INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Fortgeschrittenen-Praktikum

Implementierung eines CORBA TopologyService in Java

Bearbeiter:	Harald Rölle
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Boris Gruschke
	Stephen Heilbronner

Inhalt

Abbildungs und Tabellenverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Die im Rahmen dieses Fortgeschrittenen-Praktikums zu bearbeitende Aufgabe bestand in der Realisierung einer Implementierung des CORBA TopologyService, wie er in [1] spezifiziert ist. Als Laufzeitumgebung war das an der Forschungs- und Lehreinheit entwickelte MASA Agentensystem vorgegeben ([7]).

1.2 Überblick über den CORBA TopologyService

Zur Einführung sollen zunächst die wichtigsten Eigenschaften des CORBA TopologyService dargestellt werden. Für eine ausführliche Darstellung sei auf [1], Kapitel 2 verwiesen.

Zweck des TopologyService ist es, eine standardisierte Umgebung zur persistenten Speicherung und Abfrage weitgehend frei definierbarer topologischer Informationen zu schaffen. Dies kann beispielsweise bedeuten, daß die Hardware-Topologie eines Rechnernetzes (Netzkarten, Hubs, Bridges, etc.) hinterlegt werden kann oder aber auch die Beziehungen zwischen Benutzern, Rechnern und Anwendungsprogrammen.

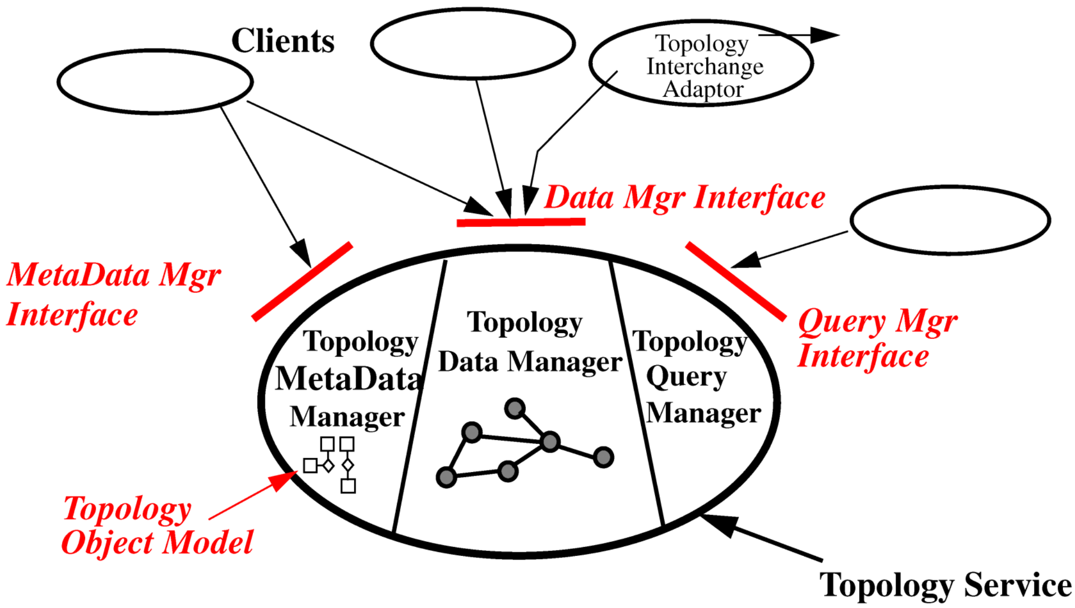

Abbildung 1.1: Überblick über die Architektur des TopologyService (Quelle: [1], Seite 8)

Wie in Abbildung 1.1 ersichtlich, besteht der CORBA TopologyService aus drei sogenannten Managern. Im *TopologyMetaDataManager* wird das Modell der zu verwaltenden Topologie(n) abgebildet. Der *TopologyMetaDataManager* verwaltet dann die konkrete Ausprägung der topologischen Informationen, welche dann mittels des *TopologyQueryManager* abgefragt werden können.

1.2.1 Typ-Verwaltung und Objekte einer Topologie

Zur Verwaltung topologischer Informationen wird zunächst eine Typ-Verwaltung durch den *TopologyMetaDataManager* im TopologyService bereitgestellt. Hiermit lassen sich frei definierbare Typen in einer hierarchischen Eltern/Kind-Beziehung anordnen (siehe Abbildung 1.2).

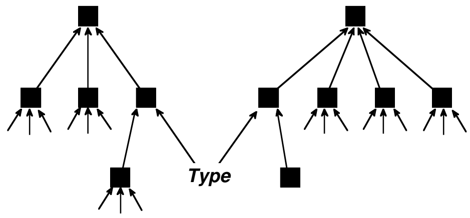

Abbildung 1.2: Schematische Darstellung der Typ-Hierarchie

Ein Beispiel für eine solche Typ-Hierarchie könnte lauten:

„Netzwerkkarte“ ← „EthernetKarte“ ← „TwistedPairKarte“

Daneben verwaltet der TopologyService im *TopologyDataManager* die in einer Topologie enthaltenen Objekte, hier genannt *Entities*. Jeder *Entity* wird ein eindeutiger Identifikator (*PartitionedIdentifier*) und ein Typ aus der Typ-Hierarchie (*TopologicalType*) zugeordnet, außerdem kann optional eine Referenz auf ein beliebiges CORBA-Objekt hinterlegt werden. Diese Referenz stellt die Schnittstelle zu anderen CORBA-Anwendungen auf Ebene der *Entities* dar.

1.2.2 Relationen

Für die Verknüpfung von *Entities* sind im *TopologyDataManager* zwei Relationsklassen definiert.

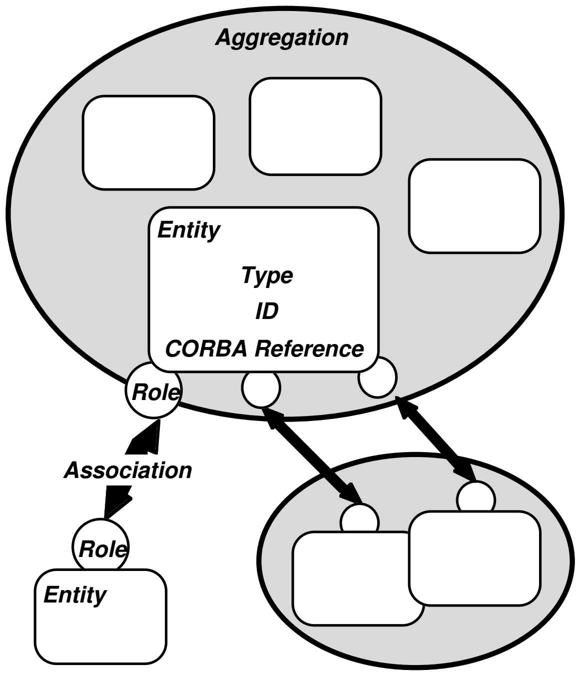

Abbildung 1.3: Schematischer Zusammenhang von Objekten und Relationen

Zweistellige Relationen, *Associations* (Abbildung 1.3, links unten) genannt, werden benutzt um genau zwei *Entities* miteinander zu verbinden. Den Teilnehmern kann jeweils eine *Role* (Rolle) zugeordnet werden. *Roles* sind freie Zeichenketten, die nicht weiter im TopologyService verwaltet werden. Die

Bildung von *Associations* kann mittels zur Laufzeit definierbarer *Rules* (Regeln) statisch eingeschränkt werden. Zur fallweisen Überprüfung, ob eine *Association* eingegangen werden darf, sind sogenannte *Enforcerer*, spezielle CORBA-Objekte, definiert. Ein Beispiel für eine *Association* wäre die Relation zwischen einem Netzwerkinterface (mit der Rolle „Client“) und einem Hub (mit der Rolle „Port“).

Aggregation (Abbildung 1.3, oben), der zweite Relationstyp, der durch den TopologyService definiert ist, dient der Gruppenbildung. Hiermit können mehrere *Entities* in einer Menge zusammengefaßt werden, diese wird dann als *AggregateEntity* (*AE*) bezeichnet. In den diversen Methoden des TopologyService können dann alle *Entities* einer *Aggregation* gemeinsam angesprochen werden. Einer *Aggregation* wird dabei kein eigener Identifikator zugewiesen, vielmehr repräsentiert jede *Entity* einer *Aggregation* die gesamte Gruppe. Die Spezifikation macht allerdings eine starke Einschränkung darüber, welche *Entities* zu einer *Aggregation* zusammengefaßt werden können: Die transitiven Hüllen der *TopologicalTypes* aller an einer *Aggregation* teilnehmenden *Entities* müssen disjunkt sein.

1.2.3 Weitere Merkmale

Die genannten Daten werden im TopologyService persistent gespeichert und können über den *TopologyQueryManager* mittels einer Abfragesprache ausgelesen werden. Hierfür ist die eigene Abfragesprache *Topology Query Language* (*TQL*) definiert.

Ergänzend sei erwähnt, daß die Spezifikation keine Aussagen darüber macht, wie konkrete Topologien mit den zu Verfügung gestellten Mitteln zu modellieren sind.

1.3 Überblick über „Mobile Agent System Architecture“ (MASA)

Zum Abschluß der Einführung soll nun noch die in [7] entwickelte „Mobile System Agent Architecture“ (MASA) kurz vorgestellt werden. MASA stellt eine plattformunabhängige Laufzeit- und Kommunikationsarchitektur für mobile Agenten dar. Abbildung 1.1 liefert eine Übersicht der MASA-Architektur.

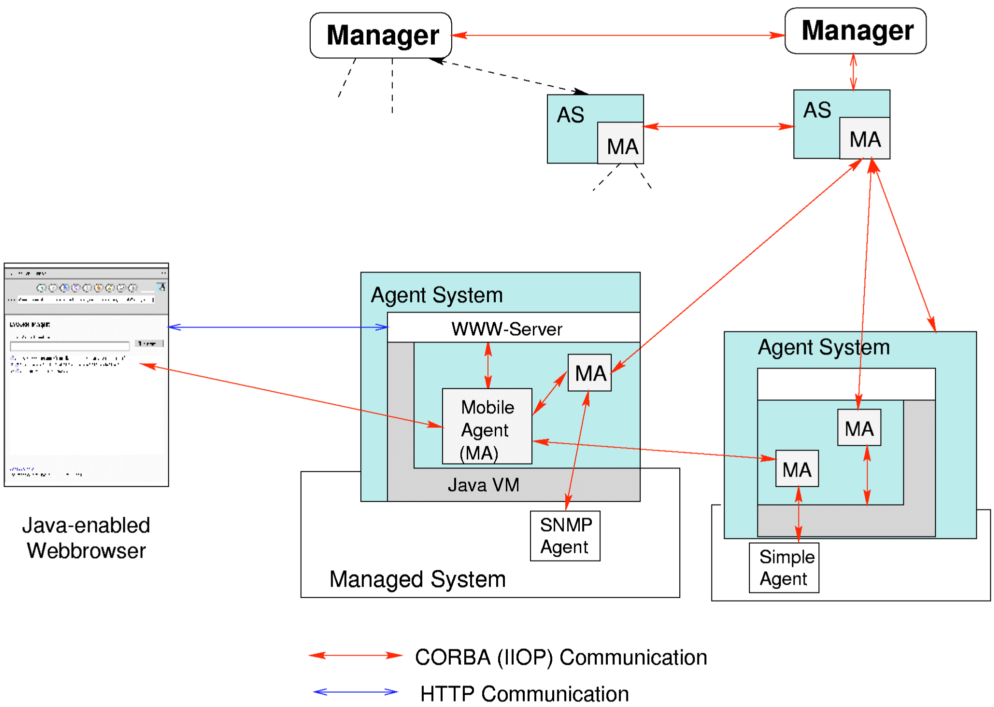

Abbildung 1.4: Übersicht über das MASA-System (Quelle: Homepage „Flexible Agent“)

1.3.1 Komponenten

Auf einem zu managenden Endsystem (*managed system*) wird ein Agentensystem (*agent system*) ausgeführt, das die Laufzeitumgebung für alle Agenten (*mobile agents*) auf einem Endsystem darstellt. Agentensystem und Agenten sind in der Programmiersprache Java implementiert.

