsgraphen ein Fehler auf, werden zusätzlich zu den in dem Graph modellierten Kompensationen alle Default-Kompensationen der bisher aufgerufenen Quellfunktionen ausgeführt.

Nach der korrekten Ausführung des Wrappers speichert das System den momentanen Zustand des Wrappers so lange, bis die Ausführung des globalen Abhängigkeitsgraphen beendet wurde. Tritt während der weiteren Ausführung des Abhängigkeitsgraphen ein Fehler auf, werden die Fehlerkanten des Wrappers aktiv und es wird erneut versucht, die Abhängigkeiten innerhalb des Wrappers zu erfüllen. So werden die zu der Quellfunktion gehörenden Kompensationsaktionen ausgeführt.

Die Zusammenfassung der Nachbildung von Exceptions und Default-Kompensationen in einem Abhängigkeitsgraphen ist ebenfalls möglich. Die Abbildungen 4.34 und 4.35 zeigen einen solchen Graphen für die Funktion `add_part()` aus Werk 2.

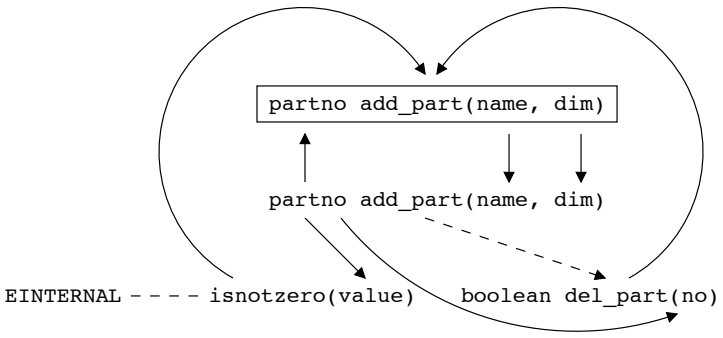


Abbildung 4.34: Defaultkompensation (I).

```
<graph type="wrapper">
  <node xlink:href="#add_part" xlink:role="add_part.target"
   instance="fix:target"/>
  <node xlink:href="#add_part.name" xlink:role="add_part.name.target"
   instance="fix:target"/>
  <node xlink:href="#add_part.dim" xlink:role="add_part.dim.target"
   instance="fix:target"/>
  <node xlink:href="#add_part.return"
   xlink:role="add_part.return.target" instance="fix:target"/>

  <node xlink:href="#add_part" xlink:role="add_part"/>
  <node xlink:href="#add_part.name" xlink:role="add_part.name"/>
  <node xlink:href="#add_part.dim" xlink:role="add_part.dim"/>
  <node xlink:href="#add_part.return" xlink:role="add_part.return"/>

  <node xlink:href="#del_part" xlink:role="del_part"/>
  <node xlink:href="#del_part.no" xlink:role="del_part.no"/>

  <node xlink:href="stdlib.xml#isnotzero" xlink:role="isnotzero"/>
  <node xlink:href="stdlib.xml#isnotzero.value"
   xlink:role="isnotzero.value"/>
  <node xlink:href="global.xml#EINTERNAL" xlink:role="EINTERNAL"/>

  <dep xlink:from="add_part.name.target" xlink:to="add_part.name"/>
  <dep xlink:from="add_part.dim.target" xlink:to="add_part.dim"/>
  <dep xlink:from="add_part.return" xlink:to="add_part.return.target"/>

  <dep xlink:from="add_part.return" xlink:to="del_part.no"/>
  <dep xlink:from="add_part" xlink:to="del_part"
   xlink:role="compensation"/>
  <dep xlink:from="del_part" xlink:to="add_part.target"/>

  <dep xlink:from="add_part.return" xlink:to="isnotzero.value"/>
  <dep xlink:from="isnotzero" xlink:to="add_part.target"/>
  <dep xlink:from="isnotzero" xlink:to="EINTERNAL"
   xlink:role="exception"/>
</graph>
```

Abbildung 4.35: Defaultkompensation (II).

Normalerweise werden sowohl Default-Kompensationen als auch explizit innerhalb eines Abhängigkeitsgraphen modellierte Kompensationen ausgeführt. Wird innerhalb eines <node> Elements das compensation Attribut auf den Wert false gesetzt, wird die Ausführung der Default-Kompensation unterbunden.

4.10.13 Externe Graphen

Die Modellierung der Abhängigkeitsbeschreibung muß nicht innerhalb der Funktions- oder Attributdefinition erfolgen, sondern kann sich auch in einem externen Dokument befinden. Durch <ext> Elemente innerhalb von <graph> Elementen können extern (innerhalb eines <mapping> Elements) definierte Graphen referenziert werden.

<ext>

Das <ext> Element dient zum Verweis auf externe Graphen. Ein <ext> Element kann nur innerhalb eines <graph> Elements mit einem xlink:role="xlink:external-linkset" Attribut vorkommen. Es darf nur ein <ext> Element innerhalb eines <graph> Elements vorkommen.

```
<!ELEMENT ext                  EMPTY>
<!ATTLIST ext
          xlink:type           (locator) #FIXED "locator"
          xlink:href           %uri; #REQUIRED>
```

xlink:type — ist festgesetzt auf den Wert locator, um dieses Element als Locator Element eines Extended Links zu markieren.

xlink:href — verweist auf einen externen Graphen.

<mapping>

Das <mapping> Element dient als Container für externe Graphen. <mapping> ist ein Wurzelelement in dem beliebig viele <graph> Elemente vorkommen können. Abbildung 4.36 veranschaulicht die Verwendung der <ext> und <mapping> Elemente.

```
<!ELEMENT mapping              (graph)*>
```

4.11 Semantische Attribute

Semantische Attribute (siehe auch Abschnitt 3.7.2) werden in FIX durch das semantic Attribut modelliert. Dies kann sowohl ein <op> Element als auch ein <node> Element näher beschreiben. Zeigt ein <node> Element, welches ein semantic Attribut besitzt, auf ein <op> Element, das ebenfalls ein semantic Attribut besitzt, so gilt der Wert des <node> Elements.

Die von FIX definierten möglichen Werte für das semantic Attribut zeigt Tabelle 4.2. Die Implementierung des semantic Attributs ist für FIX kompatible Software optional. Wird das semantic Attribut

```
Datei "system.xml":

<op>
  <title>example</title>
  ...
  <graph xlink:role="xlink:external-linkset">
    <ext xlink:href="extgraphs.xml#example"/>
  </graph>
</op>

Datei "extgraphs.xml":

<mapping>
  <graph id="example">
    ...
  </graph>
</mapping>
```

Abbildung 4.36: Externe Graphen.

durch ein bestimmtes Programm ausgewertet, so müssen die in Tabelle 4.2 definierten Werte auf jeden Fall unterstützt werden. Die dargestellten Werte sind innerhalb des FIX Namensraums definiert. Sie müssen also mit den für das `<system>` Element vorgeschriebenen Namensraumdeklarationen sowohl mit als auch ohne vorangestellte `fix:` Präfix erkannt werden.

Wert	Bedeutung
`convert`	Konvertierungsfunktion
`condition`	Bedingung
`read`	lesender Zugriff
`add`	Hinzufügen eines Elements
`delete`	Löschen eines Elements
`update`	Verändern eines Elements
`other`	ein nicht-Standardfall
`undefined`	keine Semantik angegeben

Tabelle 4.2: Mögliche Werte des `semantic` Attributs.

Es ist möglich, daß bestimmte FIX Software zusätzliche Werte für das `semantic` Attribut definiert. Diese neuen Werte müssen innerhalb eines separaten Namensraums definiert werden, damit die Eindeutigkeit gewahrt bleibt. Der benutzte Namensraum muß dabei entsprechend im `<system>` Element deklariert werden (siehe auch Abschnitt 4.3).

Die Unterstützung in anderen Namensräumen definierter Werte für das `semantic` Attribut ist optional. Enthält ein Dokument einen Wert des `semantic` Attributs, der in einem Namensraum deklariert wurde, welche von der Software nicht unterstützt wird, muß diese statt dessen den Wert `fix:other` annehmen.

4.12 FIXPointer

FIXPointer erweitern die normalen XPointer aus [XPOINTER] um die drei neuen Achsen (axis) `member`, `element` und `parameter`. Mit Hilfe dieser neuen Achsen können Teile von Parametern durch `<node>` Elemente referenziert werden. Um einen FIXPointer von einem normalen XPointer zu unterscheiden wird er durch ein umschließendes `fixpointer()` gekennzeichnet.

element

Die `element` Achse dient zur Referenzierung eines bestimmten Elements innerhalb eines Arrays, einer Sequenz oder eines Strings. Als Node-Test muß die Nummer des Elements angegeben werden, wobei die Numerierung bei 0 beginnt. Der FIXPointer aus Abbildung 4.37 referenziert beispielsweise das fünfte Element des Rückgabewerts der Funktion `all_parts()` von Werk 2:

```
<node xlink:href='werk2.xml#fixpointer(id("all_parts.return")/element::5'
 xlink:role="element5"/>
```

Abbildung 4.37: `element` Achse.

member

Durch die `member` Achse können einzelne Komponenten von Strukturen referenziert werden. Des weiteren kann die Achse zur Referenzierung von Attributen, Operationen oder Statemembers eines Value Types dienen. Als Node-Test muß dabei die ID der gewünschten Komponente des Parameters angegeben werden. In Abbildung 4.38 referenziert das `<node>` Element die Komponente `x` des Rückgabeparameters `dim` der Funktion `get_part()` von Werk 2:

```
<node
 xlink:href='werk2.xml#fixpointer(id("get_part.dim")/
 member::dimension.x)'
 xlink:role="memberx"/>
```

Abbildung 4.38: `member` Achse.

parameter

Diese Achse dient zur Referenzierung von Parametern einer Methode. Sie findet bei der in Abschnitt 4.10.6 beschriebenen Integration objektorientierter Systeme Verwendung. Der FIXPointer aus Abbildung 4.39 referenziert das `<param>` Element mit der ID `convert.object` innerhalb des `<op>` Elements mit der ID `convert` des durch die Factory `factory2` erzeugten Objekts.

```
<node xlink:href="local.xml#fixpointer(id("factory2")/
 member::convert/parameter::convert.object)"
 xlink:role="factory2.convert.object"/>
```

Abbildung 4.39: `parameter` Achse.

Gemeinsame Verwendung von XPointer und FIXPointer Schemata

Um die Verarbeitung von FIX Dokumenten auch durch generische XLink Software zu ermöglichen, sollte bei der Verwendung von FIXPointern immer zusätzlich auch ein XPointer angegeben werden. Der XPointer sollte dabei auf den Parameter als Ganzes zeigen. Abbildung 4.40 zeigt das Beispiel aus Abbildung 4.38 mit mit einem zusätzlichen XPointer.

```
<node
 xlink:href='werk2.xml#fixpointer(id("get_part.dim")/
 member::dimsion.x)xpointer(id("get_part.dim"))'
 xlink:role="memberx"/>
```

Abbildung 4.40: Gemeinsame Verwendung von XPointer und FIXPointer Schemata.

4.13 Standardbibliothek

Die folgenden Funktionen müssen von einer FIX-konformen Software in der sogenannten Standardbibliothek implementiert werden. Innerhalb von Abhängigkeitsbeschreibungen können diese innerhalb des Dokuments `http://dcx.com/fix/stdlib.xml` referenziert werden. Die IDL Definition und das FIX Dokument `stdlib.xml` befinden sich ebenfalls in Anhang 3.7.1.

Im Rahmen der Weiterentwicklung von FIX sollten vor allem die innerhalb von `stdlib.xml` definierten Funktionen wesentlich erweitert werden.

`eval(format, value)` akzeptiert als Parameter einen Formatstring und einen Wert. Der Formatstring kann dabei ein Platzhalter in der Syntax eines C `printf()` Formatstrings beinhalten. Nach dem Aufruf der Funktion wird der Platzhalter durch den übergebenen Wert ersetzt. Der sich ergebende Ausdruck wird ausgewertet. Ergibt die Auswertung einen negativen Wahrheitswert, wirft die Funktion die Exception EFALSE, bei einem positiven Wert kehrt die Funktion ohne Exception zurück. Der folgende Aufruf wird auf jeden Fall eine Exception auslösen:

```
eval("%u > 10", 5)
```

`evalseq(format, values)` ist eine Erweiterung der Funktion `eval()` auf mehrere Parameter. Als zweiter Parameter muß eine Sequenz übergeben werden. Die Anzahl der Elemente in der Sequenz muß mit der Anzahl der Platzhalter

	im Formatstring übereinstimmen. Der folgende Aufruf vergleicht die beiden Parameter und gibt bei Ungleichheit eine Exception zurück. `eval("%d = %d", {a, b})`
`istrue(value)`	Wird dieser Funktion der Wert `TRUE` übergeben, führt sie keinerlei Operationen durch. Wird der Wert `FALSE` übergeben, kehrt sie mit der Exception `EFALSE` zurück.
`isfalse(value)`	Diese Funktion verhält sich komplementär zu `istrue()`, entspricht also `istrue(!value)`.
`isequal(a, b)`	vergleicht die beiden übergebenen Parameter. Stimmen diese nicht überein, kehrt `isequal()` mit der Exception `EFALSE` zurück.
`isnotequal(a, b)`	vergleicht ebenso die beiden übergebenen Parameter, kehrt aber mit einer Exception zurück, falls die beiden Parameter übereinstimmen.
`iszero(value)`	löst die Exception `EFALSE` aus, falls der übergebene Wert ungleich 0 ist.
`isnotzero(value)`	löst eine Exception aus, wenn der übergebene Wert gleich 0 ist.
`loop(max, count)`	realisiert eine einfache Schleife. Die Funktion erhöht den in `count` übergebenen Wert um eins. Ist der neue Wert größer als der in `max` übergebene, wird die Exception `EFALSE` ausgelöst, andernfalls wird der neu berechnete Wert wieder in `count` zurückgegeben.
`incr(value)`	erhöht den übergebenen Wert um 1.
`decr(value)`	erniedrigt den übergebenen Wert um 1.
`long strtoint(str)`	konvertiert den übergebenen String soweit möglich in einen Integer.
`string inttostr(int)`	konvertiert den übergebenen Integer in einen String.

4.14 Konvertierungsmöglichkeiten

Eine Möglichkeit der Verarbeitung von FIX Dokumenten ist die Konvertierung in andere Formate. Denkbar ist die beispielsweise die Erzeugung von IDL Definitionen aus FIX Dokumenten oder die Konvertierung in HTML Dokumente zur Generierung einer Online Dokumentation. Durch die Verwendung von XML als Grundlage von FIX bietet sich die Benutzung bereits existierender Technologien zu diesem Zweck an.

Die XSL Transformations [XSLT] sind ein Teil der Extensible Stylesheet Language [XSL], einer Sprache zur Definition sogenannter Stylesheets, also Darstellungsanweisungen für XML Dokumente. Mit Hilfe von XSLT ist die Beschreibung von Regeln zur Umwandlung von XML Dokumenten in andere XML Dokumente oder reine ASCII Dokumente möglich.

Das in Anhang D enthaltene XSLT Stylesheet realisiert beispielhaft eine Konvertierung von FIX Dokumenten in IDL Definitionen. Das Stylesheet konvertiert die in einem FIX Dokument definierte

Schnittstelle in die IDL Syntax. `<const>` Elemente werden dabei nur konvertiert, falls ein zugehöriges `<expr>` Element existiert und der enthaltene Wert im Format `text/plain` oder `text/x-idl` ist. Abhängigkeitsbeschreibungen innerhalb von `<graph>` Elementen werden ignoriert.

Kapitel 5

Ausführungsmodell

Nachdem in den vorangehenden Kapiteln ein Beschreibungsmodell und darauf aufbauend die Beschreibungssprache FIX vorgestellt wurden, sollen in diesem Kapitel die Anforderungen an ein mögliches Ausführungsmodell dargestellt werden. Weiterhin soll anhand des IBM MQ Workflow Management Systems knapp beschrieben werden, inwieweit ein Workflow Management System als Ausführungsumgebung benutzt werden kann.

5.1 Einleitung

Die im letzten Kapitel vorgestellte Sprache FIX wurde als reine Beschreibungssprache modelliert. Um die in einem FIX Dokument beschriebenen Abhängigkeiten zu verarbeiten, wird eine entsprechende Ausführungsumgebung benötigt. Es ergeben sich dabei folgende Anforderungen an diese Ausführungsumgebung.

Konstrukte: Das verwendete System muß eine geeignete Repräsentation der grundlegenden, in FIX verwendeten Konstrukte wie Funktionen und Parameter bieten.

Datentypen: Neben den einfachen Datentypen müssen auch die in FIX definierten Vorlagen- bzw. komplexen Datentypen unterstützt werden.

Abhängigkeiten: Die Ausführungsumgebung muß aus einem FIX Abhängigkeitsgraph eine Aufrufreihenfolge der lokalen Funktionen generieren können. Sie muß dabei insbesondere auch die Konstrukte für bedingte Anweisungen und Schleifen unterstützen und eine parallele Ausführung ermöglichen können.

Schnittstellen: Die Ausführungsumgebung muß Schnittstellen zu den lokalen Systemen besitzen. Es sollte also eine Unterstützung von Systemen wie CORBA oder Java RMI vorhanden sein. Weiterhin muß das System die generierten Zielfunktionen exportieren. Aus diesem Grund wird zusätzlich ein ORB benötigt.

GUI: Zusätzlich zur reinen Ausführungsumgebung, sollte das verwendete System den Benutzer bei der Erstellung der Abhängigkeitsgraphen durch ein graphisches Werkzeug unterstützen.

Transaktionen: Sollen Transaktionen, wie in Abschnitt 3.5 beschrieben, verwirklicht werden, muß das System selbst als Ressourcenmanager an einer verteilten Transaktion teilnehmen können.

In [Vogt, 1999, Abschnitt 6.2] wurde als Beispiel eine Ausführungsumgebung auf der Basis von `make` vorgestellt. Im Rahmen dieser Arbeit soll evaluiert werden, inwieweit das MQ Series Workflow Management System von IBM [MQWORKFLOW1] als Ausführungsengine genutzt werden kann.

5.2 Workflow Management Systeme

Unter einem Workflow oder auch (Geschäfts-) Prozeß (Business Process) versteht man eine endliche Folge von Aktivitäten. Die Aktivitäten sind untereinander durch den sogenannten Kontrollfluß verknüpft, so daß sich eine Ausführungsreihenfolge ergibt. Die Folge wird dabei durch ein Ereignis ausgelöst und kann ganz oder teilweise (ausgelöst durch Bedingungen), sequentiell oder parallel ablaufen. Das Beispiel in Abbildung 5.1 (aus [Leymann, 2000, Abbildung 2.3]) zeigt einen typischen Geschäftsprozeß zur Genehmigung eines Kredits.

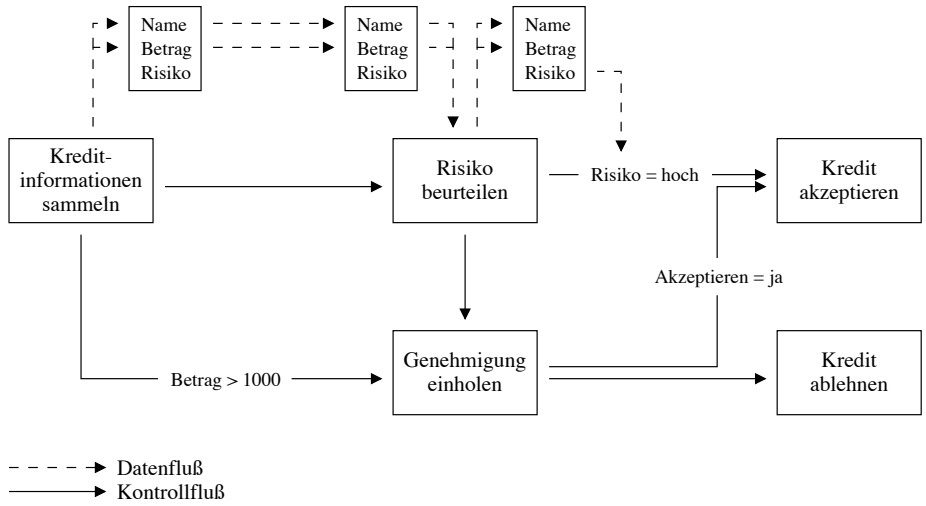


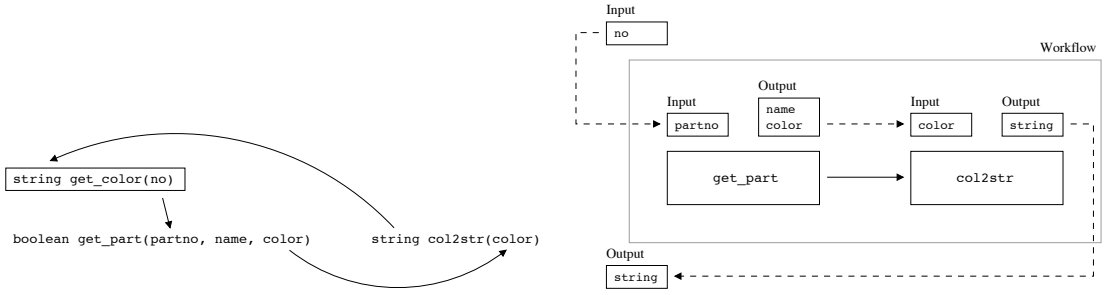
Abbildung 5.1: Geschäftsprozeß.

Den einzelnen Aktivitäten werden dabei sogenannte Input- bzw. Output-Container zugeordnet (in Abbildung 5.1 für die beiden Aktivitäten "Kreditinformationen sammeln" und "Risiko beurteilen" dargestellt), über die Daten von einer Aktivität zu einer anderen übergeben werden können.

Ein Workflow Management System (WfMS) ist ein Softwaresystem, daß den Benutzer bei allen bei der Modellierung, Simulation, Ausführung oder Steuerung eines Workflows auftretenden Aufgaben unterstützt.

Aktivitäten innerhalb eines Workflows können dabei entweder die Interaktion mit einem Benutzer erfordern oder automatisch ablaufen, indem sie z.B. einen entsprechenden Prozeß auf einem Server starten. Betrachtet man Prozesse, die vollständig aus automatisch ablaufenden Aktivitäten bestehen, werden einige Übereinstimmungen mit den besprochenen Abhängigkeitsrelationen deutlich. Abbildung 5.2 zeigt die Modellierung der globalen Funktion `get_part()` durch einen Abhängigkeitsgraphen und deren Umsetzung in einem Workflow.

Dabei entsprechen die einzelnen Aktivitäten den Funktionen innerhalb des Abhängigkeitsgraphen. Die Parameter und Rückgabewerte werden durch die Input- bzw. Output-Container der Aktivitäten repräsentiert. Während die Parameterabhängigkeiten des Abhängigkeitsgraphen dem Datenfluß des Workflows entsprechen, muß die sich durch den Abhängigkeitsgraphen implizit ergebende Ausführungsreihenfolge innerhalb des Workflows explizit durch den Kontrollfluß modelliert werden.



(a) Abhängigkeitsgraph (b) Workflow

Abbildung 5.2: Vergleich zwischen Abhängigkeitsgraphen und Workflows.

WfMSe erfüllen die grundlegenden Anforderungen an eine Ausführungsumgebung für Abhängigkeitsgraphen. Neben den beschriebenen Übereinstimmungen der verwendeten Konstrukte verfügen WfMSe über rudimentäre Mechanismen zum Starten von lokalen oder entfernten Prozessen. Mit der sogenannten Build-Time Komponente verfügen WfMSe ebenfalls über ein graphisches Werkzeug zur Erstellung des Workflows.

Für eine weiterführende Beschreibung von Workflows bzw. Workflow Management Systemen sei der interessierte Leser auf [Leymann, 2000] verwiesen.

5.3 MQ Workflow

MQ Workflow ist eine Weiterentwicklung des IBM Workflow Management Systems Flowmark, basierend auf dem Middleware Produkt MQ Series [MQSERIES] und der Datenbank DB2 [DB2]. MQ Workflow besteht aus den Build-Time und Run-Time Komponenten. Während für die Run-Time Komponente, also die eigentliche Ausführungsumgebung, Portierungen für eine große Anzahl verschiedener Plattformen, darunter gängige Unix Derivate und die Windows Plattform existieren, ist die Build-Time Komponente, also das graphische Werkzeug zur Erstellung von Workflows, nur unter Windows lauffähig.

MQ Workflow benutzt zur Spezifikation der Definitionen von Prozessen und dazu gehörenden weiteren Daten (Daten über beteiligte Server, Benutzer, etc.) die vom Vorgänger Flowmark übernommene Flowmark Definition Language (FDL) [MQWORKFLOW2], die nicht kompatibel zu der von der Workflow Management Coalition vorgeschlagenen Workflow Process Definition Language (WPDL) [INTERFACE1] ist. Es existieren neben der mitgelieferten Build-Time Komponente weitere Werkzeuge von Drittherstellern, die ebenfalls FDL Dokumente verarbeiten können.

5.4 Architektur

MQ Workflow stellt Schnittstellen in Form von APIs für ein breites Spektrum an Funktionen zur Verfügung [MQWORKFLOW3]. Es ist möglich, per API die Ausführung eines Prozesses zu beeinflussen, also z.B. ihn zu starten oder abzubrechen. Außerdem können Programme, die Aktivitäten implementieren, über ein API auf Informationen über den Zustand des Prozesses zugreifen oder diese verändern. So können diese z.B. den mit einer Aktivität betrauten Benutzer erfragen oder den Inhalt des zu der Aktivität gehörenden Input Container lesen und Ergebnisse in den Output Container schreiben. Weiterhin stehen APIs zur Administration und Überwachung des WfMS und der einzelnen Workflowprozesse zur Verfügung.

Die APIs stehen für mehrere Sprachen zur Verfügung, u.a. für C++ und Java. Zusätzlich ist eine auf XML-Nachrichten basierte Version des APIs vorhanden. MQ Workflow stellt seine Schnittstellen nicht über verteilte Objektdienste zur Verfügung, insbesondere implementiert es keinen CORBA ORB. Um die Schnittstelle eines durch die Integration entstandenen globalen Systems exportieren zu können, muß also zusätzlich ein ORB integriert werden, welcher die globale Schnittstelle exportiert. Eine ORB / WfMS Schnittstelle muß Funktionsaufrufe des ORBs in Aufrufe des MQ Workflow APIs zur Ausführung von Workflows umsetzen. Dazu besteht die Implementierung jeder von dem ORB angebotenen Funktion aus einem kurzen Wrapper, welcher die übergebenen Parameter per API in den Input Container des zugehörigen Workflows schreibt und diesen startet. Nach der Abarbeitung des Workflows wird die Rückgabe von dem Wrapper aus dem Output Container ausgelesen und an den aufrufenden Client zurückgegeben.

MQ Workflow kann im Rahmen einer Aktivität ein Programm ausführen oder eine Funktion einer Shared Library aufrufen. Dies kann sowohl lokal als auch auf einem entfernten Rechner erfolgen. Für den entfernten Aufruf verwendet MQ Workflow ein eigenes Protokoll. Dazu muß auf dem jeweiligen entfernten Rechner ein sogenannter Program Execution Agent installiert werden.

MQ Workflow unterstützt nicht die Benutzung von Schnittstellen, die über RPC- oder Objektdienste exportiert wurden. Um diese nutzen zu können, müssen zu diesem Zweck Quellsystemwrapper entwickelt werden, welche die folgenden Schritte ausführen:

1. Auslesen der Werte

   Der Wrapper kann die im Input Container vorhandenen Werte per API auslesen. Im Input Container müssen dabei neben den der Funktion zu übergebenden Eingabeparameter auch die Adresse des entfernten Systems und die Signatur der Funktion enthalten sein. Da die Einträge im Input Container die Form $Name = Wert$ besitzen, kann der Wrapper zusammen mit der Signatur der Funktion die im Input Container übergebenen Werte den Parametern der aufzurufenden Funktion zuordnen.

2. Aufruf der Funktion

   Zum Aufruf der entfernten Funktion muß zunächst ein Marshalling der Parameter erfolgen, anschließend erfolgt dem verwendeten Protokoll entsprechend der Aufruf der entfernten Funktion und die Übertragung der Parameter.

3. Rückgabe der Ergebnisse

   Die von dem entfernten System zurück erhaltenen Ausgabeparameter und der Rückgabewert werden von dem Wrapper ausgelesen und anschließend wiederum über API Funktionen in den Output Container zurückgeschrieben.

Auf dieselbe Weise müssen Quellsystemwrapper auch den Zugriff auf Quellsystemattribute behandeln können, falls das verwendete Quellsystem diese unterstützt. Dadurch, daß die Quellsystemwrapper auf alle benötigten Informationen (Adresse des Systems, Funktionssignatur, Parameterwerte) durch Auslesen des Input Containers zugreifen können, müssen diese nur einmal für jedes verwendete Kommunikationsprotokoll implementiert werden.

Abbildung 5.3 zeigt den gesammten Aufbau einer auf einem WfMS aufbauenden Ausführungsumgebung für FIX Dokumente.

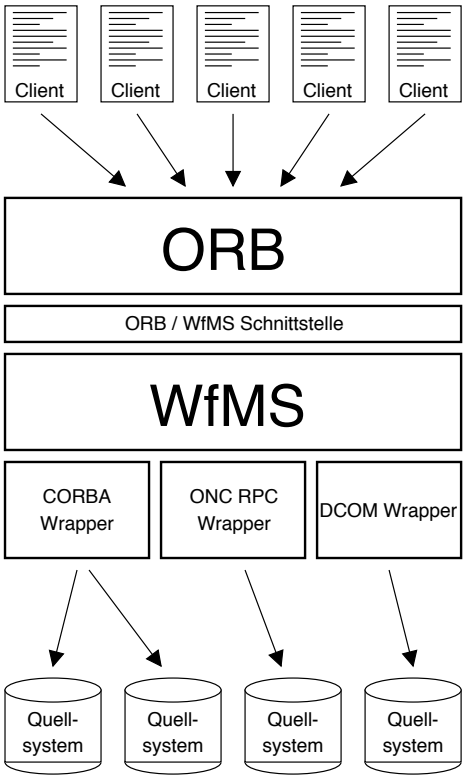


Abbildung 5.3: Ausführungsmodell.

5.5 Notwendige Arbeiten

Da ein FIX Dokument nicht von MQ Workflow direkt verarbeitet werden kann, muß es zuvor entsprechend umgewandelt werden. Dabei wird neben der Umwandlung in das von MQ Workflow verwendete FDL Format die Generierung weiterer Dateien (beispielsweise eine IDL Definition des Zielsystems) notwendig. Die im folgenden verwendeten Beispiele beziehen sich auf die Umsetzung der globalen Funktion `get_color()`. Das komplette Beispiel befindet sich in Anhang E.6.

ORB

Der ORB stellt die globale Schnittstelle zur Verfügung. Aus diesem Grund muß auch eine Definition der exportierten Schnittstelle vorhanden sein. Da in diesem Fall CORBA verwendet wird, ist

die Generierung einer Schnittstellendefinition in IDL aus dem vorhandenen FIX Dokument wie in Abschnitt 4.14 notwendig. Zusätzlich muß der notwendige Code für die globalen Wrapper erzeugt werden.

Quellwrapper

Die Quellwrapper müssen nicht für jedes FIX Dokument neu implementiert werden, sondern nur einmal für jedes Kommunikationsprotokoll (IIOP, RMI, etc.). Die notwendigen Daten über die aufzurufende Funktion (Adresse des Systems, Portnummer, Signatur der Funktion, Werte Parameter) kann der Wrapper aus dem Input Container der Aktivität auslesen.

Die einzelnen Wrapper müssen innerhalb des erzeugten FDL Dokuments durch Programmdefinitionen registriert werden. Abbildung 5.4 enthält den notwendigen FDL Code zur Registrierung eines CORBA Wrappers. Dabei unterdrückt `STYLE INVISIBLE` die Bildschirmausgabe während das Schlüsselwort `STRUCTURES_FROM_ACTIVITY` signalisiert, daß das Programm nicht auf eine bestimmte Struktur des Input Containers angewiesen ist.

```
PROGRAM 'corba'
  STRUCTURES_FROM_ACTIVITY
  WINNT EXE PATH_AND_FILENAME
    "C:\Programme\FIX\Wrapper\CORBA.EXE"
     STYLE INVISIBLE
END 'corba'
```

Abbildung 5.4: Registrierung des CORBA Wrappers.

WfMS

Die in einem FIX Dokument beschriebenen Zielfunktionen müssen in Workflow-Definitionen innerhalb eines FDL Dokuments umgewandelt werden. Abbildung 5.5 zeigt den Rumpf der zu der globalen Funktion `get_color()` gehörenden Prozeßdefinition. Die Parameter der Funktion müssen dabei zuvor in Eingabe- und Ausgabeparameter aufgeteilt werden und als separate Strukturen definiert werden. Innerhalb der Prozeßdefinition werden diese beiden Strukturen durch den Ausdruck `('get_part_in', 'get_part_out')` als Typ des Input- bzw. Output-Containers spezifiziert. Die beiden weiteren Zeilen innerhalb der Prozeßdefinition dienen zur Definition des Start- bzw. Zielpunkts des Workflows.

Die benutzten Quellfunktionen müssen als Aktivitäten innerhalb eines Workflows definiert werden. Die Definition dieser Aktivitäten ist nicht zentral möglich, sondern muß innerhalb jeder Workflow-Definition erneut erfolgen. Abbildung 5.6 zeigt die Definition der zu der Quellfunktion `get_part()` gehörenden Aktivität. Durch das `INPUT_CONTAINER` Konstrukt werden Werte innerhalb des Input Containers mit Voreinstellungen belegt. Dies wird benutzt, um die von dem Quellwrapper[1] benötigten Daten anzugeben. Die referenzierten Strukturen `get_part_in` bzw. `get_part_out` müssen analog zu Abbildung 5.5 definiert werden.

[1] Der in diesem Fall die entfernte Funktion per ONC RPC aufruft.

```
STRUCTURE 'get_color_in'
  'no': LONG;
END 'get_color_in'

STRUCTURE 'get_color_out'
  'return': STRING;
END 'get_color_out'

/* weitere Strukturdefinitionen */

PROCESS 'get_color' ('get_color_in', 'get_color_out')
  SOURCE 1 XPOS -700 YPOS 600
  SINK 1 XPOS -700 YPOS -200

  /* Prozeßdefinition */

END 'get_part'
```

Abbildung 5.5: Definition eines Workflows.

```
PROGRAM_ACTIVITY 'get_part' ( 'get_part_in', 'get_part_out' )
  INPUT_CONTAINER
    'hostname' INITIAL_VALUE 'werk1.example.com'
    'program' INITIAL_VALUE '600000000'
    'version' INITIAL_VALUE '1'
    'signature' INITIAL_VALUE
      'void get_part(in partno_t partno, out string name, out color_t color)'
  PROGRAM 'oncrpc'
END 'get_part'
```

Abbildung 5.6: Definition einer Aktivität.

Die in einem Graphen enthaltenen Abhängigkeiten müssen innerhalb der Prozeßdefinition in Modellierungen des Datenflusses oder Kontrollflusses umgesetzt werden. Die durch den Abhängigkeitsgraphen implizit vorgegebene Ausführungsreihenfolge muß explizit modelliert werden. Die Modellierung von Kontroll- bzw. Datenfluß erfolgt in FDL durch `CONTROL` bzw. `DATA` Schlüsselworte (siehe Abbildung 5.7).

```
CONTROL
  FROM 'get_part' TO 'col2str'

DATA
  FROM SOURCE 1 TO 'get_part'
    MAP 'no' TO 'partno'

DATA
  FROM 'get_part' TO 'col2str'
    MAP 'color' TO 'color'

DATA
  FROM 'col2str'
    MAP 'return' TO 'return'
```

Abbildung 5.7: Modellierung des Kontroll- bzw. Datenflusses.

5.6 Zusammenfassung

In diesem Kapitel wurde eine mögliche Ausführungsumgebung anhand des MQ Workflow Management Systems von IBM vorgestellt. Es wurde aufgezeigt, welche zusätzlichen Komponenten implementiert bzw. integriert werden müssen, und wie FIX Dokumente in die von MQ Workflow verwendete Flowmark Definition Language konvertiert werden können. Bei der Verwendung von MQ Workflow als Ausführungsumgebung ergeben sich dabei folgende Schwierigkeiten:

Datentypen

MQ Workflow unterstützt die folgenden einfachen Datentypen:

- `STRING` — Strings variabler Länge,
- `BINARY` — Binärdaten variabler Länge,
- `LONG`,
- `FLOAT`.

Zusätzlich werden auf diesen Datentypen aufbauende Arrays und Strukturen unterstützt. Weitergehende, in CORBA bzw. FIX vorhandene Datentypen wie Unionen oder Aufzählungen, aber auch die verschiedenen Genauigkeiten von Fließkomma- oder Integerdatentypen müssen nachgebildet werden.

Im obigen Beispiel mußte aus diesem Grund der Typ `color_t` durch einen `LONG` Typ nachgebildet werden. Des weiteren ist eine Verwendung von Objekten als Parameter ebenfalls nicht möglich, Attributzugriffe müssen ebenso über Aktivitäten nachgebildet werden.

Leistung

Bedingt durch die Verwendung von DB2 und MQ Series sind die Hardwareanforderungen sehr hoch. Es wurde in Rahmen dieser Arbeit keine Analyse der zu erwarteten Leistung eines zur Funktionsintegration eingesetzten WfMS erstellt.

Durch den großen sich ergebenden Overhead ist es aber warscheinlich, daß ein Einsatz dieses Systems erst ab einer gewissen Komplexität der Abbildung sinnvoll ist. Durch die einfache Möglichkeit des Clustering bei MQ Series ist hierbei auch eine gute Skalierbarkeit zu erwarten.

Schnittstellen

MQ Workflow bietet keine Unterstützung für RPC- oder Objektdienste, aus diesem Grund müssen diese Schnittstellen erst durch Wrapper implementiert werden. Zusätzlich ist zur Implementierung des Zielsystems eine Verbindung des WfMS mit einem ORB zu realisieren.