Basis des Agentensystems und Abstraktionsschicht von plattformabhängigen Teilen des Endsystems ist die Java Virtual Machine, die Java-eigene Laufzeitumgebung. In ihr eingebettet werden die Agenten ausgeführt. Weiterhin ist ein eigener Webserver Teil eines jeden Agentensystems. Dieser wird sowohl in der Kommunikation zur Steuerung der Agenten, als auch des Agentensystems selbst eingesetzt.

1.3.2 Kommunikation

Agenten können mit anderen Agenten kommunizieren und selbständig zu anderen Agentensystemen migrieren. Dabei erfolgt die gesamte Kommunikation über CORBA ([9]). Zur Steuerung eines Agenten muß dieser ein eigenes, wiederum in Java implementiertes, Applet mitbringen. Will ein Benutzer auf einen Agenten zugreifen, fordert er vom Webserver des Agentensystems über das HTTP-Protokoll das Applet des Agenten an. Dieses wird dann auf seinen Webbrowser übertragen und dort ausgeführt. Im folgenden kommuniziert dann das Applet via CORBA direkt mit dem zu steuernden Agenten.

2 Entwicklungsumgebung

Die vorliegende Implementierung des TopologyService basiert auf der Programmiersprache Java Version 1.1 ([11] und [12], Anhang D). Die verwendete Entwicklungsumgebung hierfür war Sun JDK Version 1.1.6 ([13]), unter den Betriebssystemen Linux und Solaris.

Obwohl Java sich als eine plattformunabhängige Sprache versteht, führte die Verwendung von unterschiedlichen Betriebssystemen (Solaris bzw. Linux) zu diversen Problemen. Hauptursache hierfür war, daß die am Lehrstuhl bevorzugte CORBA-Laufzeitumgebung „Visibroker“ ([5]) nicht für Linux verfügbar ist, und deshalb eine andere CORBA-Implementierung gewählt werden mußte. Dabei zeigte sich, daß die beiden Laufzeitumgebungen keineswegs reibungslos gegeneinander austauschbar sind, wie dies der CORBA-Standard 2.0 ([9]) eigentlich impliziert. Als in der Praxis besonders problematisch erwies sich die Tatsache, daß in beiden ORBs funktional gleiche Methoden unterschiedliche Signaturen aufweisen.

Um trotzdem alternierend unter Solaris und Linux sinnvoll entwickeln zu können, mußten einige Voraussetzungen geschaffen werden, die in diesem Kapitel erläutert werden.

2.1 ORBs: ORBacus und Visibroker

Für die Linux-Plattform wählte der Autor das „ORBacus“ Paket der Firma „Object Oriented Concepts“, das in der Version 3.0 vorliegt ([4]). ORBacus 3.0 ist eine CORBA-Entwicklungs- und Laufzeitumgebung für die Programmiersprachen C++ und Java. Zu den unterstützen Betriebssystemen gehören nach Herstellerangaben diverse Unix Systeme (wie Linux, Solaris, HP/UX), sowie Microsoft Windows NT. Mit ORBacus wurde vom Autor ausschließlich unter Linux entwickelt. Das System liegt im Quelltext vor und die Benutzung des Systems ist für nicht-kommerzielle Anwendungen frei, konnte also für die vorliegende Implementierung legal verwendet werden.

Zu ORBacus 3.0 gehört auch ein „IDL nach Java“-Übersetzer (`jidl`), der für die unterschiedlichen Betriebssysteme als fertige, ausführbare Applikation geliefert wird. Dieser Übersetzer ist für die Umsetzung von IDL-Quelltexten, der Schnittstellen-Beschreibungssprache von CORBA, nach Java verantwortlich. Dabei werden sowohl die Klassen für die Implementierung eines CORBA-Servers als auch jene für CORBA-Clients erzeugt.

Leider zeigte `jidl`, zumindest die Linux Variante, eine gravierende Schwäche. Es fehlt die Möglichkeit, `jidl` anzuweisen, daß Klassen nicht nur die durch den IDL-Quelltext vorgegebenen Schnittstellen und Klassen implementieren bzw. erben, sondern daß zusätzlich die Java Schnittstelle `java.lang.Serializable` implementiert wird. Um den in Java vorgesehenen Serialisierungsmechanismus nutzen zu können, ist dies aber notwendig (näheres siehe [2] und [3]). Behoben wird dieses Manko mittels eines Shellskripts[1], das über ein `awk`-Skript die notwendigen Ergänzungen vornimmt. Das Ergebnis entspricht einem Aufruf von Visibrokers „`idl2java`“ mit der `-all_serializable` Option.

[1] Quelltext sie Anhang B

Unterschiede im API von ORBacus und Visibroker zeigten sich an den folgenden Stellen

	ORBacus 3.0	Visibroker 3.0
Namen der Tie-Klassen	`_<module_name>ImplBase_tie`	`_tie_<module_name>`
Anmeldung des Server, der mittels Tie-Mechanismus implementiert wurde	`orb.connect( EM_tie);`	`boa.obj_is_ready( EM_tie);`
Blockierendes Warten auf Anfragen	`boa.impl_is_ready( null);`	`boa.impl_is_ready();`
Abmelden des Servers	`boa.deactivate_impl( null);`	`boa.shutdown();`
Bezug von Objekt-Referenzen über bind Mechanismus	Nicht vorhanden, da nicht Standard	Proprietäre Erweiterung

Tabelle 2.1: Unterschiede der Signaturen in ORBacus/Visibroker

2.2 Java Compiler und Laufzeitumgebung

2.2.1 Präprozessor

Für die flexible Verwendung des Quelltexts für beide ORBs ist die konditionale Übersetzung durch Steuerung über globale Konstanten (wie in Java üblich, siehe [2], Seite 22) nicht möglich. Der Grund hierfür liegt darin, daß bei dieser Methode die syntaktische Korrektheit aller Varianten überprüft werden muß, was durch die unterschiedlichen Signaturen zentraler Methoden und Klassennamen nicht möglich ist.

Die Verwendung eines Präprozessors, wie er aus der Programmiersprache C bekannt ist, stellt hier einen Ausweg dar, da dieser die jeweiligen, nicht zur Übersetzung anstehenden Teile ausfiltert, bevor der Quelltext an den Übersetzer übergeben wird. Leider ist ein Präprozessor in Java nicht vorgesehen.

Um dennoch einen Präprozessor nutzen zu können, mußte eine Eigenschaft des verwendeten Java Entwicklungssystems beachtet werden. Im Gegensatz zu C wird in Java eine automatische Auflösung von syntaktischen Abhängigkeiten auf Java-Ebene zwischen unterschiedlichen Quelldateien durch den Übersetzer vorgenommen und diese bei Bedarf auch sofort übersetzt. Somit ist eine, unter Unix gängige, „pipe“-Verkettung von Präprozessor und Java Compiler ausgeschlossen. Vielmehr müssen alle Quelldateien vor dem ersten Aufruf des Java Compilers den Präprozessor durchlaufen haben, da im zuge des Übersetzungsvorgangs nicht nur die angegebene Datei betrachtet wird, sondern auch all jene, von denen diese Datei abhängig ist. Dies wiederum führt zu dem Problem, daß nun die Ergebnisse der Präprozessor-Läufe explizit in Form von Dateien vorliegen müssen, die, durch den Suchalgorithmus von Java bedingt, mit identischen Dateinamen in korrekter Hierarchie zu den Quelldateien vorliegen müssen.

Wegen dieser Randbedingungen entstand eine Konstruktion, die zwischen den universellen Quelldateien und den Präprozessor-Ergebnissen unterscheidet. Die Präprozessor-Ergebnisse werden in einem sogenannten „Produktionsverzeichnis“ abgelegt, in dem dann auch die eigentliche Übersetzung durch den Java Compiler durchgeführt wird und sich schließlich auch die übersetzten Programme befinden.

2.2.2 Makefiles

Realisiert wurde oben beschriebene Verfahren durch eine Reihe von Makefiles, die den gesamten Entwicklungsvorgang unterstützen:

Im Wurzelverzeichnis des Installationspakets befindet sich das „Master Makefile“, das alle Aktionen initiiert und koordiniert. Dort werden auch alle Parameter und plattformspezifischen Einstellungen aus `Makefile.DEF` importiert. Das Master Makefile unterstützt die folgenden Aktionen:

- Erstellen eines Produktionsverzeichnises
- Generierung von Java Quellen aus IDL-Definitionen
- Übersetzung der diversen Quellen
- Installation im MASA Agentensystem
- Ausführung der übersetzten Programme
- Erstellen einer Javadoc Dokumentation

Die konkreten Namen der `make`-Regeln sind aus der `INSTALL`-Datei ersichtlich.

In den Quellverzeichnissen befinden sich Makefiles, die im wesentlichen die Ausführung des Präprozessors steuern und die Ergebnisse im Produktionsverzeichnis ablegen.

Im Produktionsverzeichnis selbst befindet sich das Makefile, das schließlich die diversen Übersetzer beziehungsweise Tools aufruft. Abbildung 2.1 zeigt schematisch den Aufbau der Verzeichnisse.