GUI

Das von der Build-Time Komponente von MQ Workflow verwendete GUI ist für die Funktionsintegration eher ungeeignet, da es sich sehr stark an dem Konzept des “people driven” Workflows orientiert. Dadurch ist es zur Erstellung von Abhängigkeitsgraphen zur Funktionsintegration eher ungeeignet.

FDL

FDL eignet sich nur bedingt für die Beschreibung und Dokumentation von Anwendungssystemen. Zum einen müssen alle Daten der innerhalb eines Graphen verwendeten Quellfunktionen innerhalb desselben FDL Dokuments verwaltet werden. Die Dokumentation der einzelnen Systeme kann also nicht in mehrere (u.U. wiederverwendbare) Dokumente wie bei FIX erfolgen. Außerdem ist die Dokumentation durch das `DOCUMENTATION` Schlüsselwort nur begrenzt möglich.

Die Syntax von FDL ähnelt einer traditionellen Programmiersprache. Deshalb kann FDL nicht die Vorteile bieten, die sich bei FIX durch die Verwendung von XML ergeben.

Kapitel 6

Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Beschreibungssprache zur Funktionsintegration entwickelt. Dazu wurde zunächst ein allgemeines Modell zur Integration von Anwendungssystemen basierend auf der Beschreibung von Parameterabhängigkeiten vorgestellt. Es erfolgte eine Erweiterung dieses Modells um Konstrukte zur Kontrollflußsteuerung und Fehlerbehandlung. Außerdem wurde die Anwendung des Modells auf objektorientierte Systeme und den schreibenden Zugriff erläutert.

In Kapitel 4 wurde darauf aufbauend die Beschreibungssprache FIX vorgestellt. FIX ermöglicht sowohl die Beschreibung der exportierten Schnittstellen von Anwendungssystemen als auch die Modellierung der zur Integration benötigten Abhängigkeitsgraphen. Dabei wurde die Sprache so ausgelegt, daß einfache Integrationsaufgaben durch relativ einfache Graphen modelliert werden können. Kompliziertere Abbildungen sind durch die Benutzung von bedingten Anweisungen und Schleifenkonstrukte aber ebenfalls möglich.

Abschließend wurde anhand von MQ Workflow die mögliche Benutzung eines Workflow Management Systems als Ausführungsmodell für FIX Dokumente vorgestellt. Es wurde aufgezeigt, welche Erweiterungen an MQ Workflow vorgenommen werden müssen, damit FIX Dokumente verarbeitet werden können.

In zukünftigen Arbeiten sollte die Sprache vor allem an neue Versionen der verwendeten Standards angepaßt werden. So sind sowohl der XLink als aus XPointer Draft zum Zeitpunkt der Erstellung dieser Arbeit im “Last Call Status”, eine baldige Verabschiedung als W3C Recommendation ist also zu erwarten. Des weiteren sollte von der Validierung gegenüber einer DTD auf ein XML Schema umgestellt werden, sobald dieser Draft standardisiert wurde. XML Schemas bieten einige Vorteile gegenüber DTDs, vor allem im Bezug auf die Einbindung von anderen Namensräumen und der Spezifikation der in den einzelnen Elementen erlaubten Datentypen. Es sollte ebenfalls geprüft werden, inwieweit es sinnvoll ist, das existierende Document Object Model [DOM] für FIX Dokumente zu erweitern.

Die Dokumentation der verwendeten Systeme soll ein weiterer Schwerpunkt der Sprache FIX werden. Momentan wird dies nur rudimentär durch die `<desc>` und `<a>` Elemente unterstützt. Es sollte in zukünftigen Arbeiten geprüft werden, welche Daten über die Systeme (beispielsweise Hardware, Standort, etc.) und ihre exportierten Schnittstellen (z.B. in Form von Vor- und Nachbedingungen) innerhalb von FIX Dokumenten gesammelt werden müssen, damit diese als eine Art “System Repository”, also eine umfassende Dokumentationssammlung über die verwendeten Systeme, benutzt werden können. Ebenfalls sind die Möglichkeiten zur Verknüpfung von FIX Dokumenten mit bereits

bestehender Dokumentation zu prüfen. Des weiteren sind weitere XSLT Stylesheets zur Konvertierung in andere Formate (beispielsweise HTML) denkbar.

Eine Schwäche von XML basierten Sprachen gegenüber traditionellen Computersprachen ist, daß eine Programmierung "von Hand" zwar möglich, aber eher umständlich ist. Aus diesem Grund ist die Entwicklung einer graphischen Oberfläche zur Modellierung der Abhängigkeitsgraphen (und damit der FIX Dokumente) notwendig. Diese könnte sich an den in dieser Arbeit verwendeten Diagrammen orientieren. Die Build-Time Komponente des MQ Workflow Systems ist hierfür nicht geeignet, da sie sich zu sehr an dem Modell der "people driven" Workflows orientiert.

Zukünftige Arbeiten sollten ebenfalls eine Erweiterung der in den Abschnitten 3.7.1 bzw. 4.13 beschriebenen Standardbibliothek beinhalten. Eine umfangreiche Standardbibliothek führt zu einer vereinfachten Modellierung der Abhängigkeitsgraphen und zu weniger Inkompatibilitäten zwischen den Implementierungen.

Um MQ Workflow als Ausführungsumgebung benutzen zu können sind einige weitere Arbeiten notwendig. So ist die Erstellung von Wrappern zum Zugriff auf RPC- oder Objektdienste notwendig, außerdem muß eine Anbindung an einen CORBA ORB realisiert werden. Des weiteren müssen Konvertoren (eventuell ebenfalls auf Basis von XSLT) zwischen FIX und der von MQ Workflow verwendeten Flowmark Definition Language erstellt werden. Außerdem sollten innerhalb einer zukünftigen Arbeit evaluiert werden, inwieweit das sich bei der Verwendung eines WfMS ergebende Leistungsverhalten den Anforderungen entspricht.

Anhang A

DTD

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!--
        FIX DTD V1.0

        SYSTEM "http://dcx.com/fix/fix.dtd"

        $Id: fix.dtd,v 1.32 2000/06/15 11:10:42 claude Exp $
-->

<!--
          builtin entities
-->

<!ENTITY  lt                    "&#38;#60;">
<!ENTITY  gt                    "&#62;">
<!ENTITY  amp                   "&#38;#38;">
<!ENTITY  apos                  "&#39;">
<!ENTITY  quot                  "&#34;">

<!--
          Entities zur Beschreibung von Attributtypen
-->

<!-- XML Schema Datentypen -->

<!ENTITY  % string              "CDATA">
<!ENTITY  % boolean             "(true | 1 | false | 0)">
<!ENTITY  % float               "CDATA">
<!ENTITY  % uri                 "CDATA">
<!ENTITY  % language            "NMTOKEN">
<!ENTITY  % Name                "NMTOKEN">
<!ENTITY  % QName               "NMTOKEN">
<!ENTITY  % NCName              "NMTOKEN">
<!ENTITY  % integer             "CDATA">
<!ENTITY  % non-negative-integer "CDATA">
<!ENTITY  % positive-integer    "CDATA">

<!-- weitere Datentypen -->
```

```
<!-- MIME Typ (RFC 2045) -->

<!ENTITY  % contenttype         "CDATA">

<!-- Semantic -->

<!ENTITY  % semantic            "NMTOKEN">

<!-- Coords (HTML 4.01) -->

<!ENTITY  % coords              "CDATA">

<!--
          Entities zur Beschreibung oft benötigter Attribute
-->

<!ENTITY  % id
          "id                   ID #IMPLIED">

<!ENTITY  % simplelink
          'xlink:type           (simple) #FIXED "simple"
          xlink:href            %uri; #REQUIRED'>

<!--
          Entities zur Beschreibung bestimmter Typklassen
-->

<!-- % base -->

<!ENTITY  % base                "(float | int | char | bool | octet | any |
                                object | valuebase)">

<!-- % template -->

<!ENTITY  % template            "(sequence | string | fixed)">

<!-- % simple -->

<!ENTITY  % simple              "(%base; | %template; | type)">

<!-- % constr -->

<!ENTITY  % constr              "(struct | union | enum)">

<!-- % datatype -->

<!ENTITY  % datatype            "(%simple; | %constr;)">

<!-- % param -->

<!ENTITY  % param               "(%base; | string | type)">

<!-- % type_dcl -->

<!ENTITY  % type_dcl            "(typedef | struct | union | enum | native)">

<!-- % definition -->
```

```
<!ENTITY  % definition          "(%type_dcl; | const | except | interface |
                                module | valuetype)">

<!-- % export -->

<!ENTITY  % export              "(%type_dcl; | const | except | attr | op)">

<!--
          Entities zur Beschreibung oft benötigten Contents
-->

<!ENTITY  % doc                 "(title, desc*)">

<!ENTITY  % typedef             "(%doc;, %datatype;, array*)">

<!--
          Allgemeine Elemente
-->

<!-- title -->

<!ELEMENT title                 (#PCDATA)>
<!ATTLIST title
          xlink:type            (title) #FIXED "title">

<!-- desc -->

<!ELEMENT desc                  (#PCDATA | a)*>
<!ATTLIST desc
          xml:lang              %language; #IMPLIED>

<!-- a -->

<!ELEMENT a                     (#PCDATA)>
<!ATTLIST a
          %simplelink;>

<!--
          Systembeschreibung
-->

<!-- system - Root Element -->

<!ELEMENT system                (%doc;, address?,
                                (dcom | iiop | local | oncrpc | rmi),
                                (%definition;)*)>
<!ATTLIST system
          xmlns                 %uri; #FIXED "http://dcx.com/fix"
          xmlns:fix             %uri; #FIXED "http://dcx.com/fix"
          xmlns:xlink           %uri; #FIXED "http://www.w3.org/1999/xlink">

<!-- address -->

<!ELEMENT address               (#PCDATA)>

<!-- dcom -->
```

```
<!ELEMENT dcom                  EMPTY>
<!ATTLIST dcom
          uuid                  %string; #REQUIRED
          version               %float; #REQUIRED>

<!-- iiop -->

<!ELEMENT iiop                  EMPTY>
<!ATTLIST iiop
          port                  %positive-integer; #REQUIRED>

<!-- local -->

<!ELEMENT local                 EMPTY>
<!ATTLIST local
          filename              %string; #REQUIRED>

<!-- oncrpc -->

<!ELEMENT oncrpc                EMPTY>
<!ATTLIST oncrpc
          program               %string; #REQUIRED
          version               %non-negative-integer; #REQUIRED
          transport             (tcp | udp) "udp">

<!-- rmi -->

<!ELEMENT rmi                   EMPTY>
<!ATTLIST rmi
          uri                   %uri; #REQUIRED>

<!--
          Schnittstellendefinition
-->

<!--
          Definition neuer Datentypen
-->

<!-- typedef -->

<!ELEMENT typedef               (%typedef;)>
<!ATTLIST typedef
          %id;>

<!-- array -->

<!ELEMENT array                 EMPTY>
<!ATTLIST array
          count                 %positive-integer; #REQUIRED>

<!-- type -->

<!ELEMENT type                  EMPTY>
<!ATTLIST type
          %simplelink;>
```

```
<!--
          Einfache Datentypen
-->

<!-- float -->

<!ELEMENT float                EMPTY>
<!ATTLIST float
          type                 (float | double | long_double) "double">

<!-- int -->

<!ELEMENT int                  EMPTY>
<!ATTLIST int
          type                 (short | long | long_long) "long"
          unsigned             %boolean; "false">

<!-- char -->

<!ELEMENT char                 EMPTY>
<!ATTLIST char
          wide                 %boolean; "false">

<!-- bool -->

<!ELEMENT bool                 EMPTY>

<!-- octet -->

<!ELEMENT octet                EMPTY>

<!-- any -->

<!ELEMENT any                  EMPTY>

<!-- object -->

<!ELEMENT object               EMPTY>

<!-- valuebase -->

<!ELEMENT valuebase            EMPTY>

<!--
          Vorlagentypen
-->

<!-- sequence -->

<!ELEMENT sequence             (%simple;)>
<!ATTLIST sequence
          count                %positive-integer; #IMPLIED>

<!-- string -->

<!ELEMENT string               EMPTY>
```

```
<!ATTLIST string
          count                %positive-integer; #IMPLIED
          wide                 %boolean; "false">

<!-- fixed -->

<!ELEMENT fixed                EMPTY>
<!ATTLIST fixed
          digits               %positive-integer; #IMPLIED
          scale                %positive-integer; #IMPLIED>

<!--
          Strukturierte Datentypen
-->

<!-- struct -->

<!ELEMENT struct               (%doc;, member+)>
<!ATTLIST struct
          %id;>

<!-- member -->

<!ELEMENT member               (%typedef;)>
<!ATTLIST member
          %id;>

<!-- union -->

<!ELEMENT union                (%doc;, (int | char | bool | enum |
                               type), case+, default?)>
<!ATTLIST union
          %id;>

<!-- case -->

<!ELEMENT case                 (%typedef;, expr)>
<!ATTLIST case
          %id;>

<!-- default -->

<!ELEMENT default              (%typedef;)>
<!ATTLIST default
          %id;>

<!-- enum -->

<!ELEMENT enum                 (%doc;, enumerator+)>
<!ATTLIST enum
          %id;>

<!-- enumerator -->

<!ELEMENT enumerator           (%doc;)>
<!ATTLIST enumerator
          %id;>
```

```
<!--
          native
-->

<!-- native -->

<!ELEMENT native                (%doc;)>
<!ATTLIST native
          %id;>

<!--
          Konstanten
-->

<!-- const -->

<!ELEMENT const                 (%doc;, (int | char | bool | float |
                                string | fixed | type | octet), expr?)>
<!ATTLIST const
          %id;>

<!-- expr -->

<!ELEMENT expr                  ANY>
<!ATTLIST expr
          type                  %contenttype; #REQUIRED
          xml:space             (default | preserve) "default">

<!--
          Operationen und Attribute
-->

<!-- op -->

<!ELEMENT op                    (%doc;, param*, return?, raises*, context*,
                                effect*, graph*)>
<!ATTLIST op
          %id;
          oneway                %boolean; "false"
          semantic              %semantic; "undefined">

<!-- param -->

<!ELEMENT param                 (%doc;, %param;)>
<!ATTLIST param
          %id;
          type                  (in | out | inout) "in">

<!-- return -->

<!ELEMENT return                (desc*, %param;)>
<!ATTLIST return
          %id;>

<!-- raises -->
```

```
<!ELEMENT raises                EMPTY>
<!ATTLIST raises
          %simplelink;>

<!-- except -->

<!ELEMENT except                (%doc;, member*)>
<!ATTLIST except
          %id;>

<!-- context -->

<!ELEMENT context               (#PCDATA)>

<!-- effect -->

<!ELEMENT effect                (%doc;)>
<!ATTLIST effect
          %id;
          type                  (exec | read | write) "exec">

<!-- attr -->

<!ELEMENT attr                  (%doc;, %param;, effect*, graph*)>
<!ATTLIST attr
          %id;
          readonly              %boolean; "false">

<!--
          Module und Interfaces
-->

<!-- module -->

<!ELEMENT module                (%doc;, (%definition;)+)>

<!-- interface -->

<!ELEMENT interface             (%doc;, inherits*, (%export;)*)>
<!ATTLIST interface
          %id;
          abstract              %boolean; "false">

<!-- inherits -->

<!ELEMENT inherits              EMPTY>
<!ATTLIST inherits
          %simplelink;>

<!--
          Value Types
-->

<!-- valuetype -->

<!ELEMENT valuetype             (%doc;, inherits*, supports*,
                                ((%export; | statemember | factory)* | boxed))>
```

```
<!ATTLIST valuetype
          %id;
          custom                %boolean; "false"
          abstract              %boolean; "false"
          truncatable           %boolean; "false">

<!-- supports -->

<!ELEMENT supports              EMPTY>
<!ATTLIST supports
          %simplelink;>

<!-- boxed -->

<!ELEMENT boxed                 (%datatype;)>

<!-- statemember -->

<!ELEMENT statemember           (%typedef;, graph*)>
<!ATTLIST statemember
          %id;
          type                  (public | private) "public">

<!-- factory -->

<!ELEMENT factory               (%doc;, param*, graph*)>
<!ATTLIST factory
          %id;
          semantic              %semantic; "undefined">

<!--
          Abhängigkeitsbeschreibung
-->

<!-- mapping - Root Element -->

<!ELEMENT mapping               (graph)*>

<!-- graph -->

<!ELEMENT graph                 (desc*, ((node | todo | dep)+ | ext))>
<!ATTLIST graph
          %id;
          xlink:type            (extended) #FIXED "extended"
          xlink:role            (exec | read | write |
                                xlink:external-linkset) "exec"
          type                  (implementation | wrapper) "implementation">

<!-- node -->

<!ELEMENT node                  (%doc;)>
<!ATTLIST node
          xlink:type            (locator) #FIXED "locator"
          xlink:href            %uri; #REQUIRED
          xlink:role            %Name; #REQUIRED
          instance              %Name; "fix:default"
          semantic              %semantic; "undefined"
```

```
          compensation          %boolean; "true"
          coords                %coords; #IMPLIED>

<!-- todo -->

<!ELEMENT todo                  (%doc;)>
<!ATTLIST todo
          xlink:type            (resource) #FIXED "resource"
          xlink:role            %Name; #REQUIRED
          coords                %coords; #IMPLIED>

<!-- dep -->

<!ELEMENT dep                   (%doc;)>
<!ATTLIST dep
          xlink:type            (arc) #FIXED "arc"
          xlink:from            %Name; #REQUIRED
          xlink:to              %Name; #REQUIRED
          xlink:role            (implementation | compensation | exception)
                                "implementation"
          invocation            (single | cyclic) "single"
          completion            (required | sufficient | optional) "sufficient"
          priority              %integer; "0"
          coords                %coords; #IMPLIED>