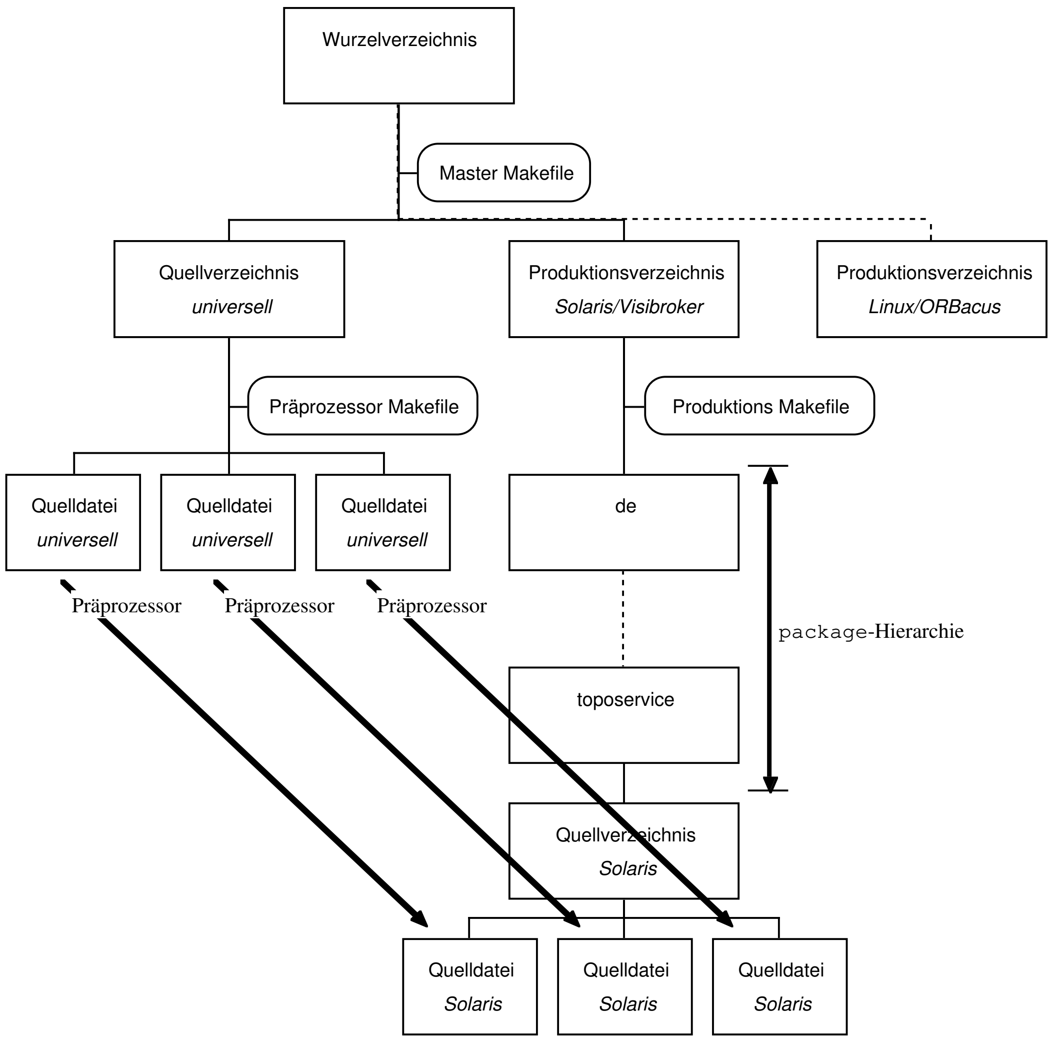

Abbildung 2.1: Schematischer Verzeichnisaufbau

Durch die Verwendung eines Präprozessors wurde außerdem eine weitere Automatisierungsstufe im Entwicklungsprozeß möglich. Gewöhnlich muß sich die package-Hierarchie der Java-Klassen in der die Klassen implementierenden Datei-Hierarchie wiederspiegeln, damit die automatische Auflösung von Abhängigkeiten durch den Java Compiler funktioniert. Durch den Präprozessor sind nun auch Textersetzungen möglich, was es erlaubt, die package-Hierarchie zentral als Parameter festzulegen. Die Quelldateien selbst müssen deshalb weder die konkrete Spezifikation der package-Hierarchie enthalten, noch ist eine Anordnung der Quelldateien entsprechend package-Hierarchie notwendig.

In der Anwendung bedeutet dies, daß in den Quelltexten beispielsweise anstatt der konstanten `package`-Anweisung

```
package de.unimuenchen.informatik.mnm.masa.agent.toposervice.TopologyData;
```

die Zeile

```
package __SERVER_PACKAGE__(TopologyData);
```

steht. In `Makefile.DEF` wird dann mittels der Zuweisung

```
export SERVER_PACKAGE = de.unimuenchen.informatik.mnm.masa.agent.toposervice
```

das entsprechende package konkret festgelegt.

Die korrekte Anordnung der Quelldateien geschieht erst im Produktionsverzeichnis und wird automatisch durch die Makefiles vorgenommen. Damit ist es möglich, auf einfache Weise, das gesamte Paket in der Hierarchie der Java-Klassen neu anzuordnen.

Details zu den in diesem Kapitel beschriebenen Mechanismen und Parameter entnehme man der `README` und `INSTALL`-Datei im Wurzelverzeichnis des Installationspakets.

2.3 GUI-Bibliothek „Swing“

Obwohl der CORBA TopologyService selbst keine graphische Benutzerschnittstelle enthält, mußte für die Implementierung als MASA-Agent eine solche entwickelt werden (siehe Kapitel 1.3). Für alle GUI-Elemente wurde hierzu die „Swing“ Bibliothek von „Sun Microsystems“ in der aktuellen Version 1.1beta2 ([8]) benutzt.

Obwohl sich dieses Oberflächenbibliothek noch in der Entwicklungsphase befindet wählte der Autor dieses Paket aus den folgenden Gründen:

- Wesentlich bessere Architektur als beim Vorgänger AWT
- Vorversion 1.0 ist bereits abgekündigt, wird bald durch Version 1.1 ersetzt
- Schnelle Realisierung durch bereits vorhandene, umfangreiche Komponenten
- Wird der neue Standard der zukünftigen „Java 2 Platform“
- Unterstützung des nativen „Look-And-Feel“ für die wichtigsten Betriebsysteme (Windows, MacOS, CDE)

Durch die Verwendung des Model-View-Controller Entwurfschemas wurde bei Swing eine „saubere“ Trennung zwischen Oberflächen-Repräsentation und zugrundeliegender Daten realisiert. Dies bringt, neben dem schwerwiegenden Vorteil einer besseren Wartbarkeit und Übersichtlichkeit, meist eine erhebliche Verringerung des Implementierungsaufwands. In vielen Fällen müssen nur Klassen implementiert werden, die als „Adapter“ zu den die Komponente darstellenden Klassen dienen.

Eine weitere Vereinfachung in der Implementierung ergibt aus den umfangreichen Angebot an in Swing bereits enthaltenen Komponenten. Im Swing Vorgänger AWT sind nur Basis-Objekte wie *Window*, *Button*, *Checkbox* vorhanden, komplexere Oberflächenelemente wie Tabellen müssen dort vom Entwickler erst zeitaufwendig erstellt werden. Dagegen enthält Swing komplexe Objekte wie *TreeView*, *Table* und *TabPanel* für verschiedenste Anwendungen, die dann nur noch um die darzustellenden Daten angereichert werden müssen. Dies ermöglicht die Erstellung optisch und funktionell ansprechender Oberflächen in vergleichsweise geringer Zeit.

Obwohl AWT weiterhin Teil des Java Entwicklungssystems bleiben wird (tatsächlich baut Swing selbst auf AWT auf), ist bereits abzusehen, daß zukünftige Projekte direkt in Swing realisiert werden. Weiterhin wird Swing noch weiterentwickelt, während die Standardisierung von AWT in seiner jetzigen Fassung als abgeschlossen angesehen werden kann. Somit können die Oberflächenkomponenten der Implementierung noch von zukünftigen Erweiterungen profitieren.

Einer der wichtigsten Unterschiede im Umgang mit Swing im Vergleich zu AWT ist die Kodierung von Objektpositionen am Bildschirm. In AWT war es noch möglich (wenn auch nicht erwünscht), daß Objekt-Positionen und Ausmaße mit absoluten Werten in Pixel angegeben werden konnten. Dies führte meist auch zu korrekten Ergebnissen, da AWT Komponenten plattformübergreifend gleiches Aussehen hatten. Mit Swing aber unterscheidet sich durch das native „Look-And-Feel“ nicht nur das Aussehen gleicher Objekte auf unterschiedlichen Plattformen, unter Umständen müssen auch Regeln zur Positionierung der Objekte auf dem Schirm an die jeweilige Plattform angepaßt werden. Entsprechend erfolgt die Objektpositionierung und Spezifikation der Größen über eine Reihe spezieller Objekte, in denen die relativen Beziehungen der Komponenten untereinander hinterlegt werden.

Als Folge läßt sich, soweit dem Autor bekannt, keines der bisher auf dem Markt gängigen Werkzeuge zur Erstellung Graphischer Oberflächen in Java nutzen, da keines die Kodierung der Objektpositionen über relative Konstrukte zufriedenstellend beherrscht und außerdem die neuen Komponenten der Swing Bibliothek unbekannt sind.

Durch das Beta-Stadium der Swing Bibliothek weisen die Oberflächenelemente dieser Implementierung des TopologyService noch einige Fehler und Unzulänglichkeiten auf. Diese sollte aber durch eine neue Versionen der Swing Bibliothek behoben werden. Details hierzu entnehme man bitte der `RELEASE_NOTES`-Datei im Wurzelverzeichnis des Installationspakets.

3 Implementierung

Die für die konkrete Implementierung vorgegebenen Randbedingungen bestanden aus:

- In der Spezifikation definierte IDL-Schnittstellen
- Implementierung muß als Agent(en) im MASA Agentensystem ([7]) ablauffähig sein
- Implementierungssprache ist Java

3.1 Persistente Speicherung

Eine entscheidende Frage für die Implementierung war jene, wie die persistente Speicherung der Daten des TopologyService zu bewerkstelligen sei. Aus Zeitgründen wurde auf die Verwendung eines externen DBMS verzichtet. Stattdessen werden die in Java vorhandenen Serialisierungsmechanismen benutzt, um Daten persistent in Dateien auf dem lokalen Dateisystem abzulegen.

Um eine spätere Portierung auf ein DBMS zu erleichtern, sind jedoch alle Operationen auf persistent zu speichernde Daten in eigenen Klassen gekapselt. Diese Klassen tragen die Endung „`Storage`" im Namen. Während des Ablaufs des TopologyService werden alle Daten im Hauptspeicher gehalten, erst bei Terminierung des TopologyService werden alle Daten auf den Datenträger geschrieben. Jede dieser Klassen schreibt zur Speicherung eine eigene Datei. Die Ablage im Hauptspeicher wurde mit der in Java vorhandenen Klasse für Hash-Tabellen realisiert.

Die Nutzung des Hauptspeichers für den gesamten Datenbestand stellt, neben den Nachteilen für die Betriebssicherheit, sicherlich auch eine Einschränkung hinsichtlich der Skalierbarkeit der Menge der zu verwaltenden Informationen dar. Auch die Verwendung von Hash-Tabellen ist im Hinblick auf die strukturellen Zusammenhänge (Bäume und Netze) zwischen den zu verwaltenden Daten nicht als optimal zu bezeichnen. Durch den zu bewältigenden Umfang der Aufgabe und die Beschränkung des Implementierungszeitraums mußte jedoch auf, für die Implementierungsplatform bereits vorhandene und einfach zu nutzende, Mechanismen zurückgegriffen werden, wodurch sich die Wahl von Hash-Tabellen begründet.

3.2 Implementierungsarchitektur

Eine weitere Fragestellung zur konkreten Implementierung betraf die Modularisierung des Projekts. Seitens der Spezifikation wird eine solche bereits durch die diversen „Manager" (siehe Kapitel 1.2) angedeutet. Die konkret implementierte Abbildung auf MASA-Agenten wird in nachfolgender Abbildung dargestellt:

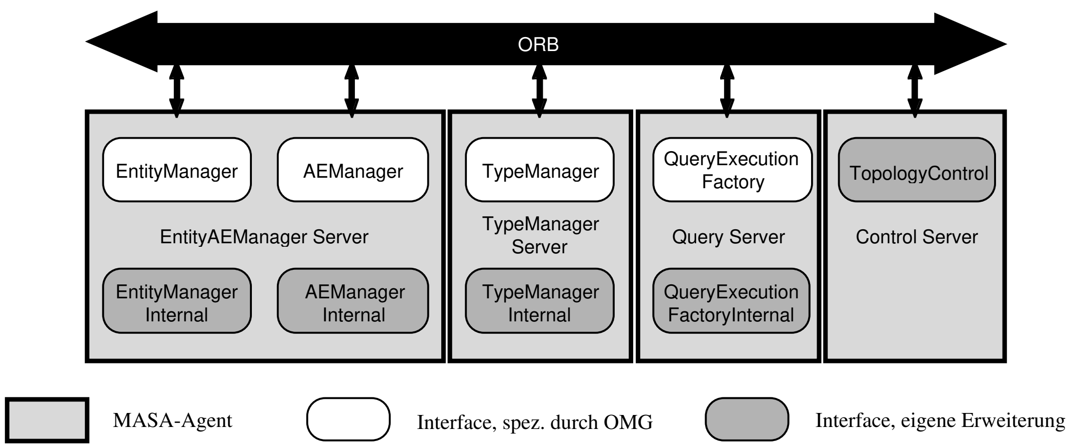

Abbildung 3.1: Implementierungsarchitektur

In dieser Architektur finden sich nun die in Abbildung 1.1 dargestellten Teile unmittelbar wieder:

- *EntityAEManagerServer* implementiert den *TopologyDataManager*
- *TypeManagerServer* implementiert den *TopologyMetaDataManager*
- *QueryServer* implementiert den *TopologyQueryManager*

Weiterhin wurde mit dem *ControlServer* eine Instanz geschaffen, die für Koordinierungsaufgaben zwischen allen Servern der Implementierung genutzt wird (näheres sie Kapitel 3.4).