<!-- ext -->

<!ELEMENT ext                   EMPTY>
<!ATTLIST ext
          xlink:type            (locator) #FIXED "locator"
          xlink:href            %uri; #REQUIRED>
```

Anhang B

Schema

Das hier vorgestellte Schema basiert auf den XML Schema Drafts vom 17.12.99 ([SCHEMA1] und [SCHEMA2]

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE schema
          PUBLIC "-//W3C//DTD XML Schema Version 1.0//EN"
"http://www.w3.org/TR/1999/WD-xmlschema-1-19991217/structures.dtd" [
<!ATTLIST schema
          xmlns:xml             CDATA #FIXED
                                "http://www.w3.org/XML/1998/namespace"
          xmlns:fix             CDATA #FIXED
                                "http://dcx.com/fix">
]>

<!--
        FIX Schema V1.0

        $Id: fix.xsd,v 1.8 2000/06/15 11:10:43 claude Exp $
-->

<schema version="$Id: fix.xsd,v 1.8 2000/06/15 11:10:43 claude Exp $"
 xmlns="http://www.w3.org/1999/XMLSchema"
 targetNamespace="http://dcx.com/fix"
 xmlns:xml="http://www.w3.org/XML/1998/namespace"
 xmlns:fix="http://dcx.com/fix"
 exactDefault="#all" finalDefault="#all">

  <!-- Schema Imports -->

  <import namespace="http://www.w3.org/XML/1998/namespace"
   schemaLocation="http://www.w3.org/XML/1998/xml.xsd"/>
  <import namespace="http://www.w3.org/1999/xlink"/><!-- Schema existiert
                                                        noch nicht -->

  <!-- Datentypen -->

  <datatype name="contenttypeType" source="string">
    <pattern value="[^/]*(/[^/]*)?"/>
```

```
</datatype>

<datatype name="semanticType" source="NCName"/>

<datatype name="coordsType" source="string">
  <pattern value="[0-9]+(,[0-9]+)*"/>
</datatype>

<!-- Groups -->

<group name="base" order="choice">
  <element name="float" type="fix:floatType"/>
  <element name="int" type="fix:intType"/>
  <element name="char" type="fix:charType"/>
  <element name="bool" type="fix:boolType"/>
  <element name="octet" type="fix:octetType"/>
  <element name="any" type="fix:anyType"/>
  <element name="object" type="fix:objectType"/>
  <element name="valuebase" type="fix:valuebaseType"/>
</group>

<group name="template" order="choice">
  <element name="sequence" type="fix:sequenceType"/>
  <element name="string" type="fix:stringType"/>
  <element name="fixed" type="fix:fixedType"/>
</group>

<group name="simple" order="choice">
  <group ref="fix:base"/>
  <group ref="fix:template"/>
  <element name="type" type="fix:typeType"/>
</group>

<group name="constr" order="choice">
  <element name="struct" type="fix:structType"/>
  <element name="union" type="fix:unionType"/>
  <element name="enum" type="fix:enumType"/>
</group>

<group name="datatype" order="choice">
  <group ref="fix:simple"/>
  <group ref="fix:constr"/>
</group>

<group name="param" order="choice">
  <group ref="fix:base"/>
  <element name="string" type="fix:stringType"/>
  <element name="type" type="fix:typeType"/>
</group>

<group name="type_dcl" order="choice">
  <element name="typedef" type="fix:typedefType"/>
  <element name="struct" type="fix:structType"/>
  <element name="union" type="fix:unionType"/>
  <element name="enum" type="fix:enumType"/>
  <element name="native" type="fix:nativeType"/>
</group>
```

```
<group name="definition" order="choice">
  <group ref="fix:type_dcl"/>
  <element name="const" type="fix:constType"/>
  <element name="except" type="fix:exceptType"/>
  <element name="interface" type="fix:interfaceType"/>
  <element name="module" type="fix:moduleType"/>
  <element name="valuetype" type="fix:valuetypeType"/>
</group>

<group name="export" order="choice">
  <group ref="fix:type_dcl"/>
  <element name="const" type="fix:constType"/>
  <element name="except" type="fix:exceptType"/>
  <element name="attr" type="fix:attrType"/>
  <element name="op" type="fix:opType"/>
</group>

<!-- Allgemeine Elemente -->

<type name="simpleLinkType" content="empty" final="">
  <attribute name="xlink:type" type="NCName" fixed="simple"/>
  <attribute name="xlink:href" type="uri" minOccurs="1"/>
</type>

<type name="descType" content="mixed">
  <element name="a">
    <type content="textOnly" source="fix:simpleLinkType"
     derivedBy="extension"/>
  </element>
  <any namespace="http://www.w3.org/1999/xhtml"/>
  <attributeGroup ref="xml:lang"/>
</type>

<type name="docNoIdType" final="">
  <element name="title">
    <type content="textOnly">
      <attribute name="xlink:type" type="NCName" fixed="title"/>
    </type>
  </element>
  <element name="desc" type="fix:descType" minOccurs="0" maxOccurs="*"/>
</type>

<type name="docType" source="fix:docNoIdType" derivedBy="extension" final="">
  <attribute name="id" type="ID"/>
</type>

<!-- Systembeschreibung -->

<element name="system">
  <type source="fix:docNoIdType" derivedBy="extension">
    <element name="address" minOccurs="0">
      <datatype source="string">
        <pattern value="[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*"/>
      </datatype>
    </element>
    <group order="choice">
```

```
      <element name="dcom">
        <type content="empty">
          <attribute name="uuid" type="string" minOccurs="1"/>
          <attribute name="version" type="float" minOccurs="1"/>
        </type>
      </element>
      <element name="iiop">
        <type content="empty">
          <attribute name="port" type="positive-integer" minOccurs="1"/>
        </type>
      </element>
      <element name="local">
        <type content="empty">
          <attribute name="filename" type="string" minOccurs="1"/>
        </type>
      </element>
      <element name="oncrpc">
        <type content="empty">
          <attribute name="program" minOccurs="1">
            <datatype source="positive-integer">
              <encoding value="hex"/>
            </datatype>
          </attribute>
          <attribute name="version" type="non-negative-integer"
           minOccurs="1"/>
          <attribute name="transport" default="udp">
            <datatype source="NCName">
              <enumeration value="tcp"/>
              <enumeration value="udp"/>
            </datatype>
          </attribute>
        </type>
      </element>
      <element name="rmi">
        <type content="empty">
          <attribute name="uri" type="uri" minOccurs="1"/>
        </type>
      </element>
    </group>
    <group ref="fix:definition" minOccurs="0" maxOccurs="*"/>
  </type>
</element>

<!-- Schnittstellendefinition -->

<!-- Definition neuer Datentypen -->

<type name="typedefType" source="fix:docType" derivedBy="extension" final="">
  <group ref="fix:datatype"/>
  <element name="array" minOccurs="0" maxOccurs="*">
    <type content="empty">
      <attribute name="count" type="positive-integer"/>
    </type>
  </element>
</type>

<type name="typeType" source="fix:simpleLinkType"/>
```

```
<!-- Einfache Datentypen -->

<type name="floatType" content="empty">
  <attribute name="type" default="double">
    <datatype source="NCName">
      <enumeration value="float"/>
      <enumeration value="double"/>
      <enumeration value="long_double"/>
    </datatype>
  </attribute>
</type>

<type name="intType" content="empty">
  <attribute name="type" default="long">
    <datatype source="NCName">
      <enumeration value="short"/>
      <enumeration value="long"/>
      <enumeration value="long_long"/>
    </datatype>
  </attribute>
  <attribute name="unsigned" type="boolean" default="false"/>
</type>

<type name="charType" content="empty">
  <attribute name="wide" type="boolean" default="false"/>
</type>

<type name="boolType" content="empty"/>

<type name="octetType" content="empty"/>

<type name="anyType" content="empty"/>

<type name="objectType" content="empty"/>

<type name="valuebaseType" content="empty"/>

<!-- Vorlagentypen -->

<type name="sequenceType">
  <group ref="fix:simple"/>
  <attribute name="count" type="positive-integer"/>
</type>

<type name="stringType" content="empty">
  <attribute name="count" type="positive-integer"/>
  <attribute name="wide" type="boolean" default="false"/>
</type>

<type name="fixedType" content="empty" final="">
  <attribute name="digits" type="positive-integer" minOccurs="1"/>
  <attribute name="scale" type="positive-integer" minOccurs="1"/>
</type>

<!-- Strukturierte Datentypen -->
```

```
<type name="structType" source="fix:docType" derivedBy="extension">
  <element name="member" type="fix:typedefType" maxOccurs="*"/>
</type>

<type name="unionType" source="fix:docType" derivedBy="extension">
  <group order="choice">
    <element name="int" type="fix:intType"/>
    <element name="char" type="fix:charType"/>
    <element name="bool" type="fix:boolType"/>
    <element name="enum" type="fix:enumType"/>
    <element name="type" type="fix:typeType"/>
  </group>
  <element name="case" maxOccurs="*">
    <type source="fix:typedefType" derivedBy="extension">
      <element name="expr" type="fix:exprType"/>
    </type>
  </element>
  <element name="default" type="fix:typedefType" minOccurs="0"/>
</type>

<type name="enumType" source="fix:docType" derivedBy="extension">
  <element name="enumerator"  type="fix:docType" maxOccurs="*"/>
</type>

<!-- native -->

<type name="nativeType" source="fix:docType"/>

<!-- Konstanten -->

<type name="constType" source="fix:docType" derivedBy="extension">
  <group order="choice">
    <element name="int" type="fix:intType"/>
    <element name="char" type="fix:charType"/>
    <element name="bool" type="fix:boolType"/>
    <element name="float" type="fix:float"/>
    <element name="string" type="fix:stringType"/>
    <element name="fixed">
      <type source="fix:fixedType" derivedBy="restriction">
        <restrictions>
          <attribute name="digits" type="positive-integer" minOccurs="0"
           maxOccurs="0"/>
          <attribute name="scale" type="positive-integer" minOccurs="0"
           maxOccurs="0"/>
        </restrictions>
      </type>
    </element>
    <element name="type" type="fix:typeType"/>
    <element name="octet" type="fix:octetType"/>
  </group>
  <element name="expr" type="fix:exprType" minOccurs="0"/>
</type>

<type name="exprType" content="mixed">
  <any/>
  <attribute name="type" type="fix:contenttypeType"/>
  <attributeGroup ref="xml:space"/>
```

```
</type>

<!-- Operationen und Attribute -->

<type name="opType" source="fix:docType" derivedBy="extension">
  <element name="param" type="fix:paramType" minOccurs="0" maxOccurs="*"/>
  <element name="return" minOccurs="0">
    <type content="elementOnly">
      <element name="desc" type="fix:descType" minOccurs="0" maxOccurs="*"/>
      <group ref="fix:param"/>
    </type>
  </element>
  <element name="raises" type="fix:simpleLinkType" minOccurs="0"
   maxOccurs="*"/>
  <element name="context" minOccurs="0" maxOccurs="*">
    <type content="textOnly"/>
  </element>
  <element name="effect" minOccurs="0" maxOccurs="*">
    <type source="fix:docType" derivedBy="extension">
      <attribute name="type" fixed="exec"/>
    </type>
  </element>
  <element name="graph" minOccurs="0" maxOccurs="2">
    <type source="fix:graphType" derivedBy="restriction">
      <restrictions>
        <attribute name="xlink:role" default="exec">
          <datatype source="Name">
            <enumeration value="exec"/>
            <enumeration value="xlink:external-linkset"/>
          </datatype>
        </attribute>
      </restrictions>
    </type>
  </element>
  <attribute name="oneway" type="boolean" default="false"/>
  <attribute name="semantic" type="semanticType" default="undefined"/>
</type>

<type name="paramType" source="fix:docType" derivedBy="extension" final="">
  <group ref="fix:param"/>
  <attribute name="type" default="in">
    <datatype source="NCName">
      <enumeration value="in"/>
      <enumeration value="out"/>
      <enumeration value="inout"/>
    </datatype>
  </attribute>
</type>

<type name="exceptType" source="fix:docType" derivedBy="extension">
  <element name="member" type="fix:typedefType" minOccurs="0"
   maxOccurs="*"/>
</type>

<type name="attrType" source="fix:docType" derivedBy="extension">
  <group ref="fix:param"/>
  <element name="effect" minOccurs="0" maxOccurs="*">
```

```
    <type source="fix:docType" derivedBy="extension">
      <attribute name="type" default="exec">
        <datatype source="NCName">
          <enumeration value="exec"/>
          <enumeration value="read"/>
          <enumeration value="write"/>
        </datatype>
      </attribute>
    </type>
  </element>
  <element name="graph" minOccurs="0" maxOccurs="4">
    <type source="fix:graphType" derivedBy="restriction">
      <restrictions>
        <attribute name="xlink:role" minOccurs="1">
          <datatype source="Name">
            <enumeration value="read"/>
            <enumeration value="write"/>
            <enumeration value="xlink:external-linkset"/>
          </datatype>
        </attribute>
      </restrictions>
    </type>
  </element>
  <attribute name="readonly" type="boolean" default="false"/>
</type>

<!-- Module und Interfaces -->

<type name="moduleType" source="fix:docNoIdType" derivedBy="extension">
  <group ref="fix:definition" maxOccurs="*"/>
</type>

<type name="interfaceType" source="fix:docType" derivedBy="extension">
  <element name="inherits" type="fix:simpleLinkType" minOccurs="0"
   maxOccurs="*"/>
  <group ref="fix:export" minOccurs="0" maxOccurs="*"/>
  <attribute name="abstract" type="boolean" default="false"/>
</type>

<!-- Value Types -->

<type name="valuetypeType" source="fix:docType" derivedBy="extension">
  <element name="inherits" type="fix:simpleLinkType" minOccurs="0"
   maxOccurs="*"/>
  <element name="inherits" type="fix:simpleLinkType" minOccurs="0"
   maxOccurs="*"/>
  <group order="choice">
    <group order="choice" minOccurs="0" maxOccurs="*">
      <group ref="fix:export"/>
      <element name="statemember">
        <type source="fix:typedefType" derivedBy="extension">
          <element name="graph" minOccurs="0" maxOccurs="4">
            <type source="fix:graphType" derivedBy="restriction">
              <restrictions>
                <attribute name="xlink:role" minOccurs="1">
                  <datatype source="Name">
                    <enumeration value="read"/>
```

```
                    <enumeration value="write"/>
                    <enumeration value="xlink:external-linkset"/>
                  </datatype>
                </attribute>
              </restrictions>
            </type>
          </element>
          <attribute name="type" default="public">
            <datatype source="NCName">
              <enumeration value="public"/>
              <enumeration value="private"/>
            </datatype>
          </attribute>
        </type>
      </element>
      <element name="factory">
        <type source="fix:docType" derivedBy="extension">
          <element name="param" minOccurs="0" maxOccurs="*">
            <type source="fix:paramType" derivedBy="restriction">
              <restrictions>
                <attribute name="type" type="NCName" fixed="in"/>
              </restrictions>
            </type>
          </element>
          <element name="graph" minOccurs="0" maxOccurs="2">
            <type source="fix:graphType" derivedBy="restriction">
              <restrictions>
                <attribute name="xlink:role" default="exec">
                  <datatype source="Name">
                    <enumeration value="exec"/>
                    <enumeration value="xlink:external-linkset"/>
                  </datatype>
                </attribute>
              </restrictions>
            </type>
          </element>
          <attribute name="semantic" type="semanticType"
           default="undefined"/>
        </type>
      </element>
    </group>
    <element name="boxed">
      <type>
        <group ref="fix:datatype"/>
      </type>
    </element>
  </group>
  <attribute name="custom" type="boolean" default="false"/>
  <attribute name="abstract" type="boolean" default="false"/>
  <attribute name="truncatable" type="boolean" default="false"/>
</type>

<!-- Abhängigkeitsbeschreibung -->

<element name="mapping">
  <type content="elementOnly">
    <element name="graph" type="fix:graphType" minOccurs="0" maxOccurs="*"/>
```

```
  </type>
</element>

<type name="graphType" content="elementOnly" final="">
  <element name="desc" type="fix:descType" minOccurs="0" maxOccurs="*"/>
  <group order="choice">
    <group order="choice" maxOccurs="*">
      <element name="node" type="fix:nodeType"/>
      <element name="todo" type="fix:todoType"/>
      <element name="dep" type="fix:depType"/>
    </group>
    <element name="ext" type="fix:extType"/>
  </group>
  <attribute name="id" type="ID"/>
  <attribute name="xlink:type" type="NCName" fixed="extended"/>
  <attribute name="xlink:role" default="exec">
    <datatype source="Name">
      <enumeration value="exec"/>
      <enumeration value="read"/>
      <enumeration value="write"/>
      <enumeration value="xlink:external-linkset"/>
    </datatype>
  </attribute>
  <attribute name="type" default="implementation">
    <datatype source="NCName">
      <enumeration value="implementation"/>
      <enumeration value="wrapper"/>
    </datatype>
  </attribute>
</type>

<type name="nodeType" source="fix:docNoIdType">
  <attribute name="xlink:type" type="NCName" fixed="locator"/>
  <attribute name="xlink:href" type="uri" minOccurs="1"/>
  <attribute name="xlink:role" type="Name" minOccurs="1"/>
  <attribute name="instance" type="Name" default="fix:default"/>
  <attribute name="semantic" type="fix:semanticType" default="undefined"/>
  <attribute name="compensation" type="boolean" default="true"/>
  <attribute name="coords" type="fix:coordsType"/>
</type>

<type name="todoType" source="fix:docNoIdType">
  <attribute name="xlink:type" type="NCName" fixed="resource"/>
  <attribute name="xlink:role" type="Name" minOccurs="1"/>
  <attribute name="coords" type="fix:coordsType"/>
</type>

<type name="depType" source="fix:docNoIdType">
  <attribute name="xlink:type" type="NCName" fixed="arc"/>
  <attribute name="xlink:from" type="Name" minOccurs="1"/>
  <attribute name="xlink:to" type="Name" minOccurs="1"/>
  <attribute name="xlink:role" default="implementation">
    <datatype source="NCName">
      <enumeration value="implementation"/>
      <enumeration value="compensation"/>
      <enumeration value="exception"/>
    </datatype>
```

```
    </attribute>
    <attribute name="invocation" default="single">
      <datatype source="NCName">
        <enumeration value="single"/>
        <enumeration value="cyclic"/>
      </datatype>
    </attribute>
    <attribute name="completion" default="sufficient">
      <datatype source="NCName">
        <enumeration value="required"/>
        <enumeration value="sufficient"/>
        <enumeration value="optional"/>
      </datatype>
    </attribute>
    <attribute name="priority" type="integer" default="0"/>
    <attribute name="coords" type="fix:coordsType"/>
  </type>

  <type name="extType" content="empty">
    <attribute name="xlink:type" type="NCName" fixed="locator"/>
    <attribute name="xlink:href" type="uri" minOccurs="1"/>
  </type>

</schema>
```

Anhang C

Standardbibliothek

C.1 stdlib.idl

```
/* Standard Library
 *
 * $Id: stdlib.idl,v 1.6 2000/07/20 08:02:25 claude Exp $
 */

interface stdlib {

  typedef sequence<any> anyseq;

  exception EFALSE {};

  void eval(in string format, inout any value) raises (EFALSE);
  void evalseq(in string format, inout anyseq values) raises (EFALSE);

  void istrue(inout boolean value) raises (EFALSE);
  void isfalse(inout boolean value) raises (EFALSE);

  void isequal(inout any a, inout any b) raises (EFALSE);
  void isnotequal(inout any a, inout any b) raises (EFALSE);

  void iszero(inout any value) raises (EFALSE);
  void isnotzero(inout any value) raises (EFALSE);

  void loop(in long max, inout long count) raises (EFALSE);

  void incr(inout long value);
  void decr(inout long value);

  long strtoint(in string str);
  string inttostr(in long int);

  /* Konvertierungsfunktionen */
  /* Listenfunktionen */
  /* Stringfunktionen */

}
```

C.2 stdlib.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE system SYSTEM "http://dcx.com/fix.fix.dtd">

<!-- $Id: stdlib.xml,v 1.9 2000/07/20 08:05:45 claude Exp $ -->

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>stdlib</title>
  <local filename="stdlib.o"/>

  <interface id="stdlib">
    <title>stdlib</title>

    <typedef id="anyseq">
      <title>anyseq</title>
      <sequence>
        <any/>
      </sequence>
    </typedef>

    <except id="EFALSE">
      <title>EFALSE</title>
    </except>

    <op id="eval" semantic="condition">
      <title>eval</title>
      <param id="eval.format">
        <title>format</title>
        <string/>
      </param>
      <param id="eval.value" type="inout">
        <title>value</title>
        <any/>
      </param>
      <raises xlink:href="#EFALSE"/>
    </op>

    <op id="evalseq" semantic="condition">
      <title>evalseq</title>
      <param id="evalseq.format">
        <title>format</title>
        <string/>
      </param>
      <param id="evalseq.values" type="inout">
        <title>values</title>
        <type xlink:href="#anyseq"/>
      </param>
      <raises xlink:href="#EFALSE"/>
    </op>

    <op id="istrue" semantic="condition">
      <title>istrue</title>
      <param id="istrue.value" type="inout">
        <title>value</title>
```

```
    <bool/>
  </param>
  <raises xlink:href="#EFALSE"/>
</op>

<op id="isfalse" semantic="condition">
  <title>isfalse</title>
  <param id="isfalse.value" type="inout">
    <title>value</title>
    <bool/>
  </param>
  <raises xlink:href="#EFALSE"/>
</op>

<op id="isequal" semantic="condition">
  <title>isequal</title>
  <param id="isequal.a" type="inout">
    <title>a</title>
    <any/>
  </param>
  <param id="isequal.b" type="inout">
    <title>b</title>
    <any/>
  </param>
  <raises xlink:href="#EFALSE"/>
</op>

<op id="isnotequal" semantic="condition">
  <title>isnotequal</title>
  <param id="isnotequal.a" type="inout">
    <title>a</title>
    <any/>
  </param>
  <param id="isnotequal.b" type="inout">
    <title>b</title>
    <any/>
  </param>
  <raises xlink:href="#EFALSE"/>
</op>

<op id="iszero" semantic="condition">
  <title>iszero</title>
  <param id="iszero.value" type="inout">
    <title>value</title>
    <any/>
  </param>
  <raises xlink:href="#EFALSE"/>
</op>

<op id="isnotzero" semantic="condition">
  <title>isnotzero</title>
  <param id="isnotzero.value" type="inout">
    <title>value</title>
    <any/>
  </param>
  <raises xlink:href="#EFALSE"/>
</op>
```

```
    <op id="loop" semantic="other">
      <title>loop</title>
      <param id="loop.max">
        <title>max</title>
        <int/>
      </param>
      <param id="loop.count" type="inout">
        <title>count</title>
        <int/>
      </param>
      <raises xlink:href="#EFALSE"/>
    </op>

    <op id="incr">
      <title>incr</title>
      <param id="incr.value" type="inout">
        <title>value</title>
        <int/>
      </param>
    </op>

    <op id="decr">
      <title>decr</title>
      <param id="decr.value" type="inout">
        <title>value</title>
        <int/>
      </param>
    </op>

    <op id="strtoint" semantic="convert">
      <title>strtoint</title>
      <param id="strtoint.str">
        <title>str</title>
        <string/>
      </param>
      <return id="strtoint.return">
        <int/>
      </return>
    </op>

    <op id="inttostr" semantic="convert">
      <title>inttostr</title>
      <param id="inttostr.int">
        <title>int</title>
        <int/>
      </param>
      <return id="inttostr.return">
        <string/>
      </return>
    </op>

  </interface>

</system>
```

Anhang D

Stylesheet

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE xsl:stylesheet [

  <!ENTITY  base               "fix:float  |  fix:int | fix:char | fix:bool |
                               fix:octet | fix:any | fix:object |
                               fix:valuebase">

  <!ENTITY  template           "fix:sequence | fix:string | fix:fixed">

  <!ENTITY  simple             "&base; | &template; | fix:type">

  <!ENTITY  constr             "fix:struct | fix:union | fix:enum">

  <!ENTITY  datatype           "&simple; | &constr;">

  <!ENTITY  type_dcl           "fix:typedef | fix:struct | fix:union |
                               fix:enum | fix:native">

  <!ENTITY  param              "&base; | fix:string | fix:type">

  <!ENTITY  definition         "&type_dcl; | fix:const | fix:except |
                               fix:interface | fix:module | fix:valuetype">

  <!ENTITY  export             "&type_dcl; | fix:const | fix:except |
                               fix:attr | fix:op">

  <!ENTITY  cr                 "<xsl:text>&#xA;</xsl:text>">

]>

<!-- $Id: idl.xsl,v 1.11 2000/07/20 08:07:04 claude Exp $ -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0">

  <xsl:output method="text" encoding="iso-8859-1"
   media-type="application/x-idl"/>
  <xsl:strip-space elements="*"/>
```

```
<!-- mode="crossref" -->

<xsl:template match="*" mode="crossref">
  <xsl:apply-templates select="fix:title"/>
</xsl:template>

<!-- system -->

<xsl:template match="fix:system">
  <xsl:text>/*</xsl:text>&cr;
  <xsl:text>   System:  </xsl:text>
  <xsl:apply-templates select="fix:title"/>&cr;
  <xsl:text>   Address: </xsl:text>
  <xsl:apply-templates select="fix:address"/>&cr;
  <xsl:text>   Type:    </xsl:text>
  <xsl:apply-templates
   select="fix:dcom|fix:iiop|fix:local|fix:oncrpc|fix:rmi"/>&cr;
  &cr;
  <xsl:text>   THIS IS A GENERATED FILE. DO NOT EDIT!</xsl:text>&cr;
  <xsl:text> */</xsl:text>&cr;
  &cr;
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:for-each select="&definition;">
    <xsl:apply-templates select="."/><xsl:text>;</xsl:text>&cr;
  </xsl:for-each>
  &cr;
</xsl:template>

<!-- title|address -->

<xsl:template match="fix:title|fix:address">
  <xsl:value-of select="normalize-space(.)"/>
</xsl:template>

<!-- desc -->

<xsl:template match="fix:desc">
  &cr;
  <xsl:text>/* </xsl:text>
  <xsl:apply-templates/>&cr;
  <xsl:text> */</xsl:text>&cr;
</xsl:template>

<xsl:template match="fix:a">
  <xsl:apply-templates/>
  <xsl:text> (</xsl:text>
  <xsl:value-of select="@xlink:href"/>
  <xsl:text>)</xsl:text>
</xsl:template>

<!-- iiop -->

<xsl:template match="fix:iiop">
  <xsl:text>Corba</xsl:text>
</xsl:template>
```

```
<!-- dcom -->

<xsl:template match="fix:dcom">
  <xsl:text>DCOM</xsl:text>
</xsl:template>

<!-- rmi -->

<xsl:template match="fix:rmi">
  <xsl:text>RMI</xsl:text>
</xsl:template>

<!-- oncrpc -->

<xsl:template match="fix:oncrpc">
  <xsl:text>ONC RPC (Program </xsl:text>
  <xsl:value-of select="@program"/>
  <xsl:text>)</xsl:text>
</xsl:template>

<!-- local -->

<xsl:template match="fix:local">
  <xsl:text>Local (Filename </xsl:text>
  <xsl:value-of select="@filename"/>
  <xsl:text>)</xsl:text>
</xsl:template>

<!-- typedef -->

<xsl:template match="fix:typedef">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:text>typedef </xsl:text>
  <xsl:apply-templates select="&datatype;"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
  <xsl:apply-templates select="fix:array"/>
</xsl:template>

<!-- array -->

<xsl:template match="fix:array">
  <xsl:text>[</xsl:text>
  <xsl:value-of select="@count"/>
  <xsl:text>]</xsl:text>
</xsl:template>

<!-- const -->

<xsl:template match="fix:const">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:if test="not(fix:expr[@type='text/x-idl'
                             or @type='text/plain'
                             or @type='application/fix+xml'])">/* </xsl:if>
  <xsl:text>const </xsl:text>
```

```
    <xsl:apply-templates select="&datatype;"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="fix:title"/>
    <xsl:text> = </xsl:text>
    <xsl:choose>
      <xsl:when test="fix:expr[@type='text/x-idl'
                               or @type='text/plain'
                               or @type='application/fix+xml']">
        <xsl:apply-templates select="fix:expr"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>VALUE NOT SPECIFIED OR UNPARSABLE */</xsl:text>
        <xsl:message terminate="no">
Constant value not specified or unparsable!</xsl:message>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <!-- expr -->

  <xsl:template match="fix:expr">
    <xsl:choose>
      <xsl:when test="@type='application/fix+xml'">
        <xsl:apply-templates select="fix:type"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="@type='text/plain'">"</xsl:if>
        <xsl:choose>
          <xsl:when test="@xml:space='preserve'">
            <xsl:value-of select="."/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="normalize-space(.)"/>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:if test="@type='text/plain'">"</xsl:if>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <!-- except -->

  <xsl:template match="fix:except">
    <xsl:apply-templates select="fix:desc"/>
    &cr;
    <xsl:text>exception </xsl:text>
    <xsl:apply-templates select="fix:title"/>
    <xsl:text> {</xsl:text>&cr;
    <xsl:for-each select="fix:member">
      <xsl:apply-templates select="."/><xsl:text>;</xsl:text>&cr;
    </xsl:for-each>
    <xsl:text>}</xsl:text>
  </xsl:template>

  <!-- module -->

  <xsl:template match="fix:module">
```

```
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:text>module </xsl:text>
  <xsl:value-of select="fix:title"/>
  <xsl:text> {</xsl:text>&cr;
  <xsl:for-each select="&definition;">
    <xsl:apply-templates select="."/><xsl:text>;</xsl:text>&cr;
  </xsl:for-each>
  &cr;
  <xsl:text>}</xsl:text>
</xsl:template>

<!-- interface -->

<xsl:template match="fix:interface">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:if test="@abstract='true'">
    <xsl:text>abstract </xsl:text>
  </xsl:if>
  <xsl:text>interface </xsl:text>
  <xsl:value-of select="fix:title"/>
  <xsl:text> </xsl:text>
  <xsl:if test="fix:inherits">
    <xsl:text>: </xsl:text>
    <xsl:for-each select="fix:inherits">
      <xsl:apply-templates select="."/>
      <xsl:if test="following-sibling::fix:inherits">, </xsl:if>
    </xsl:for-each>
  </xsl:if>
  <xsl:if test="&export;">
    <xsl:text> {</xsl:text>&cr;
    <xsl:for-each select="&export;">
      <xsl:apply-templates select="."/><xsl:text>;</xsl:text>&cr;
    </xsl:for-each>
    &cr;
    <xsl:text>}</xsl:text>
  </xsl:if>
</xsl:template>

<!-- inherits|supports|type|raises -->

<xsl:template match="fix:inherits|fix:supports|fix:type|fix:raises">
  <xsl:variable name="link" select="@xlink:href"/>
  <xsl:apply-templates select="//*[@id=substring($link,2)]" mode="crossref"/>
</xsl:template>

<!-- valuetype -->

<xsl:template match="fix:valuetype">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:choose>
    <xsl:when test="@abstract='true'">abstract </xsl:when>
    <xsl:when test="@custom='true'">custom </xsl:when>
  </xsl:choose>
  <xsl:text>valuetype </xsl:text>
```

```
  <xsl:value-of select="fix:title"/>
  <xsl:text> </xsl:text>
  <xsl:if test="fix:inherits">
    <xsl:if test="@truncatable='true'">truncatable </xsl:if>
    <xsl:text>: </xsl:text>
    <xsl:for-each select="fix:inherits">
      <xsl:apply-templates select="."/>
      <xsl:if test="following-sibling::fix:inherits">, </xsl:if>
    </xsl:for-each>
  </xsl:if>
  <xsl:if test="fix:supports">
    <xsl:text> supports </xsl:text>
    <xsl:for-each select="fix:supports">
      <xsl:apply-templates select="."/>
      <xsl:if test="following-sibling::fix:supports">, </xsl:if>
    </xsl:for-each>
  </xsl:if>
  <xsl:choose>
    <xsl:when test="&export;|fix:statemember|fix:factory">
      <xsl:text> {</xsl:text>&cr;
      <xsl:for-each select="&export;|fix:statemember|fix:factory">
        <xsl:apply-templates select="."/>
        <xsl:text>;</xsl:text>&cr;
      </xsl:for-each>
      &cr;
      <xsl:text>}</xsl:text>
    </xsl:when>
    <xsl:when test="fix:boxed">
      <xsl:apply-templates select="fix:boxed"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<!-- boxed -->

<xsl:template match="fix:boxed">
  <xsl:apply-templates/>
</xsl:template>

<!-- statember -->

<xsl:template match="fix:statemember">
  <xsl:apply-templates select="fix:desc"/>
  <xsl:apply-templates select="@type"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="&datatype;"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
  <xsl:apply-templates select="fix:array"/>
</xsl:template>

<!-- factory -->

<xsl:template match="fix:factory">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:text>factory </xsl:text>
```

```
  <xsl:apply-templates select="fix:title"/>
  <xsl:text>(</xsl:text>
  <xsl:if test="fix:param">
    <xsl:for-each select="fix:param">
      <xsl:apply-templates select="."/>
      <xsl:if test="following-sibling::fix:param">, </xsl:if>
    </xsl:for-each>
  </xsl:if>
  <xsl:text>)</xsl:text>
</xsl:template>

<!-- float -->

<xsl:template match="fix:float">
  <xsl:choose>
    <xsl:when test="@type='float'">float</xsl:when>
    <xsl:when test="@type='long_double'">long double</xsl:when>
    <xsl:otherwise>double</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- int -->

<xsl:template match="fix:int">
  <xsl:if test="@unsigned='true'">unsigned </xsl:if>
  <xsl:choose>
    <xsl:when test="@type='short'">short</xsl:when>
    <xsl:when test="@type='long_long'">long long</xsl:when>
    <xsl:otherwise>long</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- char -->

<xsl:template match="fix:char">
  <xsl:if test="@wide='true'">w</xsl:if>
  <xsl:text>char</xsl:text>
</xsl:template>

<!-- bool -->

<xsl:template match="fix:bool">
  <xsl:text>boolean</xsl:text>
</xsl:template>

<!-- octet -->

<xsl:template match="fix:octet">
  <xsl:text>octet</xsl:text>
</xsl:template>

<!-- any -->

<xsl:template match="fix:any">
  <xsl:text>any</xsl:text>
</xsl:template>
```

```
<!-- object -->

<xsl:template match="fix:object">
  <xsl:text>Object</xsl:text>
</xsl:template>

<!-- valuebase -->

<xsl:template match="fix:valuebase">
  <xsl:text>ValueBase</xsl:text>
</xsl:template>

<!-- sequence -->

<xsl:template match="fix:sequence">
  <xsl:text>sequence&lt;</xsl:text>
  <xsl:apply-templates/>
  <xsl:if test="@count">
    <xsl:text>, </xsl:text>
    <xsl:value-of select="@count"/>
  </xsl:if>
  <xsl:text>&gt;</xsl:text>
</xsl:template>

<!-- string -->

<xsl:template match="fix:string">
  <xsl:if test="@wide='true'">w</xsl:if>
  <xsl:text>string</xsl:text>
  <xsl:if test="@count">
    <xsl:text>&lt;</xsl:text>
    <xsl:value-of select="@count"/>
    <xsl:text>&gt;</xsl:text>
  </xsl:if>
</xsl:template>

<!-- fixed -->

<xsl:template match="fix:fixed">
  <xsl:text>fixed&lt;</xsl:text>
  <xsl:value-of select="@digits"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="@scale"/>
  <xsl:text>&gt;</xsl:text>
</xsl:template>

<xsl:template match="fix:const/fix:fixed">
  <xsl:text>fixed</xsl:text>
</xsl:template>

<!-- struct -->

<xsl:template match="fix:struct">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:text>struct </xsl:text>
  <xsl:apply-templates select="fix:title"/>
```

```
  <xsl:text> {</xsl:text>&cr;
  <xsl:for-each select="fix:member">
    <xsl:apply-templates select="."/><xsl:text>;</xsl:text>&cr;
  </xsl:for-each>
  <xsl:text>}</xsl:text>
</xsl:template>

<!-- member -->

<xsl:template match="fix:member">
  <xsl:apply-templates select="fix:desc"/>
  <xsl:apply-templates select="&datatype;"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
  <xsl:apply-templates select="fix:array"/>
</xsl:template>

<xsl:template match="fix:union">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:text>union </xsl:text>
  <xsl:apply-templates select="fix:title"/>
  <xsl:text> switch (</xsl:text>
  <xsl:apply-templates select="fix:int|fix:char|fix:bool|fix:enum|fix:type"/>
  <xsl:text>) {</xsl:text>&cr;
  <xsl:for-each select="fix:case|fix:default">
    <xsl:apply-templates select="."/><xsl:text>;</xsl:text>&cr;
  </xsl:for-each>
  <xsl:text>}</xsl:text>
</xsl:template>

<!-- case -->

<xsl:template match="fix:case">
  <xsl:apply-templates select="fix:desc"/>
  <xsl:text>case </xsl:text>
  <xsl:apply-templates select="fix:expr"/>
  <xsl:text> : </xsl:text>
  <xsl:apply-templates select="&datatype;"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
</xsl:template>

<!-- default -->

<xsl:template match="fix:default">
  <xsl:apply-templates select="fix:desc"/>
  <xsl:text>default : </xsl:text>
  <xsl:apply-templates select="&datatype;"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
</xsl:template>

<!-- enum -->

<xsl:template match="fix:enum">
  <xsl:apply-templates select="fix:desc"/>
```

```
  &cr;
  <xsl:text>enum </xsl:text>
  <xsl:apply-templates select="fix:title"/>
  <xsl:text> {</xsl:text>&cr;
  <xsl:for-each select="fix:enumerator">
    <xsl:apply-templates select="."/>
    <xsl:if test="following-sibling::fix:enumerator">,</xsl:if>&cr;
  </xsl:for-each>
  <xsl:text>}</xsl:text>
</xsl:template>

<!-- enumerator -->

<xsl:template match="fix:enumerator">
  <xsl:apply-templates select="fix:desc"/>
  <xsl:apply-templates select="fix:title"/>
</xsl:template>

<!-- native -->

<xsl:template match="fix:native">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:text>native </xsl:text>
  <xsl:apply-templates select="fix:title"/>
</xsl:template>

<!-- attr -->

<xsl:template match="fix:attr">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:if test="@readonly='true'">readonly </xsl:if>
  <xsl:text>attribute </xsl:text>
  <xsl:apply-templates select="&param;"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
</xsl:template>

<!-- op -->

<xsl:template match="fix:op">
  <xsl:apply-templates select="fix:desc"/>
  &cr;
  <xsl:if test="@oneway='true'">oneway</xsl:if>
  <xsl:choose>
    <xsl:when test="fix:return">
      <xsl:apply-templates select="fix:return"/>
    </xsl:when>
    <xsl:otherwise>void</xsl:otherwise>
  </xsl:choose>
  <xsl:text> </xsl:text>
  <xsl:apply-templates select="fix:title"/>
  <xsl:text>(</xsl:text>
  <xsl:if test="fix:param">
    <xsl:for-each select="fix:param">
      <xsl:apply-templates select="."/>
```

```
        <xsl:if test="following-sibling::fix:param">, </xsl:if>
      </xsl:for-each>
    </xsl:if>
    <xsl:text>)</xsl:text>
    <xsl:if test="fix:raises">
      <xsl:text> raises (</xsl:text>
      <xsl:for-each select="fix:raises">
        <xsl:apply-templates select="."/>
        <xsl:if test="following-sibling::fix:raises">, </xsl:if>
      </xsl:for-each>
      <xsl:text>)</xsl:text>
    </xsl:if>
    <xsl:if test="fix:context">
      <xsl:text> context (</xsl:text>
      <xsl:for-each select="fix:context">
        <xsl:apply-templates select="."/>
        <xsl:if test="following-sibling::fix:context">, </xsl:if>
      </xsl:for-each>
      <xsl:text>)</xsl:text>
    </xsl:if>
  </xsl:template>

  <!-- param -->

  <xsl:template match="fix:param">
    <xsl:choose>
      <xsl:when test="@type='out'">out </xsl:when>
      <xsl:when test="@type='inout'">inout </xsl:when>
      <xsl:otherwise>in </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates select="&param;"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="fix:title"/>
  </xsl:template>

  <!-- return -->

  <xsl:template match="fix:return">
    <xsl:apply-templates select="&param;"/>
  </xsl:template>

  <!-- context -->

  <xsl:template match="fix:context">
    <xsl:text>"</xsl:text>
    <xsl:value-of select="normalize-space(.)"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Anhang E

Beispiele

E.1 Quellsystem Werk 1

IDL