Neben den bereits in der Spezifikation vorgegebenen Schnittstellen ([1], Kapitel A) wurden noch zusätzlich interne Schnittstellen geschaffen, die zur internen CORBA-Kommunikation zwischen den einzelnen Agenten dienen (siehe Abbildung 3.1 und Kapitel 3.4).

3.3 Abweichungen von der Spezifikation

Durch den Verzicht auf Verwendung eines DBMS ergeben sich einige unmittelbare Folgen für die Konformität der Implementierung zur Spezifikation:

Wegen des fehlenden DBMS konnte keine Transaktions-Behandlung in vertretbarem Zeitaufwand realisiert werden, da ein großer Teil hierfür seitens des DBMS übernommen werden würde.

Dies hat unmittelbar zur Folge, daß einzelne Methoden-Aufrufe nicht durch Transaktionen gesichert werden, d. h., daß die Möglichkeit besteht, durch einen Fehler während der Ausführung einer Methode, die interne Datenbasis des TopologyService in einen nicht konsistenten Zustand zu versetzen.

Neben der lokalen Transaktions-Sicherung ist ebenfalls keine methodenübergreifende Transaktions-Behandlung implementiert. Diese würde es ermöglichen, daß mehrere Aufrufe des TopologyService als eine einzige Transaktion behandelt werden. Essentiell ist dies vor allem für die Durchsetzung bestimmter Kardinalitätsbedingungen mittels *Rules* für *Associations*.

Nach Spezifikation ist es möglich, beispielsweise eine *Rule* zu definieren, die besagt, daß mindesten 2, aber höchsten 4 *Associations* für einen Typ X vorgeschrieben sind. Allerdings ermöglicht es die spezifizierte Methode zur Erstellung von *Entities* nicht, daß gleichzeitig mit der Erstellung auch *Associations* eingegangen werden. Somit ist es ohne methodenübergreifende Transaktionen im Zusammenhang mit der oben spezifizierten *Rule* nicht möglich, daß eine *Entity* vom Typ X angelegt wird, da nach dem Anlegen der *Entity* die *Rule* verletzt werden würde. Nachfolgender Pseudo-Code veranschaulicht dies:

1. Aufruf:	`manage( id, X)`
Aktion:	Überprüfung aller *Rules*
	⇒ Verletzung der *Rule* min=2 für Typ X
	⇒ Fehler

Mit methodenübergreifenden Transaktionen allerdings wäre dies möglich, da dann die Überprüfung aller Konsistenzeigenschaften auf das Beenden der Transaktion verschoben wird:

1. Aufruf:	`beginTransaction`
2. Aufruf:	`manage( id, X)`
3. Aufruf:	`associate( id, id2)`
4. Aufruf:	`associate( id, id3)`
5. Aufruf:	`endTransaction`
Aktion:	Überprüfung aller *Rules*
	⇒ *Rule* min=2 für Typ X ist OK
	⇒ Transaktion OK

Um nun eine Blockierung des Systems durch *Rules* mit minimaler Kardinalität größer Null zu vermeiden, wird in der vorliegenden Implementierung, unter Einschränkung der Spezifikation, eine minimale Kardinalität von Null für alle *Rules* vorgeschrieben.

Eine weitere Vereinfachung wurde im Falle der *TQL* vorgenommen. Hier wurden die iterativen Konstrukte (siehe [1], Kapitel 7.3) aus der *TQL* gestrichen. Dadurch wird allerdings die prinzipielle Leistungsfähigkeit der *TQL* nicht beschnitten, da diese Konstrukte durch mehrere einzelne *TQL* Anfragen nachgebildet werden können. Beispielsweise kann das Ergebnis der „iterativen“ *TQL*-Anfrage

```
{ AEID "A" - [0..2 ANY -] AEID "B" }
```

auch durch Zusammenfassung der Ergebnisse aus der folgenden Anfragesequenz erreicht werden:

```
{ AEID "A" - AEID "B" }
{ AEID "A" - ANY - AEID "B" }
{ AEID "A" - ANY - ANY - AEID "B" }
```

Weiterhin wird eine Parametrisierung der Anfrage (z.B. „`create( "(P){P-AEID\"B\"}", new ParameterBinding( "P", "AEID \"A\""));`“), wie in [1], Kapitel 7.2.5 beschrieben, nicht unterstützt.

Neben diesen Veränderungen der Spezifikation wurden aus Zeitgründen einige weitere Teile der Spezifikation nicht implementiert:

- Es stehen keine *Enforcerer* (siehe Kapitel 1.2.2) zu Verfügung. Die entsprechenden Methoden können zwar aufgerufen werden, lösen aber immer eine Ausnahmebehandlung aus. So wird beispielsweise bei Aufruf der Methoden `get_enforcers()` und `define_enforcers` eine `Topology::ServiceNotAvailable`-*Exception* ausgelöst.
- *Notifications*, wie in [1], Kapitel C spezifiziert, werden nicht versandt.

3.4 Erweiterungen der Spezifikation

Neben den Einschränkungen der Spezifikation wurden aber auch eine Erweiterung seitens des Autors vorgenommen. Ursächlich hierfür ist wiederum das Fehlen eines DBMS in der Implementierung.

Um ein geregeltes Abschalten des gesamten TopologyService zu gewährleisten, ist es notwendig, daß zum Beenden alle Server gleichzeitig abgeschaltet werden, um eine konsistenten Zustand der zu schreibenden Daten sicherzustellen. Um dieses gleichzeitige Abschalten zu ermöglichen wurde eigens der *ControlServer* (angeordnet im neuen IDL-Modul `TopologyPrivate`) geschaffen, der entsprechende Methoden bereithält. Weiterhin sind Methoden vorhanden, die eine Zwischensicherung der im Hauptspeicher gehaltenen Daten ermöglicht.

Die hierfür notwendige Kommunikation zwischen den einzelnen Agenten wird dabei über die internen Schnittstellen der Agenten (Abbildung 3.1) abgewickelt.

3.5 Die einzelnen Teile der Implementierung

In diesem Kapitel wird eine knappe Beschreibung der einzelnen Teile der Implementierung gegeben. Zu Beginn eines jeden Abschnitts steht eine Graphik, die die benutzte Vererbung von Schnittstellen auf der IDL-Ebene darstellt.

Die in Kapitel 4.2.1 dargestellte Problematik im Zusammenhang von CORBA/Java mit Struktur-Datentypen wird in der vorliegenden Implementierung durch eine Vorform des in Kapitel 4.2.2

gemachten Vorschlags behandelt. Kapitel 4.2.2 ist ein Ergebnis aus den während der Implementierung gemachten Erfahrungen.

Für eine detaillierte Beschreibung der einzelnen Teile sei auf die Dokumentation in den Quelldateien verwiesen. Diese ist größtenteils nach der Javadoc-Konvention im Quelltext enthalten. Dabei ist zu beachten, daß durch einen Aufruf von `javadoc` nicht alle Quelldateien dokumentiert werden, Klassen mit dem `private`-Attribut werden nicht vollständig einbezogen[2]. In diesem Fall möge der interessierte Leser einen direkten Blick in die Quelldateien werfen.

3.5.1 Server „TopologyTypeServerStationaryAgent"

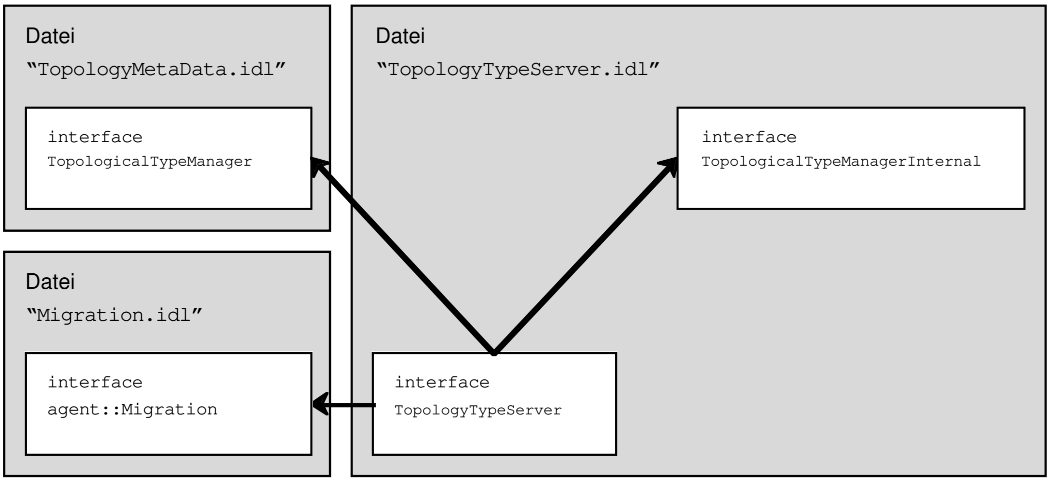

Abbildung 3.2: Schnittstellen-Vererbung des TopologyTypeServerStationaryAgent

Im Server `TopologyTypeServerStationaryAgent` wird die Schnittstelle `TopologicalTypeManager` des *TopologyMetaDataManagers* (siehe Kapitel 1.2) implementiert. Hier werden *TopologicalTypes* und *Rules* verwaltet.

Die Klassen `TypeStorage` und `RuleStorage` sind die Speicherklassen für *TopologicalTypes* und *Rules*.

Die Klasse `TopoTypeEntry` repräsentiert hierbei die *TopologicalTypes*. Neben dem Typ-Namen wird in einer Instanz auch auf die direkten Eltern- und Kind-Typen bezüglich der Typ-Hierarchie verwiesen. Eine explizite Repräsentation dieser Typ-Hierarchie in eigenen Java-Objekten existiert nicht.

---

[2] Selbst durch setzten der Option „`-private`" werden nur die Zusammenfassungen der Klassen dokumentiert. Attribute und Methoden dagegen werden für die Dokumentation vollständig ignoriert.

3.5.2 Server „TopologyEntityAEServerStationaryAgent“

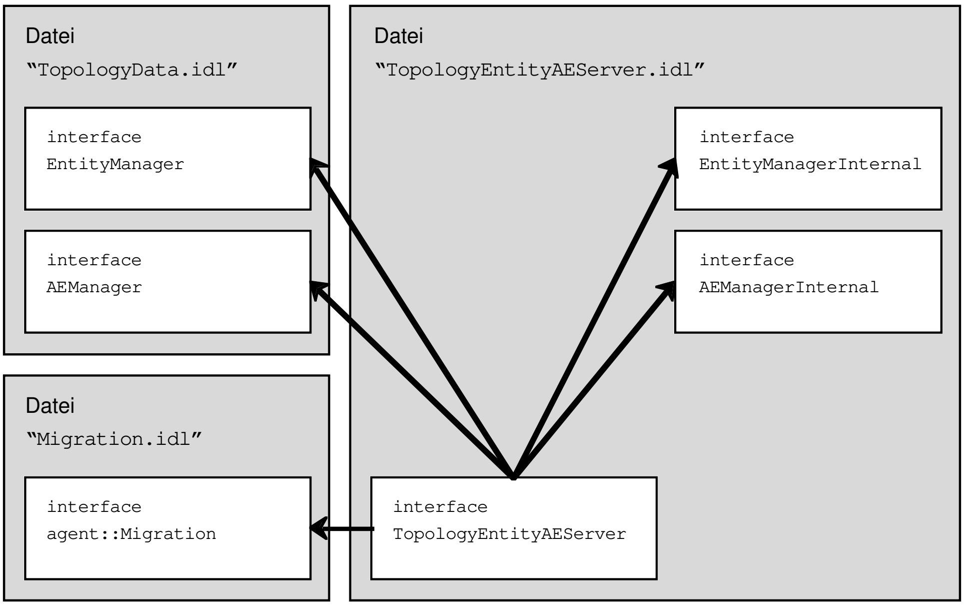

Abbildung 3.3: Schnittstellen-Vererbung des TopologyEntityAEServerStationaryAgent

Der `TopologyEntityAEServerStationaryAgent` implementiert die Schnittstellen `EntityManager` und `AEManager` des *TopologyDataManagers* (siehe Kapitel 1.2). Diese beiden Schnittstellen wurden in einem gemeinsamen Server implementiert, da beide im wesentlich auf den gleichen Daten operieren.

Die Klasse `EntityStorage` übernimmt die persistente Speicherung der *Entities*.

`TopologicalEntityInternal` ist eine Erweiterung der in der Spezifikation definierten Klasse. Dort werden neben den in Spezifikation definierten Attributen auch noch die *Associations* und *AggregationTies* der *Entity* gespeichert. *Associations* und *AggregationTies* werden ausschließlich implizit bei den beteiligten *Entities* gespeichert, eine explizite Repräsentierung der einzelnen Relationen in eigenen Java-Objekten existiert nicht.

Da *AggregateEntities* in Abfragen durch *TQL* eine entscheidende Rolle spielen, wurde noch die Hilfsklasse `AggregCache` implementiert. Diese speichert die transitiven Hüllen der *AggregationTies*. *Entities*, die an einer Aggregation beteiligt sind, enthalten einen Verweis auf die transitive Hülle ihrer *AggregationTies* in `AggregCache`. So ist es, ausgehend von einer beliebigen *Entity*, möglich, ohne weitere Berechnungen alle anderen an einer *Aggregation* beteiligten *Entities* herauszufinden. Diese Technik ist zwar aufwendig bei der Bildung und Auflösung von *Aggregation* Relationen, da dann die Hüllen jeweils neu berechnet werden müssen. Unter der Annahme, daß TopologyService in der Anwendung eher abfragelastig ist, ist es legitim zu vermuten, daß sich diese Kosten durch die Beschleunigung von Abfragen aber wieder amortisieren. Unter Betrachtung des Speicherbedarfs wäre die Benutzung von Union-Find-Datenstrukturen und zugehöriger Algorithmen für die Repräsentierung von *AggregationTies* und ihrer transitiven Hüllen effizienter, wiederum aufgrund der begrenzten Zeit wurde darauf aber verzichtet. Ohnehin würde vermutlich bei Verwendung eines DBMS ein großer Teil des Problems in das DBMS verlagert werden.

3.5.3 Server „TopologyQueryServerStationaryAgent“

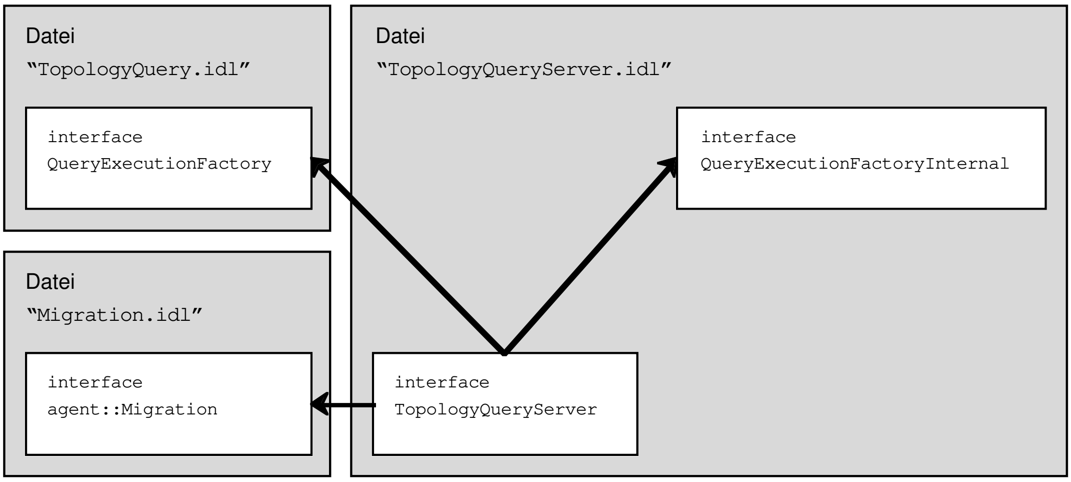

Abbildung 3.4: Schnittstellen-Vererbung des TopologyQueryServerStationaryAgent

Die in der Spezifikation definierte Schnittstelle `QueryExecutionFactory` des *TopologyQueryManagers* (siehe Kapitel 1.2) , zuständig für die Abfrage der Datenbasis des TopologyService, wird im CORBA-Server `TopologyQueryServerStationaryAgent` implementiert.

Dieser Server erzeugt, wie der Name der Schnittstelle bereits impliziert, für jede Datenbasisabfrage eine Instanz von `QueryExecution`. Die Anfrage selbst ist in *TQL* zu formulieren. *TQL* wurde hier in leicht reduzierter Form (siehe Kapitel 3.3) implementiert, konkret:

```
<TQL Query>          :=        <Query Body>
<Query Body>         :=        '{' <Item List> '}'
<Item List>          :=        <AE Headed List>
<Meta AE>            :=        <AE Pattern>
<AE Pattern>         :=        <AEP>
                            |  <AEP> <Tag List>
<AEP>                :=        'AEID' <XFN>
                            |  'ANY'
                            |  <Type Expression>
<Tag List>           :=        <Tag>
                            |  <Tag> <Tag List>
<Tag>                :=        'TAGGED' <XFN>
<Type Expression>    :=        <XFN>
                            |  <Type Expression> '&' <Type Expression>
                            |  <Type Expression> '|' <Type Expression>
                            |  '!' <Type Expression>
                            |  '(' <Type Expression> ')'
<Assoc Headed List> :=         <Assoc Pattern> <AE Headed List>
<AE Headed List>     :=        <Meta AE> <Assoc Headed List>
                            |  <Meta AE>
<Assoc Pattern>      :=        '-'
                            |  <Role> '-' <Role>
                            |  <Role> '-'
                            |  '-' <Role>
<Role>               :=        '<' <XFN> '>'
<XFN>                :=        '"' [any character]* '"'
```

EBNF-Grammatik: Implementierte Form der *TQL*

Die Bearbeitung einer Anfrage geschieht nun folgendermaßen:

Zuerst wird die *TQL* Eingabe mittels der in der Klasse `QParser` implementierten Methoden lexikalisch und syntaktisch analysiert. Dabei fordert der Parser Schritt für Schritt von der Methode `Lexan()` ein Token (`QToken`) an, das dann auf syntaktische Korrektheit im Kontext überprüft wird. Lexikalische Analyse und Parser wurden ohne die Verwendung von Werkzeugen manuell aus der Grammatik erstellt, auf die Verwendung von Compiler-Tools, wie beispielsweise `javacc`, wurde verzichtet.

Ergebnis dieses Durchgangs ist eine Sequenz von Instanzen von `QCmd`. `QCmd` formuliert Befehle einer einfachen Kellermaschine (`QueryExecMachine` und `QExecStack`), die dann das Ergebnis der Anfrage ermittelt. Weder in der `QueryExecMachine` noch im vorhergehenden Analysedurchgang werden Optimierungen (z. B. der boolschen *TQL* Konstrukte) vorgenommen. Solche Optimierungen wären zu aufwendig zu implementieren und werden bei Verwendung eines DBMS in der Regel von diesem übernommen.

Die Implementierung von `QueryExecution` ist momentan nicht nebenläufig, d. h. es kann nur eine Abfrage gleichzeitig von `TopologyQueryServerStationaryAgent` ausgewertet werden. Der Grund für diese Einschränkung ist die begrenzte Implementierungszeit und sollte bei Überarbeitung der Implementierung behoben werden. Sowohl die Java-Mechanismen für die Implementierung von Threads als auch die Benutzung eines DBMS sind hierbei sicherlich hilfreich.

3.5.4 Server „TopologyControlServerStationaryAgent"

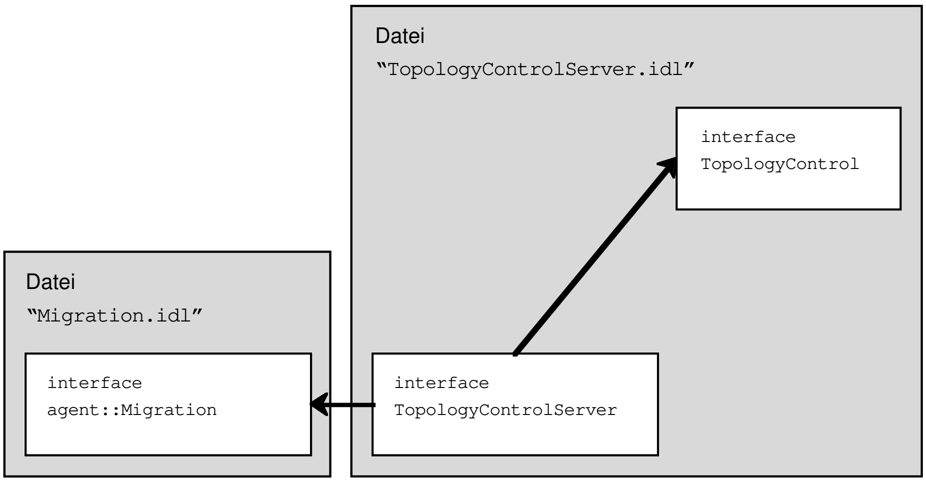

Abbildung 3.5: Schnittstellen-Vererbung des TopologyControlServerStationaryAgent

`TopologyControlServerStationaryAgent` implementiert die in Kapitel 3.4 beschriebenen Erweiterungen zur globalen Steuerung des TopologyService.

3.5.5 Gemeinsame Quell-Dateien

Im Java Package `TopologyPrivate` befinden sich neben dem `TopologyControlServerStationaryAgent` noch einige weitere Klassen, die innerhalb der gesamten Implementierung genutzt werden.

Alle Konstanten der Implementierung sind in der Klasse `Constants` zu finden. Die Namen der Java Properties für alle Server finden sich in der Klasse `PropertyNames`.

In den Klassen `API` und `API_Internal` finden sich Hilfsfunktionen, die Referenzen auf die verschiedenen Schnittstellen der Server zurückliefern. Diese Methoden wurden geschaffen, um an einer zentralen Stelle die Mechanismen für die Erlangung der Server-Referenzen zu schaffen. Die konkrete Implementierung unterscheidet sich dabei für die Varianten `STANDALONE` und `MASA_AGENT`.

Globale Hilfsfunktionen sind als statische Methoden in der Klasse `Tools` implementiert. Hier finden sich unter anderem Methoden für die Behandlung von XFNs, zum Auslesen von Java Properties, CORBA-Hilfsfunktionen und alle Funktionen zur Ausgabe von Status- und Fehlermeldungen.

3.5.6 TopoApplet

Das nach MASA-Spezifikation vorgeschriebene Steuerungs-Applet wurde unter Zuhilfenahme der GUI-Bibliothek Swing (siehe Kapitel 2.3) implementiert.

Obwohl die Implementierung des gesamten TopologyService aus vier getrennten Agenten besteht, wurde nur ein gemeinsames Steuerungs-Applet geschaffen.

Neben einer „1:1“-Abbildung der von TopologyService zur Verfügung gestellten Methoden auf eine graphische Oberfläche, enthält das Applet vor allem die Möglichkeit, die Datenbasis aller vier Server zu laden und zu speichern, sowie die geregelte Abschaltung des gesamten Services.