```
// $Id: werk1.idl,v 1.11 2000/07/20 08:31:26 claude Exp $
// vim:showmatch:fo=tcqor:cindent:tw=75
//
// Schnittstelle von Werk 1 (http://werk1.example.com)
// Interface of plant 1 (http://werk1.example.com/en/)

const string name = "Werk 1";
const version version = 42;

// Bauteile werden über eine eindeutige Teilenummer referenziert

typedef long partno_t

// mögliche Farben

typedef enum color { red, green, yellow, blue } color_t;

typedef sequence<partno_t> partno_sequence_t;

interface parts {

  exception example_e {
    long internal_error;
  };

  readonly attribute long num_parts;

  partno_sequence_t all_parts(in string department)
    raises (example_e);

  void get_part(in partno_t partno, out string name, out color_t color)
    raises (example_e);

  void upd_part(in partno_t partno, in string name, in color_t color)
    raises (example_e);
```

```
  void add_part(in partno_t partno, in string name, in color_t color)
    raises (example_e);

  void del_part(in partno_t partno)
    raises (example_e);

  void init() raises (example_e);

};
```

FIX

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE system SYSTEM "http://dcx.com/fix/fix.dtd">

<!-- $Id: werk1.xml,v 1.13 2000/07/20 08:33:18 claude Exp $ -->

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <!-- Fig: Verwendung der allgemeinen Elemente -->
  <title>Werk 1</title>
  <desc xml:lang="de">
    Schnittstelle von <a xlink:href="http://werk1.example.com">Werk 1</a>
  </desc>
  <desc xml:lang="en">
    Interface of <a xlink:href="http://werk1.example.com">plant 1</a>
  </desc>
  <address>werk1.example.com</address>
  <oncrpc program="600000000" version="1"/>

  <const id="name">
    <title>name</title>
    <string/>
    <expr type="text/plain" xml:space="preserve">Werk 1</expr>
  </const>

  <const>
    <title>version</title>
    <int/>
    <expr type="text/mathml">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <cn>42</cn>
      </math>
    </expr>
  </const>

  <typedef id="partno_t">
    <!-- Fig: Verwendung der allgemeinen Elemente -->
    <title>partno_t</title>
    <desc>
      Bauteile werden über eine
      <html:b xmlns:html="http://www.w3.org/1999/xhtml">eindeutige</html:b>
      Teilenummer referenziert
    </desc>
    <int/>
```

```
</typedef>

<typedef id="color_t">
  <title>color_t</title>
  <desc>
    mögliche Farben
  </desc>
  <enum id="color">
    <title>color</title>
    <enumerator id="color.red">
      <title>red</title>
    </enumerator>
    <enumerator id="color.green">
      <title>green</title>
    </enumerator>
    <enumerator id="color.yellow">
      <title>yellow</title>
    </enumerator>
    <enumerator id="color.blue">
      <title>blue</title>
    </enumerator>
  </enum>
</typedef>

<typedef id="partno_sequence_t">
  <title>partno_sequence_t</title>
  <sequence>
    <type xlink:href="#partno_t"/>
  </sequence>
</typedef>

<interface id="parts">
  <title>parts</title>

  <except id="example_e">
    <title>example_e</title>
    <member id="internal_error">
      <title>internal_error</title>
      <int/>
    </member>
  </except>

  <attr id="num_parts" readonly="true">
    <title>num_parts</title>
    <int/>
  </attr>

  <op id="all_parts" semantic="read">
    <title>all_parts</title>
    <param id="all_parts.department">
      <title>department</title>
      <string/>
    </param>
    <return id="all_parts.return">
      <type xlink:href="#partno_sequence_t"/>
    </return>
    <raises xlink:href="#example_e"/>
```

```
</op>

<op id="get_part" semantic="read">
  <title>get_part</title>
  <param id="get_part.partno">
    <title>partno</title>
    <type xlink:href="#partno_t"/>
  </param>
  <param id="get_part.name" type="out">
    <title>name</title>
    <string/>
  </param>
  <param id="get_part.color" type="out">
    <title>color</title>
    <type xlink:href="#color_t"/>
  </param>
  <raises xlink:href="#example_e"/>
</op>

<op id="upd_part" semantic="update">
  <title>upd_part</title>
  <param id="upd_part.partno">
    <title>partno</title>
    <type xlink:href="#partno_t"/>
  </param>
  <param id="upd_part.name">
    <title>name</title>
    <string/>
  </param>
  <param id="upd_part.color">
    <title>color</title>
    <type xlink:href="#color_t"/>
  </param>
  <raises xlink:href="#example_e"/>
</op>

<op id="add_part" semantic="add">
  <title>add_part</title>
  <param id="add_part.partno">
    <title>partno</title>
    <type xlink:href="#partno_t"/>
  </param>
  <param id="add_part.name">
    <title>name</title>
    <string/>
  </param>
  <param id="add_part.color">
    <title>color</title>
    <type xlink:href="#color_t"/>
  </param>
  <raises xlink:href="#example_e"/>
</op>

<op id="del_part" semantic="delete">
  <title>del_part</title>
  <param id="del_part.partno">
    <title>partno</title>
```

```
        <type xlink:href="#partno_t"/>
      </param>
      <raises xlink:href="#example_e"/>
    </op>

    <op id="init">
      <title>init</title>
      <raises xlink:href="#example_e"/>
      <effect id="neustart">
        <title>Neustart</title>
        <desc xml:lang="de">
          Führt einen Neustart durch
        </desc>
      </effect>
    </op>

  </interface>

</system>
```

E.2 Quellsystem Werk 2

IDL

```
// $Id: werk2.idl,v 1.11 2000/07/20 08:33:54 claude Exp $
// vim:showmatch:fo=tcqor:cindent:tw=75

typedef string<6> partno;

typedef sequence<partno> parts;

typedef struct dimension {
  float x;
  float y;
  float z;
} dimension_t;

interface parts {

  // Fehler werden jeweils durch 0, NULL oder FALSE zurückgegeben

  parts all_parts();

  boolean get_part(in partno no, out string name, out dimension_t dim);

  boolean upd_part(in partno no, in string name, in dimension_t dim);

  partno add_part(in string name, in dimension_t dim);

  boolean del_part(in partno no);

  void init();

}
```

FIX

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE system SYSTEM "http://dcx.com/fix/fix.dtd">

<!-- $Id: werk2.xml,v 1.12 2000/07/20 08:35:23 claude Exp $ -->

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <!-- Systembeschreibung -->
  <title>Werk 2</title>
  <address>werk2.example.com</address>
  <iiop port="683"/>

  <typedef id="partno">
    <title>partno</title>
    <string count="6"/>
  </typedef>

  <typedef id="parts">
    <title>parts</title>
    <sequence>
      <type xlink:href="#partno"/>
    </sequence>
  </typedef>

  <typedef id="dimension_t">
    <title>dimension_t</title>
    <struct id="dimension">
      <title>dimension</title>
      <member id="dimension.x">
        <title>x</title>
        <float type="float"/>
      </member>
      <member id="dimension.y">
        <title>y</title>
        <float type="float"/>
      </member>
      <member id="dimension.z">
        <title>z</title>
        <float type="float"/>
      </member>
    </struct>
  </typedef>

  <interface id="interface.parts">
    <title>parts</title>
    <desc xml:lang="de">
      Fehler werden jeweils durch 0, NULL oder FALSE zurückgegeben
    </desc>

    <op id="all_parts">
      <title>all_parts</title>
      <return id="all_parts.return">
        <type xlink:href="#parts"/>
      </return>
    </op>
```

```
<op id="get_part">
  <title>get_part</title>
  <param id="get_part.no">
    <title>no</title>
    <type xlink:href="#partno"/>
  </param>
  <param id="get_part.name" type="out">
    <title>name</title>
    <string/>
  </param>
  <param id="get_part.dim" type="out">
    <title>dim</title>
    <type xlink:href="#dimension_t"/>
  </param>
  <return id="get_part.return">
    <bool/>
  </return>
</op>

<op id="upd_part">
  <title>upd_part</title>
  <param id="upd_part.no">
    <title>no</title>
    <type xlink:href="#partno"/>
  </param>
  <param id="upd_part.name">
    <title>name</title>
    <string/>
  </param>
  <param id="upd_part.dim">
    <title>dim</title>
    <type xlink:href="#dimension_t"/>
  </param>
  <return id="upd_part.return">
    <bool/>
  </return>
</op>

<op id="add_part">
  <title>add_part</title>
  <param id="add_part.name">
    <title>name</title>
    <string/>
  </param>
  <param id="add_part.dim">
    <title>dim</title>
    <type xlink:href="#dimension_t"/>
  </param>
  <return id="add_part.return">
    <type xlink:href="#partno"/>
  </return>
  <effect id="hinzufuegen">
    <title>Element hinzufügen</title>
    <desc xml:lang="de">
      Das in den Parametern übergebene Element
      wurde hinzugefügt.
```

```
    </desc>
  </effect>
  <graph type="wrapper">
    <node xlink:href="#add_part" xlink:role="add_part.target"
     instance="fix:target"/>
    <node xlink:href="#add_part.name" xlink:role="add_part.name.target"
     instance="fix:target"/>
    <node xlink:href="#add_part.dim" xlink:role="add_part.dim.target"
     instance="fix:target"/>
    <node xlink:href="#add_part.return"
     xlink:role="add_part.return.target" instance="fix:target"/>

    <node xlink:href="#add_part" xlink:role="add_part"/>
    <node xlink:href="#add_part.name" xlink:role="add_part.name"/>
    <node xlink:href="#add_part.dim" xlink:role="add_part.dim"/>
    <node xlink:href="#add_part.return" xlink:role="add_part.return"/>

    <node xlink:href="#del_part" xlink:role="del_part"/>
    <node xlink:href="#del_part.no" xlink:role="del_part.no"/>

    <node xlink:href="stdlib.xml#isnotzero" xlink:role="isnotzero"/>
    <node xlink:href="stdlib.xml#isnotzero.value"
     xlink:role="isnotzero.value"/>
    <node xlink:href="global.xml#EINTERNAL" xlink:role="EINTERNAL"/>

    <dep xlink:from="add_part.name.target" xlink:to="add_part.name"/>
    <dep xlink:from="add_part.dim.target" xlink:to="add_part.dim"/>
    <dep xlink:from="add_part.return" xlink:to="add_part.return.target"/>

    <dep xlink:from="add_part.return" xlink:to="del_part.no"/>
    <dep xlink:from="add_part" xlink:to="del_part"
     xlink:role="compensation"/>
    <dep xlink:from="del_part" xlink:to="add_part.target"/>

    <dep xlink:from="add_part.return" xlink:to="isnotzero.value"/>
    <dep xlink:from="isnotzero" xlink:to="add_part.target"/>
    <dep xlink:from="isnotzero" xlink:to="EINTERNAL"
     xlink:role="exception"/>
  </graph>
</op>

<op id="del_part">
  <!-- Fig: Nachbildung von Exceptions. -->
  <title>del_part</title>
  <param id="no">
    <title>no</title>
    <type xlink:href="#partno"/>
  </param>
  <return id="del_part.return">
    <bool/>
  </return>
  <effect id="loeschen">
    <title>Element Löschen</title>
    <desc xml:lang="de">
      Das Element mit der angegebenen Artikelnummer
      wurde gelöscht.
    </desc>
```

```
      </effect>
      <graph type="wrapper">
        <node xlink:href="#del_part" xlink:role="del_part.target"
         instance="fix:target"/>
        <node xlink:href="#del_part.no" xlink:role="del_part.no.target"
         instance="fix:target"/>
        <node xlink:href="#del_part.return"
         xlink:role="del_part.return.target" instance="fix:target"/>
        <node xlink:href="#del_part.no" xlink:role="del_part.no"/>
        <node xlink:href="#del_part.return" xlink:role="del_part.return"/>
        <node xlink:href="stdlib.xml#istrue" xlink:role="istrue"/>
        <node xlink:href="stdlib.xml#istrue.value" xlink:role="istrue.value"/>
        <node xlink:href="global.xml#ENOTDELETED" xlink:role="ENOTDELETED"/>

        <dep xlink:from="del_part.no.target" xlink:to="del_part.no"/>
        <dep xlink:from="del_part.return" xlink:to="del_part.return.target"/>
        <dep xlink:from="del_part.return" xlink:to="istrue.boolean"/>
        <dep xlink:from="istrue" xlink:to="del_part"/>
        <dep xlink:from="istrue" xlink:to="ENOTDELETED"
         xlink:role="exception"/>
      </graph>
    </op>

    <op id="init">
      <title>init</title>
      <effect id="reboot">
        <title>Reboot</title>
        <desc xml:lang="de">
          Das System führt einen kompletten Reboot aus
        </desc>
      </effect>
    </op>

  </interface>

</system>
```

E.3 Quellsystem Lieferant

IDL

```
// $Id: lieferant.idl,v 1.9 2000/07/20 08:37:04 claude Exp $
// vim:showmatch:fo=tcqor:cindent:tw=75


interface lieferant {

  exception ENOTFOUND {};

  typedef fixed<7,2> price;

  typedef enum currency_e { DM, Euro, Dollar } currency_t;

  attribute currency_t currency;
```

```
  price get_price(in long no) raises (ENOTFOUND);
  void set_price(in long no, in price price);

}
```

FIX

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE system SYSTEM "http://dcx.com/fix/fix.dtd">

<!-- $Id: lieferant.xml,v 1.8 2000/07/20 08:37:41 claude Exp $ -->

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>Lieferant</title>
  <address>lieferant.example.com</address>
  <iiop port="683"/>

  <interface>
    <title>lieferant</title>

    <except id="ENOTFOUND">
      <title>ENOTFOUND</title>
    </except>

    <typedef id="price">
      <title>price</title>
      <fixed digits="7" scale="2"/>
    </typedef>

    <typedef id="currency_t">
      <title>currency_t</title>
      <enum id="currency_e">
        <title>currency_e</title>
        <enumerator id="currency_e.DM">
          <title>DM</title>
        </enumerator>
        <enumerator id="currency_e.Euro">
          <title>Euro</title>
        </enumerator>
        <enumerator id="currency_e.Dollar">
          <title>Dollar</title>
        </enumerator>
      </enum>
    </typedef>

    <const>
      <title>testconst</title>
      <type xlink:href="#currency_t"/>
      <expr type="application/fix+xml">
        <type xlink:href="#currency_e.Dollar"/>
      </expr>
    </const>

    <attr id="currency">
      <title>currency</title>
```

```
      <type xlink:href="#currency_t"/>
    </attr>

    <op id="get_price">
      <title>get_price</title>
      <param id="get_price.no">
        <title>no</title>
        <int/>
      </param>
      <return id="get_price.return">
        <type xlink:href="#price"/>
      </return>
      <raises xlink:href="#ENOTFOUND"/>
    </op>

    <op id="set_price">
      <title>set_price</title>
      <param id="set_price.no">
        <title>no</title>
        <int/>
      </param>
      <param id="set_price.price">
        <title>price</title>
        <type xlink:href="#price"/>
      </param>
    </op>

  </interface>
</system>
```

E.4 Hilfssystem

IDL

```
// $Id: helper.idl,v 1.10 2000/07/20 08:38:58 claude Exp $
// vim:showmatch:fo=tcqor:cindent:tw=75

interface helper {

  const long null = 0;
  const long eins = 1;
  const long zwei = 2;
  const string kfz = "kfz";
  const string rot = "rot";
  const string gruen = "grün";
  const string gelb = "gelb";

  void m2inch(in float m, out float inch);
  string col2str(in color_t color);

}
```

FIX

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

```
<!DOCTYPE system SYSTEM "http://dcx.com/fix/fix.dtd">

<!-- $Id: helper.xml,v 1.13 2000/07/20 08:40:08 claude Exp $ -->

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>helper</title>
  <address>server.example.com</address>
  <iiop port="683"/>

  <interface>
    <title>helper</title>

    <const id="null">
      <title>null</title>
      <int/>
      <expr type="text/x-idl">
        0
      </expr>
    </const>

    <const id="eins">
      <title>eins</title>
      <int/>
      <expr type="text/x-idl">
        1
      </expr>
    </const>

    <const id="zwei">
      <title>zwei</title>
      <int/>
      <expr type="text/x-idl">
        2
      </expr>
    </const>

    <const id="kfz">
      <title>kfz</title>
      <string/>
      <expr type="text/plain" xml:space="preserve">kfz</expr>
    </const>

    <const id="rot">
      <title>rot</title>
      <string/>
      <expr type="text/plain" xml:space="preserve">rot</expr>
    </const>

    <const id="gruen">
      <title>gruen</title>
      <string/>
      <expr type="text/plain" xml:space="preserve">grün</expr>
    </const>

    <const id="gelb">
```

```
      <title>gelb</title>
      <string/>
      <expr type="text/plain" xml:space="preserve">gelb</expr>
    </const>

    <op id="m2inch">
      <title>m2inch</title>
      <param id="m2inch.m">
        <title>m</title>
        <float type="float"/>
      </param>
      <param id="m2inch.inch" type="out">
        <title>inch</title>
        <float type="float"/>
      </param>
    </op>

    <op id="col2str">
      <title>col2str</title>
      <param id="col2str.color">
        <title>color</title>
        <type xlink:href="werk1.xml#color_t"/>
      </param>
      <return id="col2str.return">
        <string/>
      </return>
    </op>

  </interface>

</system>
```