Neben der Klasse `TopologyServiceApplet`, der wichtigsten Klasse des Applets, sind eine ganze Reihe von Verfeinerungen diverser Swing-Klassen vorhanden. Eine einzelne Beschreibung aller Klassen würde an dieser Stelle zu weit führen, dem geübten Leser erschließen sich die Klassen aber unmittelbar durch einen Blick auf den Quelltext.

3.5.7 Test-Applikation

Ausschließlich für Test- und Debugging-Zwecke wurde eine textorientierte Java Applikation implementiert. Diese bildet, wie das Steuerungs-Applet, einfach die Methoden des „1:1“ ab. Weiterhin enthält die Test-Applikation noch eine Möglichkeit automatisch eine Typ-Hierarchie und *Entities* anzulegen.

Die Test-Applikation wurde nur während der Entwicklungsphase benötigt. Sie wird nun, mit Ausnahme der automatischen Erzeugung von *TopologicalTypes* und *Entities*, vollständig durch das Steuerungs-Applet ersetzt.

4 Anmerkungen zu Java und CORBA

4.1 Referenzen in Java

4.1.1 Einführung in die Problematik

Obwohl jedem erfahrenen Entwickler der Unterschied zwischen einer Referenz und dem Datum, auf das die Referenz verweist, geläufig ist, möchte der Autor an dieser Stelle nochmals diese Thematik im Kontext von Java aufgreifen.

Datentypen in Java werden prinzipiell in die beiden großen Kategorien der primitiven Datentypen (z. B. `boolean`, `int`, `float`, etc.) und der Objekte (Klassen und ihre Instanzen) aufgeteilt. Primitive Datentypen werden dabei immer direkt angesprochen, Objekt-Instanzen werden generell über Referenzen angesprochen. Referenzen auf primitive Datentypen (wie in C++ durchaus üblich) sind nicht möglich, ebensowenig sind arithmetische Operationen auf Referenzen (Adreßrechnung) zugelassen.

Generell ist dieses Vorgehen nicht als problematisch anzusehen, im Gegenteil, es ermöglicht die Zusicherung gewisser Sicherheitseigenschaften. Exemplarisch sei hier nur erwähnt, daß es so unmöglich ist, auf einen beliebigen Speicherbereich über eine „verbogene" Referenz zuzugreifen oder Laufzeitfehler durch ungültig gewordene Referenzen zu erzeugen („dangling pointer").

Durch die strenge Trennung von primitiven Datentypen ohne Referenzen und Referenzen auf Objekt-Instanzen vereinfacht sich die Programmiersprache auch auf der syntaktischen Ebene. Spezielle Operatoren für Deklaration von Referenzen, Referenzierung und Dereferenzierung, wie beispielsweise in C++ („`*`", „`&`", „`->`", etc.), sind nicht mehr notwendig. Der Programm-Quelltext sieht einfacher aus, selbstverständlich bleibt aber der semantische Unterschied zwischen direkten Daten und Referenzen bestehen.

Leider wird in Java dieser Unterschied durch eine Eigenheit der Sprache ein wenig verschleiert. Generell wird in Java strikt zwischen der mittels einer Klasse implementierten Semantik und der Java zugrundeliegenden Semantik unterschieden. Dies zeigt sich beispielsweise daran, daß das Überladen von Operatoren in Java nicht möglich ist. Aber obwohl `String`, der Datentyp für Zeichenketten, eine Klasse darstellt sind einige Ausdrücke möglich, wie sonst nur auf primitiven Datentypen zu Verfügung stehen, d.h. hier wird die Semantik der Klasse mit jener der Sprache vermischt.

Die folgende Form der Instanziierung der Klasse `String` erweckt den Eindruck, daß es sich um eine Zuweisung zu einem primitiven Datentyp handelt:

```
String beispiel;
beispiel = "Nun gut, wer bist Du denn?";
```

Code-Fragment 4.1

Die, im Kontext der Java Syntax betrachtet, korrekte Form lautet:

```
beispiel = new String("Nun gut, wer bist Du denn?");
```

Code-Fragment 4.2

Die zweite Ausnahme betrifft die Konkatenierung zweier Zeichenketten:

```
1    String str1, str2;
2    str1 = “Ein Teil von jener Kraft,”;
3    str2 = “die stets das Böse will und stets das Gute schafft.”;
4    str3 = str1 + str2;
```

Code-Fragment 4.3

Dieses Code-Fragment zeigt eine markante Ausnahme in Java. Obwohl die Überladung von Operatoren in Java nicht möglich ist, handelt es sich in Zeile 4 tatsächlich um eine Form der Überladung. Eigentlich sieht der erzeugte Code folgendermaßen aus:

```
4    str3 = str1.concat( str2);
```

Code-Fragment 4.4

Vor den Hintergrund, daß eine Überladung eigentlich nie erfolgen kann, erweckt diese Ausnahme wiederum den Anschein, daß es sich bei `String` um einen primitiven Datentyp handelt. In den beiden folgenden Abschnitten sollen mögliche Folgen dieser Verwischung der Grenzen zwischen primitiven Datentypen und Referenzen näher betrachtet werden.

4.1.2 Vergleichs-Operationen

Nun Stellt sich die Frage warum jene Aspekte an dieser Stelle so hervorgehoben werden. Läßt man sich von obigen Ausnahmen täuschen und ist sich der Tatsache nicht bewußt, daß es sich bei `String` um eine Klasse handelt, so könnte nun der Vergleich auf Identität zweier `String` Variablen so aussehen:

```
String str1, str2;
str1 = “Das also war des Pudels Kern!”;
str2 = “Das also war des Pudels Kern!”;
boolean isIdent = (str1 == str2);
```

Code-Fragment 4.5

Leider entspricht dieses Fragment nicht der gewünschten Semantik, nämlich dem Vergleich der beiden Zeichenketten. Vielmehr wird überprüft, ob die Referenzen `str1` und `str2` auf die gleiche Objektinstanz zeigen. Entsprechend lautet das Ergebnis des Vergleichs `false`. Für die gewünschte Semantik ist eine entsprechende Methode der Klasse `String` zu verwenden:

```
boolean isIdent = str1.equals( str2);
```

Code-Fragment 4.6

4.1.3 Parameterübergabe

Nun soll anhand eines Beispiels zur Parameterübergabe nochmals die Problematik der Unterscheidung zwischen einer Referenz und der referenzierten Instanz gezeigt werden.

Wir nehmen an, daß eine Klasse `Server` einen Struktur-Datentyp `Idlstruct` als klassen-internes Datum vorhält und dieses Datum nur zu informellen Zwecken nach außen geben soll, d.h. die Daten sollen von außen nicht geändert werden können. Die Elemente von `Idlstruct` seien jeweils ein Wert vom Typ `short` und `String`.

Da Java keine eigenen Struktur-Datentypen kennt (im Gegensatz zu C++), soll dieser hier in einer Klasse ohne Methoden nur mit den Elementen der Struktur als Attribute implementiert werden:

```
class Idlstruct
{
    public short    theShort;
    public String   theString;
}
```

Code-Fragment 4.7

Eine naive Implementierung der Klasse `Server` könnte so aussehen:

```
1   class Server
2   {
3       private Idlstruct     internalStruct;
4       …
5       public Idlstruct getValue()
6       {
7           return internalStruct;
8       }
9   }
```

Code-Fragment 4.8

Diese Fassung erfüllt aber die Forderung nach Unveränderlichkeit außerhalb der Klasse des internen Datums `internalStruct` nicht.

Zur Begründung ist zunächst festzustellen, daß es sich bei `internalStruct` um eine Referenz auf eine Instanz der Klasse `Idlstruct` handelt. Zwar sichert das Attribut `private` zu, daß eine direkte Manipulation der Form

```
serverInstance.internalStruct.theShort = 13;
```

Code-Fragment 4.9

von außen unmöglich ist, dennoch ist es über die Methode `getValue`() möglich die Elemente von `internalStruct` zu verändern. Zeile 7 gibt nämlich eine Referenz auf das interne Objekt zurück und mit

```
serverInstance.getValue().theShort = 13;
```

Code-Fragment 4.10

würde der Wert des Elements `theShort` verändert werden. Da `getValue()` als `public` deklariert ist, steht dieser Weg immer offen, die eigentlich internen Datenstruktur `internalStruct` ist einer Veränderung preisgegeben.

Eine bessere Implementierung könnte in Zeile 7 so aussehen:

```
return new Idlstruct( theShort, new String( theString));
```

Code-Fragment 4.11

Nun wird, anstatt eine Referenz auf das interne Objekt zurückzugeben, zuerst eine Kopie des internen Objekts erstellt und eine Referenz darauf an den Aufrufer zurückgegeben. Somit wird bei einer Manipulation immer nur die Kopie verändert, nie das interne Original und die gestellte Forderung ist erfüllt.

Der Leser mag jetzt entgegnen, daß dieses Beispiel, besonders die Implementierung einer methodenlosen Klasse als Ersatz für Struktur-Datentypen, keine vorbildliche Umsetzung für die objekt-orientierte Programmiersprache Java darstellt, daß man vielmehr entsprechende `Set/Get`-Methoden anbieten würde, die entsprechend der gewünschten Zugriffsstruktur mit passenden Visibilitäts-Attributen versehen werden sollten. Der Autor stimmt dem zu, hat dieses Beispiel jedoch gewählt, da genau diese Implementierungstechnik im Zusammenhang mit CORBA genutzt wird, um IDL-Struktur-Datentypen auf Java abzubilden.

Diese Abbildung ist auch solange unkritisch, wie nur über CORBA-Schnittstellen mit dem Server kommuniziert wird. CORBA stellt nämlich sicher, daß bei Parameterrückgabe eines Struktur-Datentyps dieser by-value zurückgegeben wird und nicht by-reference. Benutzt man allerdings innerhalb des Servers, die gleichen Methoden, die für die Implementierung der CORBA-Schnittstellen genutzt werden, über einen reinen Java Methodenaufruf und nicht über CORBA, so findet die Übergabe by-reference statt, es wird schlicht die Referenz auf die Objektinstanz zurückgegeben. Dieses Szenario ist nicht als untypisch anzusehen, da man aus Geschwindigkeits-Gründen innerhalb eines Servers direkte Methodenaufrufe bevorzugen würde.

Abschließend sei festgestellt, daß die vermeindlich einfache Struktur der Sprache Java den Entwickler nicht davon entbindet, sich über die Natur von Datentypen und Parameterübergabe voll bewußt zu sein.

4.1.4 Ausführliches Beispiel-Programm

Ergänzend zum vorhergehenden Kapitel sei hier ein Satz kleiner Beispiel-Programme angegeben, die die geschilderte Problematik nochmals ausführlich illustrieren. Sie seien hier nur zur Nachvollziehung durch den Leser ohne ausführliche Beschreibung angegeben.

Die Klassen `ServerImpl` und `Server2Impl` implementieren die CORBA-Schnittstellen, welche in `structdemo.idl` definiert sind.

`ServerImpl` enthält einige Methoden, die zeigen wie Parameterrückgabe und Vergleichsmethoden implementiert werden könnten.

`Server2Impl` richtet sich, zusammen mit `Idlstruct_Internal.java`, nach den Vorschlägen aus Kapitel 4.2.2.