E.5 Zielsystem

IDL

```
// $Id: global.idl,v 1.14 2000/07/20 08:41:26 claude Exp $
// vim:showmatch:fo=tcqor:cindent:tw=75

interface global {

  exception EINTERNAL {
    long no;
  };
  exception ENOTFOUND {};
  exception ENOTDELETED {};

  typedef sequence<long> partlist;

  typedef enum currency_e { DM, Euro, Dollar } currency_t;

  typedef fixed<7,2> price_t;

  attribute currency_t currency;
```

```
  partlist all_parts() raises(EINTERNAL);

  void get_part(in long no, out string name, out string color,
      out price_t price) raises (ENOTFOUND, EINTERNAL);
  void upd_part(in long no, in string name, in string color)
      raises (ENOTFOUND, EINTERNAL);
  long add_part(in string name, in string color) raises (EINTERNAL);
  void del_part(in long no) raises (ENOTFOUND, EINTERNAL, ENOTDELETED);

  string get_color(in long no) raises (ENOTFOUND, EINTERNAL);
  string get_name(in long no) raises (ENOTFOUND, EINTERNAL);

  void get_size(in long no, out float x, out float y, out float z)
    raises (ENOTFOUND, EINTERNAL);
  void get_size_inch(in long no, out float x, out float y, out float z)
    raises (ENOTFOUND, EINTERNAL);

  price_t get_price_euro(in long no) raises (ENOTFOUND, EINTERNAL);
  void set_price_euro(in long no, in price_t price) raises (EINTERNAL);

  void init() raises(EINTERNAL);

};
```

FIX

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE system SYSTEM "http://dcx.com/fix/fix.dtd">

<!-- $Id: global.xml,v 1.10 2000/07/20 08:43:38 claude Exp $ -->

<system xmlns="http://dcx.com/fix" xmlns:fix="http://dcx.com/fix"
 xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>Global</title>
  <address>global.example.com</address>
  <iiop port="683"/>

  <interface>
    <title>global</title>

    <except id="EINTERNAL">
      <title>EINTERNAL</title>
      <member id="EINTERNAL.no">
        <title>no</title>
        <int/>
      </member>
    </except>

    <except id="ENOTFOUND">
      <title>ENOTFOUND</title>
    </except>

    <except id="ENOTDELETED">
      <title>ENOTDELETED</title>
    </except>
```

```
<typedef id="partlist">
  <title>partlist</title>
  <sequence>
    <int/>
  </sequence>
</typedef>

<typedef id="currency_t">
  <title>currency_t</title>
  <enum id="currency_e">
    <title>currency_e</title>
    <enumerator>
      <title>DM</title>
    </enumerator>
    <enumerator>
      <title>Euro</title>
    </enumerator>
    <enumerator>
      <title>Dollar</title>
    </enumerator>
  </enum>
</typedef>

<typedef id="price_t">
  <title>price_t</title>
  <fixed digits="7" scale="2"/>
</typedef>

<attr id="currency">
  <title>currency</title>
  <type xlink:href="#currency_t"/>
  <graph xlink:role="read">
    <node xlink:href="#currency" xlink:role="currency"/>
    <node xlink:href="lieferant.xml#currency"
     xlink:role="lieferant.currency"/>
    <dep xlink:from="lieferant.currency" xlink:to="currency"/>
  </graph>
  <graph xlink:role="write">
    <node xlink:href="#currency" xlink:role="currency"
     instance="fix:target"/>
    <node xlink:href="#currency" xlink:role="currencysource"/>
    <node xlink:href="lieferant.xml#currency"
     xlink:role="lieferant.currency"/>
    <dep xlink:from="currencysource" xlink:to="lieferant.currency"/>
    <dep xlink:from="lieferant.currency" xlink:to="currency"/>
  </graph>
</attr>

<op id="all_parts" semantic="read">
  <title>all_parts</title>
  <return id="all_parts.return">
    <type xlink:href="#partlist"/>
  </return>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#all_parts.return" xlink:role="all_parts.return"/>
    <node xlink:href="werk1.xml#all_parts.department"
```

```
     xlink:role="werk1.all_parts.department"/>
    <node xlink:href="werk1.xml#all_parts.return"
     xlink:role="werk1.all_parts.return"/>
    <node xlink:href="helper.xml#kfz" xlink:role="kfz"/>

    <dep xlink:from="kfz" xlink:to="werk1.all_parts.department"/>
    <dep xlink:from="werk1.all_parts.return" xlink:to="all_parts.return"/>
  </graph>
</op>

<op id="get_part" semantic="read">
  <title>get_part</title>
  <param id="get_part.no">
    <title>no</title>
    <int/>
  </param>
  <param id="get_part.name" type="out">
    <title>name</title>
    <string/>
  </param>
  <param id="get_part.color" type="out">
    <title>color</title>
    <string/>
  </param>
  <param id="get_part.price" type="out">
    <title>price</title>
    <type xlink:href="#price_t"/>
  </param>
  <raises xlink:href="#ENOTFOUND"/>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#get_part.no" xlink:role="global.no"/>
    <node xlink:href="#get_part.name" xlink:role="global.name"/>
    <node xlink:href="#get_part.color" xlink:role="global.color"/>
    <node xlink:href="#get_part.price" xlink:role="global.price"/>
    <node xlink:href="werk1.xml#get_part.partno"
     xlink:role="werk1.partno"/>
    <node xlink:href="werk1.xml#get_part.name"
     xlink:role="werk1.name"/>
    <node xlink:href="werk1.xml#get_part.color"
     xlink:role="werk1.color"/>
    <node xlink:href="helper.xml#col2str.color"
     xlink:role="col2str.color"/>
    <node xlink:href="helper.xml#col2str.return"
     xlink:role="col2str.return"/>
    <node xlink:href="lieferant.xml#get_price.no"
     xlink:role="lieferant.no"/>
    <node xlink:href="lieferant.xml#get_price.return"
     xlink:role="lieferant.price"/>
    <node xlink:href="helper.xml#null" xlink:role="null"/>

    <dep xlink:from="global.no" xlink:to="werk1.partno"/>
    <dep xlink:from="global.no" xlink:to="lieferant.no"/>
    <dep xlink:from="werk1.name" xlink:to="global.name"/>
    <dep xlink:from="werk1.color" xlink:to="col2str.color"/>
    <dep xlink:from="col2str.return" xlink:to="global.color"/>
    <dep xlink:from="lieferant.price" xlink:to="global.price"/>
```

```
    <dep xlink:from="null" xlink:to="global.price"
     completion="optional"/>
  </graph>
</op>

<op id="upd_part" semantic="update">
  <title>update</title>
  <param id="upd_part.no">
    <title>no</title>
    <int/>
  </param>
  <param id="upd_part.name">
    <title>name</title>
    <string/>
  </param>
  <param id="upd_part.color">
    <title>color</title>
    <string/>
  </param>
  <raises xlink:href="#ENOTFOUND"/>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#upd_part" xlink:role="global.upd_part"/>
    <node xlink:href="#upd_part.no" xlink:role="global.upd_part.no"/>
    <node xlink:href="#upd_part.name" xlink:role="global.upd_part.name"/>
    <node xlink:href="#upd_part.color" xlink:role="global.upd_part.color"/>

    <node xlink:href="werk1.xml#upd_part" xlink:role="werk1.upd_part"/>
    <node xlink:href="werk1.xml#upd_part.partno"
     xlink:role="werk1.upd_part.partno"/>
    <node xlink:href="werk1.xml#upd_part.name"
     xlink:role="werk1.upd_part.name"/>
    <node xlink:href="werk1.xml#upd_part.color"
     xlink:role="werk1.upd_part.color"/>

    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.int"
     xlink:role="inttostr.int"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.return"
     xlink:role="inttostr.return"/>

    <node xlink:href="werk2.xml#get_part.no"
     xlink:role="werk2.get_part.no"/>
    <node xlink:href="werk2.xml#get_part.dim"
     xlink:role="werk2.get_part.dim"/>

    <node xlink:href="werk2.xml#upd_part" xlink:role="werk2.upd_part"/>
    <node xlink:href="werk2.xml#upd_part.no"
     xlink:role="werk2.upd_part.no"/>
    <node xlink:href="werk2.xml#upd_part.name"
     xlink:role="werk2.upd_part.name"/>
    <node xlink:href="werk2.xml#upd_part.dim"
     xlink:role="werk2.upd_part.dim"/>

    <dep xlink:from="global.upd_part.no" xlink:to="werk1.upd_part.partno"/>
    <dep xlink:from="global.upd_part.name" xlink:to="werk1.upd_part.name"/>
    <dep xlink:from="global.upd_part.color"
     xlink:to="werk1.upd_part.color"/>
```

```
    <dep xlink:from="werk1.upd_part" xlink:to="global.upd_part"
     completion="required"/>

    <dep xlink:from="global.upd_part.no" xlink:to="inttostr.int"/>
    <dep xlink:from="global.upd_part.name" xlink:to="werk2.upd_part.name"/>
    <dep xlink:from="inttostr.return" xlink:to="werk2.get_part.no"/>
    <dep xlink:from="inttostr.return" xlink:to="werk2.upd_part.no"/>
    <dep xlink:from="werk2.get_part.dim" xlink:to="werk2.upd_part.dim"/>
    <dep xlink:from="werk2.upd_part" xlink:to="global.upd_part"
     completion="required"/>

  </graph>
</op>

<op id="add_part" semantic="add">
  <title>add_part</title>
  <param id="add_part.name">
    <title>name</title>
    <string/>
  </param>
  <param id="add_part.color">
    <title>color</title>
    <string/>
  </param>
  <return id="add_part.return">
    <int/>
  </return>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#add_part" xlink:role="add_part"/>
    <node xlink:href="#add_part.return" xlink:role="add_part.return"/>
    <node xlink:href="#add_part.name" xlink:role="add_part.name"/>
    <node xlink:href="#add_part.color" xlink:role="add_part.color"/>

    <node xlink:href="http://dcx.com/fix/stdlib.xml#strtoint.str"
     xlink:role="strtoint.str"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#strtoint.return"
     xlink:role="strtoint.return"/>

    <node xlink:href="werk2.xml#add_part" xlink:role="werk2.add_part"/>
    <node xlink:href="werk2.xml#add_part.return"
     xlink:role="werk2.add_part.return"/>
    <node xlink:href="werk2.xml#add_part.name"
     xlink:role="werk2.add_part.name"/>

    <node xlink:href="werk2.xml#del_part" xlink:role="werk2.del_part"/>
    <node xlink:href="werk2.xml#del_part.no"
     xlink:role="werk2.del_part.no"/>

    <node xlink:href="werk1.xml#add_part" xlink:role="werk1.add_part"/>
    <node xlink:href="werk1.xml#add_part.partno"
     xlink:role="werk1.add_part.partno"/>
    <node xlink:href="werk1.xml#add_part.name"
     xlink:role="werk1.add_part.name"/>
    <node xlink:href="werk1.xml#add_part.color"
     xlink:role="werk1.add_part.color"/>
```

```
    <node xlink:href="werk1.xml#del_part" xlink:role="werk1.del_part"/>
    <node xlink:href="werk1.xml#del_part.partno"
     xlink:role="werk1.del_part.partno"/>

    <node xlink:href="helper.xml#null" xlink:role="null1"
     instance="null1"/>
    <node xlink:href="helper.xml#null" xlink:role="null2"
     instance="null2"/>

    <!-- "normale" Abhängigkeiten -->

    <dep xlink:from="add_part.name" xlink:to="werk2.add_part.name"/>
    <dep xlink:from="add_part.name" xlink:to="werk1.add_part.partno"/>
    <dep xlink:from="add_part.color" xlink:to="werk1.add_part.color"/>
    <dep xlink:from="werk2.add_part.return" xlink:to="strtoint.str"/>
    <dep xlink:from="strtoint.return" xlink:to="add_part.return"/>
    <dep xlink:from="strtoint.return" xlink:to="werk1.add_part.partno"/>
    <dep xlink:from="werk1.add_part" xlink:to="add_part"/>

    <!-- Fehlerabhängigkeitsrelationen -->

    <dep xlink:from="werk2.add_part" xlink:to="werk2.del_part"
     xlink:role="compensation"/>
    <dep xlink:from="werk2.add_part" xlink:to="null2"
     xlink:role="compensation"/>
    <dep xlink:from="werk2.add_part.return"
     xlink:to="werk2.del_part.no"/>
    <dep xlink:from="werk2.del_part" xlink:to="null1"
     completion="required"/>

    <dep xlink:from="werk1.add_part" xlink:to="werk1.del_part"
     xlink:role="compensation"/>
    <dep xlink:from="werk1.add_part" xlink:to="null2"
     xlink:role="compensation"/>
    <dep xlink:from="strtoint.return"
     xlink:to="werk1.del_part.partno"/>
    <dep xlink:from="werk1.del_part" xlink:to="null"
     completion="required"/>

    <dep xlink:from="null1" xlink:to="add_part.return"/>
    <dep xlink:from="null2" xlink:to="add_part.return"
     completion="optional"/>
  </graph>
</op>

<op id="del_part" semantic="delete">
  <title>del_part</title>
  <param id="del_part.no">
    <title>no</title>
    <int/>
  </param>
  <raises xlink:href="#ENOTFOUND"/>
  <raises xlink:href="#EINTERNAL"/>
  <raises xlink:href="#ENOTDELETED"/>
</op>
```

```
<op id="get_color" semantic="read">
  <title>get_color</title>
  <param id="get_color.no">
    <title>no</title>
    <int/>
  </param>
  <return id="get_color.return">
    <string/>
  </return>
  <raises xlink:href="#ENOTFOUND"/>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#get_color.no" xlink:role="get_color.no"/>
    <node xlink:href="#get_color.return" xlink:role="get_color.return"/>
    <node xlink:href="werk1.xml#get_part.partno"
     xlink:role="get_part.partno"/>
    <node xlink:href="werk1.xml#get_part.color"
     xlink:role="get_part.color"/>
    <node xlink:href="helper.xml#col2str.color" xlink:role="col2str.color"/>
    <node xlink:href="helper.xml#col2str.return"
     xlink:role="col2str.return"/>

    <dep xlink:from="get_color.no" xlink:to="get_part.partno"/>
    <dep xlink:from="get_part.color" xlink:to="col2str.color"/>
    <dep xlink:from="col2str.return" xlink:to="get_color.return"/>
  </graph>
</op>

<op id="get_name" semantic="read">
  <title>get_name</title>
  <param id="get_name.no">
    <title>no</title>
    <int/>
  </param>
  <return id="get_name.return">
    <string/>
  </return>
  <raises xlink:href="#ENOTFOUND"/>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#get_name.no" xlink:role="get_name.no"/>
    <node xlink:href="#get_name.return" xlink:role="get_name.return"/>
    <node xlink:href="werk1.xml#get_part.partno"
     xlink:role="get_part.partno"/>
    <node xlink:href="werk1.xml#get_part.name"
     xlink:role="get_part.name"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.int"
     xlink:role="inttostr.int"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.return"
     xlink:role="inttostr.return"/>
    <node xlink:href="werk2.xml#get_part.no"
     xlink:role="werk2.get_part.no"/>
    <node xlink:href="werk2.xml#get_part.name"
     xlink:role="werk2.get_part.name"/>

    <dep xlink:from="get_name.no" xlink:to="get_part.partno"/>
    <dep xlink:from="get_name.no" xlink:to="inttostr.int"/>
```

```
    <dep xlink:from="get_part.name" xlink:to="get_name.return"/>
    <dep xlink:from="inttostr.return" xlink:to="werk2.get_part.no"/>
    <dep xlink:from="werk2.get_part.name" xlink:to="get_name.return"/>
  </graph>
</op>

<op id="get_size">
  <title>get_size</title>
  <param id="get_size.no">
    <title>no</title>
    <int/>
  </param>
  <param id="get_size.x" type="out">
    <title>x</title>
    <float type="float"/>
  </param>
  <param id="get_size.y" type="out">
    <title>y</title>
    <float type="float"/>
  </param>
  <param id="get_size.z" type="out">
    <title>z</title>
    <float type="float"/>
  </param>
  <raises xlink:href="#EINTERNAL"/>
  <raises xlink:href="#ENOTFOUND"/>
  <graph>
    <node xlink:href="#get_size.no" xlink:role="no"/>
    <node xlink:href="#get_size.x" xlink:role="x"/>
    <node xlink:href="#get_size.y" xlink:role="y"/>
    <node xlink:href="#get_size.z" xlink:role="z"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.int"
     xlink:role="inttostr.int"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.return"
     xlink:role="inttostr.return"/>
    <node xlink:href="werk2.xml#get_part.no" xlink:role="get_part.no"/>
    <node
     xlink:href='werk2.xml#fixpointer(id("get_part.dim")/member::dimension.x)'
     xlink:role="get_part.x"/>
    <node
     xlink:href='werk2.xml#fixpointer(id("get_part.dim")/member::dimension.y)'
     xlink:role="get_part.y"/>
    <node
     xlink:href='werk2.xml#fixpointer(id("get_part.dim")/member::dimension.z)'
     xlink:role="get_part.z"/>

    <dep xlink:from="no" xlink:to="inttostr.int"/>
    <dep xlink:from="inttostr.return" xlink:to="get_part.no"/>
    <dep xlink:from="get_part.x" xlink:to="x"/>
    <dep xlink:from="get_part.y" xlink:to="y"/>
    <dep xlink:from="get_part.z" xlink:to="z"/>
  </graph>
</op>

<op id="get_size_inch">
  <title>get_size_inch</title>
  <param id="get_size_inch.no">
```

```
    <title>no</title>
    <int/>
  </param>
  <param id="get_size_inch.x" type="out">
    <title>x</title>
    <float type="float"/>
  </param>
  <param id="get_size_inch.y" type="out">
    <title>y</title>
    <float type="float"/>
  </param>
  <param id="get_size_inch.z" type="out">
    <title>z</title>
    <float type="float"/>
  </param>
  <raises xlink:href="#EINTERNAL"/>
  <raises xlink:href="#ENOTFOUND"/>
  <graph>
    <node xlink:href="#get_size_inch.no" xlink:role="get_size_inch.no"/>
    <node xlink:href="#get_size_inch.x" xlink:role="get_size_inch.x"/>
    <node xlink:href="#get_size_inch.y" xlink:role="get_size_inch.y"/>
    <node xlink:href="#get_size_inch.z" xlink:role="get_size_inch.z"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.int"
     xlink:role="inttostr.int"/>
    <node xlink:href="http://dcx.com/fix/stdlib.xml#inttostr.return"
     xlink:role="inttostr.return"/>

    <node xlink:href="#m2inch.m" xlink:role="m2inch.mx" instance="x"/>
    <node xlink:href="#m2inch.inch" xlink:role="m2inch.inchx" instance="x"/>
    <node xlink:href="#m2inch.m" xlink:role="m2inch.my" instance="y"/>
    <node xlink:href="#m2inch.inch" xlink:role="m2inch.inchy" instance="y"/>
    <node xlink:href="#m2inch.m" xlink:role="m2inch.mz" instance="z"/>
    <node xlink:href="#m2inch.inch" xlink:role="m2inch.inchz" instance="z"/>
    <node xlink:href="werk2.xml#get_part.no" xlink:role="get_part.no"/>
    <node
     xlink:href="werk2.xml#fixpointer(id('get_part.dim')/member::dimension.x"
     xlink:role="get_part.dim.x"/>
    <node
     xlink:href="werk2.xml#fixpointer(id('get_part.dim')/member::dimension.y"
     xlink:role="get_part.dim.y"/>
    <node
     xlink:href="werk2.xml#fixpointer(id('get_part.dim')/member::dimension.z"
     xlink:role="get_part.dim.z"/>