`Client` greift dann sowohl über die CORBA-Schnitstelle, als auch über direkte Methodenaufrufe auf `ServerImpl` zu und veranschaulicht die Wirkungsweise der unterschiedlichen Implementierungen und benutzten Schnittstellen. Die Ausgabe von `Client` lautet:

```
==============================================================================================
Testing with Server
==============================================================================================



----------------------------------------------------------------------------------------------
 Communication via CORBA reference...

  Via 'getValue_GOOD': Using the returned reference. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_GOOD': Structs are equal. OK.

  Via 'getValue_GOOD': Using a cloned object. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_GOOD': Structs are equal. OK.

  Via 'getValue_GOOD': Returned object was changed. Comparing should return 'false'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. OK.
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. OK.
      Compare with 'isEqualToInternal_GOOD': Structs NOT equal. OK.


  Via 'getValue_BAD': Using the returned reference. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_GOOD': Structs are equal. OK.

  Via 'getValue_BAD': Returned object was changed. Comparing should return 'false'.
    Fine, the server ist untouched.
```

```
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. OK.
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. OK.
      Compare with 'isEqualToInternal_GOOD': Structs NOT equal. OK.


------------------------------------------------------------------------------------------------
 Communication via JAVA reference...

  Via 'getValue_GOOD': Using the returned reference. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_GOOD': Structs are equal. OK.

  Via 'getValue_GOOD': Using a cloned object. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. BOOM!
      Compare with 'isEqualToInternal_GOOD': Structs are equal. OK.

  Via 'getValue_GOOD': Returned object was changed. Comparing should return 'false'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs NOT equal. OK.
      Compare with 'isEqualToInternal_BAD2': Structs NOT equal. OK.
      Compare with 'isEqualToInternal_GOOD': Structs NOT equal. OK.


  Via 'getValue_BAD': Using the returned reference. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal_BAD1': Structs are equal. OK.
      Compare with 'isEqualToInternal_BAD2': Structs are equal. OK.
      Compare with 'isEqualToInternal_GOOD': Structs are equal. OK.

  Via 'getValue_BAD': Returned object was changed. Comparing should return 'false'.
    Oops, we altered internal data :-(
      Compare with 'isEqualToInternal_BAD1': Structs are equal. BOOM!
      Compare with 'isEqualToInternal_BAD2': Structs are equal. BOOM!
      Compare with 'isEqualToInternal_GOOD': Structs are equal. BOOM!


================================================================================================
Testing with Server2
================================================================================================


------------------------------------------------------------------------------------------------
 Communication via CORBA reference...

  Via 'getValue': Using the returned reference. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal': Structs are equal. OK.

  Via 'getValue': Using a cloned object. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal': Structs are equal. OK.

  Via 'getValue': Returned object was changed. Comparing should return 'false'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal': Structs NOT equal. OK.


------------------------------------------------------------------------------------------------
 Communication via JAVA reference...

  Via 'getValue': Using the returned reference. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal': Structs are equal. OK.

  Via 'getValue': Using a cloned object. Comparing should return 'true'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal': Structs are equal. OK.

  Via 'getValue': Returned object was changed. Comparing should return 'false'.
    Fine, the server ist untouched.
      Compare with 'isEqualToInternal': Structs NOT equal. OK.
```

4.1.4.1 IDL-Quelltext „structdemo.idl“

```
module demo
{
        struct Idlstruct
        {
                short           theShort;
                string          theString;
        };

        interface Server
        {
                boolean         isUnchanged();

                Idlstruct       getValue_BAD();
                Idlstruct       getValue_GOOD();

                boolean         isEqualToInternal_BAD1( in Idlstruct inStruct);
                boolean         isEqualToInternal_BAD2( in Idlstruct inStruct);
                boolean         isEqualToInternal_GOOD( in Idlstruct inStruct);
```

```
        };

        interface Server2
        {
                boolean         isUnchanged();

                Idlstruct       getValue();

                boolean         isEqualToInternal( in Idlstruct inStruct);
        };
};
```

4.1.4.2 Java-Quelltext „ServerImpl.java"

```
// Demonstrantion on the danger of dealing with objects
// in conjuncton with IDL structs.
//
// V 0.1 by Harald Roelle, December 1998
//

package demo;

class ServerImpl extends _ServerImplBase
{
    private Idlstruct     internalStruct;

    public static void main( String[] args)
    {
        org.omg.CORBA.ORB orb        = org.omg.CORBA.ORB.init( args, new java.util.Properties());
        org.omg.CORBA.BOA boa        = orb.BOA_init( args, new java.util.Properties());

        ServerImpl        serverInstance = new ServerImpl();

        try {
            String              ref     = orb.object_to_string( serverInstance);
            String              refFile = "Server.ref";
            java.io.PrintWriter out     = new java.io.PrintWriter(
                                                new java.io.FileOutputStream( refFile));
            out.println(ref);
            out.flush();
        }
        catch( java.io.IOException ex) {
            System.err.println( "Can't write to '" + ex.getMessage() + "'");
            System.exit( 1);
        }

        boa.impl_is_ready( null);
    }


    ServerImpl()
    {
        // Initialize the internal value of the struct.
        // These values are really internal and should be given to
        // clients for informational purposes only.
        // I.e. a client MUST NOT be able to change them.
        internalStruct = new Idlstruct( (short)42, "final solution");
    }


    public Idlstruct getValue_BAD()
    {
        // BOOM! (in case of java interface)
        // Here a reference to the original object is returned.
        // So a client is able to change the contents of the struct
        // via the reference, because the members are public.
        return internalStruct;
    }

    public Idlstruct getValue_GOOD()
    {
        // This one is safe.
        // We make a copy of our internal instance.
        // Changing any value in the copy is no harm for our internal one.
        // Remember: A 'String' is an object!
        return new Idlstruct( internalStruct.theShort, new String( internalStruct.theString));
    }


    public boolean isEqualToInternal_BAD1( Idlstruct inStruct)
    {
        // BOOM!
        // No checking on equality of the struct's values is done.
        // Actually this tells us if the given reference points
        // to the same object instance as our internal reference does.
        return ( internalStruct == inStruct );
    }

    public boolean isEqualToInternal_BAD2( Idlstruct inStruct)
    {
        // (Semi)BOOM!
        // Here we remembered to check for equality of the struct's
        // members, but unfortunally we forgot the fact that the Java data
        // type 'String' is an object, even if the Java syntax suggests
        // it to be a primitive data type.
        // So the same mistake is done as in the example above, only one
        // level deeper.
        if ( internalStruct.theShort != inStruct.theShort )
            return false;
        else if ( internalStruct.theString != inStruct.theString )
```

```
            return false;
        else
            return true;
    }

    public boolean isEqualToInternal_GOOD( Idlstruct inStruct)
    {
        // This one is good.
        // With primitive data types equality is checked via the built in
        // Java operator and the equality of objects is checked using an
        // appropiate method of the class.
        if ( internalStruct.theShort != inStruct.theShort )
            return false;
        else if ( ! internalStruct.theString.equals( inStruct.theString) )
            return false;
        else
            return true;
    }


    public boolean isUnchanged()
    {
        if ( internalStruct.theShort != (short)42 )
            return false;
        else if ( ! internalStruct.theString.equals( "final solution") )
            return false;
        else
            return true;
    }
}
```

4.1.4.3 Java-Quelltext „Client.java”

```
package demo;

class Client
{
    private static void PrintCompareMessage( String inCompMethod,
                                             boolean inResult, boolean inExpect)
    {
        System.out.print( "      Compare with '"+inCompMethod+"': ");

        if ( inResult )
            System.out.print( "Structs are equal.");
        else
            System.out.print( "Structs NOT equal.");

        if ( inResult == inExpect )
            System.out.println( " OK.");
        else
            System.out.println( " BOOM!");
    }


    private static void DoEqualTestsServer( String inRefMethod, String inMessage,
                                            Server inServer, Idlstruct inRef,
                                            boolean inExpectEqual)
    {
        System.out.println( "  Via '"+inRefMethod+"': "+ inMessage +
                            " Comparing should return '"+inExpectEqual+"'.");

        if ( inServer.isUnchanged() )
            System.out.println( "    Fine, the server ist untouched.");
        else
            System.out.println( "    Oops, we altered internal data :-(");

        PrintCompareMessage( "isEqualToInternal_BAD1", inServer.isEqualToInternal_BAD1( inRef),
                             inExpectEqual);
        PrintCompareMessage( "isEqualToInternal_BAD2", inServer.isEqualToInternal_BAD2( inRef),
                             inExpectEqual);
        PrintCompareMessage( "isEqualToInternal_GOOD", inServer.isEqualToInternal_GOOD( inRef),
                             inExpectEqual);
    }


    private static void DoEqualTestsServer2(    String inRefMethod, String inMessage,
                                                Server2 inServer, Idlstruct inRef,
                                                boolean inExpectEqual)
    {
        System.out.println( "  Via '"+inRefMethod+"': "+ inMessage +
                            " Comparing should return '"+inExpectEqual+"'.");

        if ( inServer.isUnchanged() )
            System.out.println( "    Fine, the server ist untouched.");
        else
            System.out.println( "    Oops, we altered internal data :-(");

        PrintCompareMessage( "isEqualToInternal", inServer.isEqualToInternal( inRef),
                             inExpectEqual);
    }


    // get a reference to an object via IOR from a file
    private static org.omg.CORBA.Object GetCorbaRef( org.omg.CORBA.ORB inOrb, String refFile)
    {
        String ref = null;
        try {
            java.io.BufferedReader in = new java.io.BufferedReader(
                                             new java.io.FileReader( refFile));
            ref = in.readLine();
```

```
        }
        catch(java.io.IOException ex) {
            System.err.println("Can't read from '" + ex.getMessage() + "'");
            System.exit(1);
        }
        return inOrb.string_to_object( ref);
    }


    public static void main( String args[])
    {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, new java.util.Properties());


        Server  corbaServer  = ServerHelper.narrow( GetCorbaRef( orb, "Server.ref"));
        Server2 corbaServer2 = Server2Helper.narrow( GetCorbaRef( orb, "Server2.ref"));

        ServerImpl  javaServer  = new ServerImpl();
        Server2Impl javaServer2 = new Server2Impl();



        System.out.println( "================================================================");
        System.out.println( "Testing with Server");
        System.out.println( "================================================================");


        System.out.println( "");
        System.out.println( "");
        System.out.println( "");
        System.out.println( "----------------------------------------------------------------");
        System.out.println( " Communication via CORBA reference...");
        System.out.println( "");

        Idlstruct corbaRef1 = corbaServer.getValue_GOOD();
        DoEqualTestsServer( "getValue_GOOD", "Using the returned reference.",
                            corbaServer, corbaRef1, true);

        System.out.println( "");

        Idlstruct corbaRef1Clone = new Idlstruct( corbaRef1.theShort,
                                                  new String( corbaRef1.theString));
        DoEqualTestsServer( "getValue_GOOD", "Using a cloned object.",
                            corbaServer, corbaRef1Clone, true);

        System.out.println( "");

        corbaRef1.theShort = 13;
        corbaRef1.theString = "Java strikes back";
        DoEqualTestsServer( "getValue_GOOD", "Returned object was changed.",
                            corbaServer, corbaRef1, false);


        System.out.println( "");
        System.out.println( "");

        Idlstruct corbaRef2 = corbaServer.getValue_BAD();
        DoEqualTestsServer( "getValue_BAD", "Using the returned reference.",
                            corbaServer, corbaRef2, true);

        System.out.println( "");

        corbaRef2.theShort  = 13;
        corbaRef2.theString = "Java strikes back";
        DoEqualTestsServer( "getValue_BAD", "Returned object was changed.",
                            corbaServer, corbaRef2, false);


        System.out.println( "");
        System.out.println( "");
        System.out.println( "");
        System.out.println( "----------------------------------------------------------------");
        System.out.println( " Communication via JAVA reference...");
        System.out.println( "");

        Idlstruct javaRef1 = javaServer.getValue_GOOD();
        DoEqualTestsServer( "getValue_GOOD", "Using the returned reference.",
                            javaServer, javaRef1, true);

        System.out.println( "");

        Idlstruct javaRef1Clone = new Idlstruct( javaRef1.theShort,
                                                 new String( javaRef1.theString));
        DoEqualTestsServer( "getValue_GOOD", "Using a cloned object.",
                            javaServer, javaRef1Clone, true);

        System.out.println( "");

        javaRef1.theShort  = 13;
        javaRef1.theString = "Java strikes back";
        DoEqualTestsServer( "getValue_GOOD", "Returned object was changed.",
                            javaServer, javaRef1, false);

        System.out.println( "");
        System.out.println( "");

        Idlstruct javaRef2 = javaServer.getValue_BAD();
        DoEqualTestsServer( "getValue_BAD", "Using the returned reference.",
                            javaServer, javaRef2, true);

        System.out.println( "");
```

```
        javaRef2.theShort = 13;
        javaRef2.theString = "Java strikes back";
        DoEqualTestsServer( "getValue_BAD", "Returned object was changed.",
                            javaServer, javaRef2, false);


        System.out.println( "");
        System.out.println( "");
        System.out.println( "");
        System.out.println( "=================================================================");
        System.out.println( "Testing with Server2");
        System.out.println( "=================================================================");

        System.out.println( "");
        System.out.println( "");
        System.out.println( "");
        System.out.println( "-----------------------------------------------------------------");
        System.out.println( " Communication via CORBA reference...");
        System.out.println( "");

        Idlstruct corbaRef3 = corbaServer2.getValue();
        DoEqualTestsServer2( "getValue", "Using the returned reference.",
                             corbaServer2, corbaRef3, true);

        System.out.println( "");

        Idlstruct corbaRef3Clone = new Idlstruct( corbaRef3.theShort,
                                                  new String( corbaRef3.theString));
        DoEqualTestsServer2( "getValue", "Using a cloned object.",
                             corbaServer2, corbaRef3Clone, true);

        System.out.println( "");

        corbaRef3.theShort  = 13;
        corbaRef3.theString = "Java strikes back";
        DoEqualTestsServer2( "getValue", "Returned object was changed.",
                             corbaServer2, corbaRef3, false);


        System.out.println( "");
        System.out.println( "");
        System.out.println( "");
        System.out.println( "-----------------------------------------------------------------");
        System.out.println( " Communication via JAVA reference...");
        System.out.println( "");

        Idlstruct javaRef3 = javaServer2.getValue();
        DoEqualTestsServer2( "getValue", "Using the returned reference.",
                             javaServer2, javaRef3, true);

        System.out.println( "");

        Idlstruct javaRef3Clone = new Idlstruct( javaRef3.theShort,
                                                 new String( javaRef3.theString));
        DoEqualTestsServer2( "getValue", "Using a cloned object.",
                             javaServer2, javaRef3Clone, true);

        System.out.println( "");

        javaRef3.theShort  = 13;
        javaRef3.theString = "Java strikes back";
        DoEqualTestsServer2( "getValue", "Returned object was changed.",
                             javaServer2, javaRef3, false);

        System.out.println( "");
    }
}
```