    <dep xlink:from="get_size_inch.no" xlink:to="inttostr.int"/>
    <dep xlink:from="inttostr.return" xlink:to="get_part.no"/>
    <dep xlink:from="get_part.dim.x" xlink:to="m2inch.mx"/>
    <dep xlink:from="get_part.dim.y" xlink:to="m2inch.my"/>
    <dep xlink:from="get_part.dim.z" xlink:to="m2inch.mz"/>
    <dep xlink:from="m2inch.inchx" xlink:to="get_size_inch.x"/>
    <dep xlink:from="m2inch.inchy" xlink:to="get_size_inch.y"/>
    <dep xlink:from="m2inch.inchz" xlink:to="get_size_inch.z"/>
  </graph>
</op>

<op id="get_price_euro">
  <title>get_price_euro</title>
```

```
  <param id="get_price_euro.no">
    <title>no</title>
    <int/>
  </param>
  <return id="get_price_euro.return">
    <type xlink:href="#price_t"/>
  </return>
  <raises xlink:href="#ENOTFOUND"/>
  <raises xlink:href="#EINTERNAL"/>
  <graph>
    <node xlink:href="#get_price_euro.no" xlink:role="get_price_euro.no"/>
    <node xlink:href="#get_price_euro.return"
     xlink:role="get_price_euro.return"/>
    <node xlink:href="lieferant.xml#currency_e.Euro" xlink:role="Euro"/>
    <node xlink:href="lieferant.xml#currency" xlink:role="currency"/>
    <node xlink:href="lieferant.xml#get_price" xlink:role="get_price"/>
    <node xlink:href="lieferant.xml#get_price.no"
     xlink:role="get_price.no"/>
    <node xlink:href="lieferant.xml#get_price.return"
     xlink:role="get_price.return"/>

    <dep xlink:from="Euro" xlink:to="currency"/>
    <dep xlink:from="currency" xlink:to="get_price"/>
    <dep xlink:from="get_price_euro.no" xlink:to="get_price.no"/>
    <dep xlink:from="get_price.return" xlink:to="get_price_euro.return"/>
  </graph>
</op>

<op id="set_price_euro">
  <title>set_price_euro</title>
  <param id="set_price_euro.no">
    <title>no</title>
    <int/>
  </param>
  <param id="set_price_euro.price">
    <title>price</title>
    <type xlink:href="#price"/>
  </param>
  <graph>
    <node xlink:href="#set_price_euro" xlink:role="set_price_euro"/>
    <node xlink:href="#set_price_euro.no" xlink:role="set_price_euro.no"/>
    <node xlink:href="#set_price_euro.price"
     xlink:role="set_price_euro.price"/>
    <node xlink:href="lieferant.xml#currency_e.Euro" xlink:role="Euro"/>
    <node xlink:href="lieferant.xml#currency" xlink:role="currency"/>
    <node xlink:href="lieferant.xml#get_price" xlink:role="get_price"/>
    <node xlink:href="lieferant.xml#get_price.no"
     xlink:role="get_price.no"/>
    <node xlink:href="lieferant.xml#get_price.return"
     xlink:role="get_price.return"/>
    <node xlink:href="stdlib.xml#isequal" xlink:role="isequal"/>
    <node xlink:href="stdlib.xml#isequal.a" xlink:role="isequal.a"/>
    <node xlink:href="stdlib.xml#isequal.b" xlink:role="isequal.b"/>
    <node xlink:href="stdlib.xml#isnotequal" xlink:role="isnotequal"/>
    <node xlink:href="stdlib.xml#isnotequal.a" xlink:role="isnotequal.a"/>
    <node xlink:href="stdlib.xml#isnotequal.b" xlink:role="isnotequal.b"/>
    <node xlink:href="lieferant.xml#set_price" xlink:role="set_price"/>
```

```
        <node xlink:href="lieferant.xml#set_price.no"
         xlink:role="set_price.no"/>
        <node xlink:href="lieferant.xml#set_price.price"
        xlink:role="set_price.price"/>

         <dep xlink:from="Euro" xlink:to="currency"/>
         <dep xlink:from="currency" xlink:to="get_price"/>
         <dep xlink:from="set_price_euro.no" xlink:to="get_price.no"/>
         <dep xlink:from="get_price.return" xlink:to="isequal.b"/>
         <dep xlink:from="set_price_euro.price" xlink:to="isequal.a"/>
         <dep xlink:from="isequal" xlink:to="set_price_euro"/>
         <dep xlink:from="get_price.return" xlink:to="isnotequal.b"/>
         <dep xlink:from="set_price_euro.price" xlink:to="isnotequal.a"/>
         <dep xlink:from="isnotequal" xlink:to="set_price"/>
         <dep xlink:from="set_price_euro.no" xlink:to="set_price.no"/>
         <dep xlink:from="set_price_euro.price" xlink:to="set_price.price"/>
         <dep xlink:from="set_price" xlink:to="set_price_euro"/>
      </graph>
    </op>

    <op id="init">
      <title>init</title>
      <raises xlink:href="#EINTERNAL"/>
      <graph>
        <node xlink:href="#init" xlink:role="global.init"/>
        <node xlink:href="werk1.xml#neustart" xlink:role="werk1.neustart"/>
        <node xlink:href="werk2.xml#reboot" xlink:role="werk2.reboot"/>

        <dep xlink:from="werk1.neustart" xlink:to="global.init"
         completion="required"/>
        <dep xlink:from="werk2.reboot" xlink:to="global.init"
         completion="required"/>
      </graph>
    </op>

  </interface>

</system>
```

E.6 Workflow

```
STRUCTURE 'get_color_in'
  'no': LONG;
END 'get_color_in'

STRUCTURE 'get_color_out'
  'return': STRING;
END 'get_color_out'

STRUCTURE 'get_part_in'
  'partno': LONG;
END 'get_part_in'

STRUCTURE 'get_part_out'
  'name': STRING;
```

```
  'color': LONG;
END 'get_part_out'

STRUCTURE 'col2str_in'
  'color': LONG;
END 'col2str_in'

STRUCTURE 'col2str_out'
  'return': STRING;
END 'col2str_out'

PROGRAM 'corba'
  STRUCTURES_FROM_ACTIVITY
  WINNT EXE PATH_AND_FILENAME
    "C:\Programme\FIX\Wrapper\CORBA.EXE"
      STYLE INVISIBLE
END 'corba'

PROGRAM 'oncrpc'
  STRUCTURES_FROM_ACTIVITY
  WINNT EXE PATH_AND_FILENAME
    "C:\Programme\FIX\Wrapper\ONCRPC.EXE"
      STYLE INVISIBLE
END 'oncrpc'

PROCESS 'get_color' ('get_color_in', 'get_color_out')
  SOURCE 1 XPOS -700 YPOS 600
  SINK 1 XPOS -700 YPOS -200

  PROGRAM_ACTIVITY 'get_part' ( 'get_part_in', 'get_part_out' )
    INPUT_CONTAINER
      'address' INITIAL_VALUE 'werk1.example.com'
      'program' INITIAL_VALUE '600000000'
      'version' INITIAL_VALUE '1'
      'signature' INITIAL_VALUE
        'void get_part(in partno_t partno, out string name, out color_t color)'
    PROGRAM 'oncrpc'
  END 'get_part'

  PROGRAM_ACTIVITY 'col2str' ( 'col2str_in', 'col2str_out' )
    INPUT_CONTAINER
      'address' INITIAL_VALUE 'server.example.com'
      'port' INITIAL_VALUE '683'
      'signature' INITIAL_VALUE 'string col2str(in color_t color)'
    PROGRAM 'corba'
  END 'col2str'

CONTROL
  FROM 'get_part' TO 'col2str'

DATA
  FROM SOURCE 1 TO 'get_part'
    MAP 'no' TO 'partno'

DATA
  FROM 'get_part' TO 'col2str'
    MAP 'color' TO 'color'
```

```
DATA
  FROM 'col2str'
    MAP 'return' TO 'return'

END 'get_color'
```

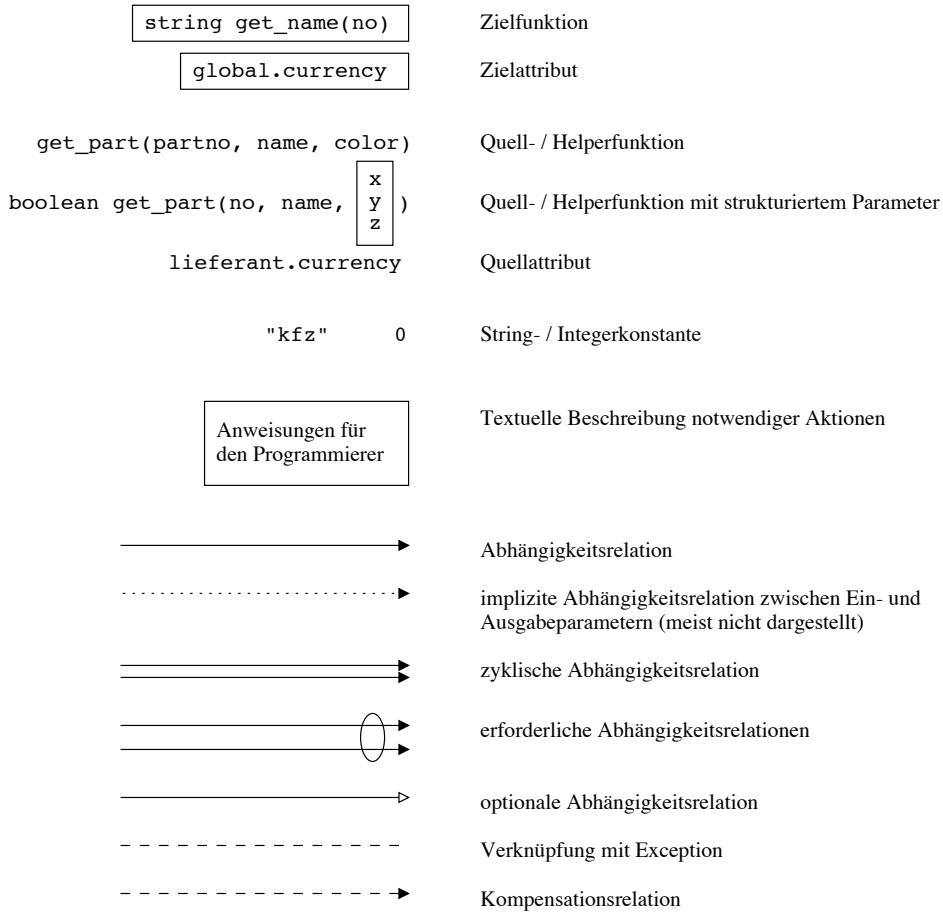
Anhang F

Mögliche Abhängigkeiten

Attribut	→	Attribut	Abschnitt 3.2.1 und 3.2.2
	→	Funktion	
	→	Konstante	
	→	Parameter	
	→	Text	
Funktion	→	Funktion	Abschnitt 3.1.4
	→	Konstante	Abschnitt 3.1.5
	→	Text	Abschnitt 3.1.8
Konstante	→	Attribut	Abschnitt 3.2.1 und 3.2.2
	→	Parameter	Abschnitt 3.1.5
	→	Text	Abschnitt 3.1.8
Konstruktor	→	Parameter	Abschnitt 3.2.3
Parameter	→	Attribut	Abschnitt 3.2.1 und 3.2.2
	→	Parameter	Abschnitt 3.1.1
	→	Text	Abschnitt 3.1.8
Text	→	Attribut	Abschnitt 3.2.1 und 3.2.2
	→	Funktion	Abschnitt 3.1.8
	→	Konstante	
	→	Parameter	
Funktion 1	⇒	Funktion 2	Abschnitt 3.3.2
Listenparameter	⇒	Parameter	
Parameter	⇒	Listenparameter	

Anhang G

Symbole



Literaturverzeichnis

[C] BRIAN W. KERNIGHAN, DENNIS M. RITCHIE: *Programmieren in C*, Prentice-Hall International, 2. Auflage, 1990.

[C++] BJARNE STROUSTRUP: *Die C++-Programmiersprache*, Addison-Wesley, 3. Auflage, 1998.

[CORBA] OPEN MANAGEMENT GROUP: *The Common Object Request Broker: Architecture and Specification*, Revision 2.3.1, 1999; zu beziehen über http://www.omg.org/library/c2indx.html.

[Dadam, 1996] PETER DADAM: *Verteilte Datenbanken und Client/Server-Systeme*, Springer Verlag, 1996.

[DB2] IBM CORP.: *DB2 Product Family*; zu beziehen über http://www.software.ibm.com/data/db2/.

[DCE] THE OPEN GROUP: *DCE 1.2.2 Documentation - Full Set*, 1997.

[DOM] VIDUR APPARAO ET AL.: *Document Object Model (DOM) Level 1 Specification*, W3C Recommendation, 1998; zu beziehen über http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.

[Gray, 1993] JIM GRAY, ANDREAS REUTER: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1993.

[Härder, 1999] THEO HÄRDER, GÜNTER SAUTER, JOACHIM THOMAS: The Intrinsic Problems of Structural Heterogeneity and an Approach to Their Solution, *VLDB Journal 8(1)*, S. 25 – 43, 1999; zu beziehen über http://wwwdbis.informatik.uni-kl.de:8080/publications/HST99.VLDB.html.

[Hergula, 1999] KLAUDIA HERGULA, THEO HÄRDER: Eine Abbildungsbeschreibung zur Funktionsintegration in heterogenen Anwendungssystemen, in: Ralf-Detlef Kutsche, Ulf Leser, Johann Christoph Freytag: *FDBS 1999, 4. Workshop Föderierte Datenbanken, 25. – 26. November*, Berlin, 1999; zu beziehen über http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-25/paper

[HTML] DAVE RAGGETT, ARNAUD LE HORS, IAN JACOBS: *HTML 4.01 Specification*, W3C Recommendation, 1999; zu beziehen über http://www.w3.org/TR/1999/REC-html401-19991224.

[INTERFACE1] WORKFLOW MANAGEMENT COALITION: *Interface 1: Process Definition Interchange, Process Modell*, 1999; zu beziehen über http://www.aiim.org/wfmc/standards/docs/if199110v11.pdf.

[Leymann, 2000] FRANK LEYMANN, DIETER ROLLER: *Production Workflow: Concepts and Techniques*, Prentice-Hall, 2000.

[MATHML] PATRICK ION, ROBERT MINER: *Mathematical Markup Language (MathML[tm]) 1.01 Specification*, W3C Recommendation, 1999; zu beziehen über http://www.w3.org/1999/07/REC-MathML-19990707.

[Meyberg, 1993] KURT MEYBERG, PETER VACHENAUER: *Höhere Mathematik 1*, Springer-Verlag, 2. Auflage, 1993.

[MIME] N. FREED, N. BORENSTEIN: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, IETF RFC 2046, 1996; zu beziehen über ftp://ftp.ietf.org/rfc/rfc2046.txt

[MQSERIES] IBM CORP.: *MQSeries Family*; zu beziehen über http://www.software.ibm.com/ts/mqseries/.

[MQWORKFLOW1] IBM CORP.: *MQSeries Workflow*; zu beziehen über http://www.software.ibm.com/ts/mqseries/workflow/.

[MQWORKFLOW2] IBM CORP.: *IBM MQSeries Workflow — Getting Started with Buildtime*, 5. Auflage, 1999.

[MQWORKFLOW3] IBM CORP.: *IBM MQSeries Workflow — Programming Guide*, 5. Auflage, 1999.

[Murata, 2000] M. MURATA, S. ST. LAURENT, D. KOHN: *XML Media Types*, Internet Draft, 15.05.2000; zu beziehen über ftp://ftp.ietf.org/internet-drafts/draft-murata-xml-04.txt.

[NAMESPACE] TIM BRAY, DAVE HOLLANDER, ANDREW LAYMAN: *Namespaces in XML*, W3C Recommendation, 1999; zu beziehen über http://www.w3.org/TR/REC-xml-names-19990114.

[ONCRPC] R. SRINIVASAN: *RPC: Remote Procedure Call Protocol Specification Version 2*, IETF RFC 1831, 1995; zu beziehen über ftp://ftp.ietf.org/rfc/rfc1831.txt.

[OPENMATH] THE OPENMATH ESPRIT CONSORTIUM: *The OpenMath Standard*, Version 0.3, 1999; zu beziehen über http://www.nag.co.uk/projects/OpenMath/omstd/.

[RMI] SUN MICROSYSTEMS: *Java^{TM} Remote Method Invocation Specification*, Revision 1.50, JDK 1.2, 1998; zu beziehen über ftp://ftp.javasoft.com/docs/jdk1.2/rmi-spec-JDK1.2.pdf.

[SAP] SAP AG: *System R/3*; zu beziehen über http://www.sap-ag.de/germany/products/r3/.

[Sauter, 1998] GÜNTER SAUTER: *Interoperabilität von Datenbanksystemen bei struktureller Heterogenität*, Dissertation, infix-Verlag, 1998.

[SCHEMA1] HENRY S. THOMPSON, DAVID BEECH, MURRAY MALONEY, NOAH MENDELSOHN: *XML Schema Part 1: Structures*, W3C Working Draft, 17.12.1999, zu beziehen über http://www.w3.org/TR/1999/WD-xmlschema-1-19991217/.

[SCHEMA2] PAUL V. BIRON, ASHOK MALHOTRA: *XML Schema Part 2: Datatypes*, W3C Working Draft, 17.12.1999; zu beziehen über http://www.w3.org/TR/1999/WD-xmlschema-2-19991217/.

[Schöning, 1995] UWE SCHÖNING: *Theoretische Informatik I*, Vorlesungsskript, 1995.

[Vogt, 1999] MARKUS VOGT: *Entwurf von Mechanismen zur Integration heterogener Anwendungssysteme*, Diplomarbeit, Universität Ulm, 1999.

[XA] THE OPEN GROUP: *Distributed TP: The XA Specification*, 1992.

[XDR] R. SRINIVASAN: *XDR: External Data Representation Standard*, IETF RFC 1832, 1995; zu beziehen über ftp://ftp.ietf.org/rfc/rfc1832.txt.

[XLINK] STEVE DEROSE, EVE MALER, DAVID ORCHARD, BEN TRAFFORD: *XML Linking Language (XLink)*, W3C Working Draft, 21.02.2000; zu beziehen über http://www.w3.org/TR/2000/WD-xlink-20000221.

[XML] TIM BRAY, JEAN PAOLI, C.M. SPERBERG-MCQUEEN: *Extensible Markup Language (XML) 1.0*, W3C Recommendation, 1998; zu beziehen über http://www.w3.org/TR/1998/REC-xml-19980210.

[XPATH] JAMES CLARK, STEVE DEROSE: *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 1999; zu beziehen über http://www.w3.org/TR/1999/REC-xpath-19991116.

[XPOINTER] STEVE DEROSE, RON DANIEL JR., EVE MALER: *XML Pointer Language (XPointer)*, W3C Working Draft, 6.12.1999; zu beziehen über http://www.w3.org/TR/1999/WD-xptr-19991206.

[XSL] SHARON ADLER ET AL.: *Extensible Stylesheet Language (XSL) Version 1.0*, W3C Working Draft, 27.03.2000; zu beziehen über http://www.w3.org/TR/2000/WD-xsl-20000327/.

[XSLT] JAMES CLARK: *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, 1999; zu beziehen über http://www.w3.org/TR/1999/REC-xslt-19991116.

Index