4.1.4.4 Java-Quelltext „ Idlstruct_Internal.java”

```
package demo;

class Idlstruct_Internal implements ExtendedIDLStruct
{
    // The default constructer just initializes to default values
    public Idlstruct_Internal()
    {
        theShort  = 0;
        theString = null;
    }

    // This one is equivalent to the one in the external type
    public Idlstruct_Internal( short _ob_a0, String _ob_a1)
    {
        theShort  = _ob_a0;
        theString = _ob_a1;
    }

    // Construct an internal type from the external one
    public Idlstruct_Internal( Idlstruct inIdlstruct)
    {
        theShort  = inIdlstruct.theShort;
        theString = inIdlstruct.theString;
    }


    // Compare to an internal type
    public boolean equals( Idlstruct_Internal inIdlstruct)
    {
        // Primitive types are compared via the built in operator.
```

```
        if ( theShort != inIdlstruct.theShort )
            return false;
        // Class types rely on an equals() method.
        else if ( ! theString.equals( inIdlstruct.theString ) )
            return false;
        else
            return true;
    }

    // Compare to an external type
    public boolean equals( Idlstruct inIdlstruct)
    {
        // This implementation just relies on the equals() for internal
        // types. For speed/memory reasons it's also possible to
        // implement it redundant to the equals() for internal types.
        return equals( new Idlstruct_Internal( inIdlstruct));
    }

    // Compare two external types
    public static boolean equals( Idlstruct inIdlstruct1, Idlstruct inIdlstruct2)
    {
        // This implementation just relies on the equals() for internal
        // types. For speed/memory reasons it's also possible to
        // implement it redundant to the equals() for internal types.
        return (new Idlstruct_Internal( inIdlstruct1)).equals( inIdlstruct2);
    }


    // Just the standard cloning for internal types
    public Object clone()
    {
        // Primitive types are cloned via assignment in the constructor.
        // Class types must be cloned via an appropiate method
        return new Idlstruct_Internal(  theShort,
                                        new String( theString) );
    }

    // Cloning to get an external type
    public Idlstruct cloneToIDLSuperClass()
    {
        // Primitive types are cloned via assignment in the constructor.
        // Class types must be cloned via an appropiate method
        return new Idlstruct(   theShort,
                                new String( theString) );
    }

    // Cloning of an external type
    public static Idlstruct cloneToIDLSuperClass( Idlstruct inObj)
    {
        return (new Idlstruct_Internal( inObj)).cloneToIDLSuperClass();
    }


    // Implementation of the method forced by the interface InternalIDLStruct
    public Class getIDLSuperClass()
        throws ClassNotFoundException
    {
        // This one makes clear which is our semantical parent class
        return Class.forName( "demo.Idlstruct");
    }


    // protect the members!
    protected   short   theShort;
    protected   String  theString;
}
```

4.1.4.5 Java-Quelltext „Server2Impl.java“

```
// Demonstrantion on the danger of dealing with objects
// in conjuncton with IDL structs.
//
// V 0.1 by Harald Roelle, December 1998
//

package demo;

class Server2Impl extends _Server2ImplBase
{
    private Idlstruct_Internal     internalStruct;

    public static void main( String[] args)
    {
        org.omg.CORBA.ORB orb       = org.omg.CORBA.ORB.init( args, new java.util.Properties());
        org.omg.CORBA.BOA boa       = orb.BOA_init( args, new java.util.Properties());

        Server2Impl       serverInstance = new Server2Impl();

        try {
            String              ref     = orb.object_to_string( serverInstance);
            String              refFile = "Server2.ref";
            java.io.PrintWriter out     = new java.io.PrintWriter(
                                                new java.io.FileOutputStream( refFile));
            out.println(ref);
            out.flush();
        }
        catch( java.io.IOException ex) {
            System.err.println( "Can't write to '" + ex.getMessage() + "'");
            System.exit( 1);
        }
```

```
        boa.impl_is_ready( null);
    }


    Server2Impl()
    {
        // Initialize the internal value of the struct.
        // These values are really internal and should be given to
        // clients for informational purposes only.
        // I.e. a client MUST NOT be able to change them.
        internalStruct = new Idlstruct_Internal( (short)42, "final solution");
    }

    public Idlstruct getValue()
    {
        return internalStruct.cloneToIDLSuperClass();
    }

    public boolean isEqualToInternal( Idlstruct inStruct)
    {
        return internalStruct.equals( inStruct);
    }


    public boolean isUnchanged()
    {
        if ( internalStruct.theShort != (short)42 )
            return false;
        else if ( ! internalStruct.theString.equals( "final solution") )
            return false;
        else
            return true;
    }
}
```

4.2 Vorschlag für die Implementierung und Nutzung von IDL-Struktur-Datentypen in Java

4.2.1 Motivation

Bei der Implementierung einer CORBA-Schnittstelle in Java wird bei der Übersetzung durch den „IDL nach Java“-Übersetzer für Struktur-Datentypen („`struct`“) folgendes Schema[3] benutzt. Eine IDL-Deklaration mit n Elementen in der Struktur

```
struct <name>
{
  <type_1>  <attrib_1>;
     :          :
  <type_n>  <attrib_n>;
}
```

Code-Fragment 4.12: IDL-Struktur-Datentyp

wird zu

[3] Teil des CORBA Standards, beispielsweise nachzuvollziehen in [5]

```
public final class <name>
{
  public <type_1>  <attrib_1>;
              :         :
  public <type_n>  <attrib_n>;

  public <name>()
  {
  }

  public <name>( <type_1> a_1, …, <type_n> a_<n>)
  {
    <attrib_1> = a_1;
         :         :
    <attrib_n> = a_<n>;
  }
}
```

Code-Fragment 4.13: Umsetzung der IDL-Definition in Java

Für eine der objektorientierten Programmierung angemessene Implementierung ist dieses Konstrukt aber nachteilig, da durch das Schlüsselwort `final` von diesen Klassen nicht mehr geerbt werden kann. Konkret bedeutet dies, daß einige in Java vorhandene Möglichkeiten oder Konventionen nicht umgesetzt werden können.

So ist es in Java üblich (und in der Java Basisklasse `Object` vorgeschlagen, siehe [2], Seite 466), daß Klassen eine eigene Methode `equals` implementieren, mit der auf Identität zweier Instanzen geprüft werden kann. Könnte man von der durch den „IDL nach Java"-Übersetzer erstellten Klasse erben, würde man schlicht eine „interne" Klasse ableiten, die die gewünschte Methode implementiert:

```
class <name>_Internal extends <name>
{
  public boolean equals( <name> obj)
  {
  …
  }
}
```

Code-Fragment 4.14: Kanonisches Vorgehen

Ebenso einfach könnten dann dort weitere Schnittstellen, z.B. für die automatische Serialisierung oder das Kopieren einer Instanz (Klonen), implementiert werden. Leider ist dies aber in der Form nicht möglich.

Während für den konkreten Fall der automatische Serialisierung viele „IDL nach Java"-Übersetzer Optionen anbieten, so daß bei der Übersetzung die entsprechenden Java Schnittstellen ergänzt werden, stellt dieses Vorgehen keine befriedigende Lösung für den allgemeinen Fall dar.

Zunächst sind nun zwei naive Ansätze denkbar, wie das Problem gelöst werden kann:

- Es wäre möglich, die gewünschten Ergänzungen direkt in der vom „IDL nach Java"-Übersetzer erzeugten Quelldatei vorzunehmen.
- Man könnte schlicht das `final`-Attribut aus der Klassendefinition streichen, um dann das oben angedeutete Schema mit Ableitung einer neuen Klasse einzusetzen.

Beide Methoden sind allerdings als kritisch anzusehen, da bei der Implementierung der CORBA-Laufzeitumgebung die Möglichkeit der Überladung sicherlich nicht berücksichtigt wurde, da das `final`-Attribut durch den Standard zugesichert wird. Daraus können Laufzeitfehler wie Typ-Verletzungen resultieren. Außerdem werden durch eine Ableitung hinzugekommene Attribute von der CORBA-Laufzeitumgebung nicht beachtet, da sie zum Zeitpunkt der Generierung der Hilfsklassen für die CORBA-Kommunikation nicht bekannt waren.

Ebenfalls beiden Methoden haftet der Nachteil an, daß bei einem weiteren Durchlauf des „IDL nach Java“-Übersetzers die gemachten Änderungen schlicht überschrieben werden. Eine nachträgliche Veränderung oder Ergänzung der IDL-Quellen oder der Einsatz eines anderen CORBA-Entwicklungssystems ist damit nicht mehr praktikabel.

Somit scheiden beide Ansätze aus.

4.2.2 Vorschlag für ein standardisiertes Vorgehen

Nachdem klar geworden ist, daß die Erzeugnisse des „IDL nach Java“-Übersetzers unantastbar sind, folgt daraus, daß solange CORBA/IDL keine anderen Übersetzungsmechanismen bietet, die nicht-Ableitbarkeit der erzeugten Klassen als gegebene Randbedingung angesehen werden muß.

Als Ausweg bleibt, eine interne Klasse zu deklarieren, die manuell eine Ableitung von Code-Fragment 4.13 herstellt, indem der vom „IDL nach Java“-Übersetzer erzeugte Code als Skelett für die Implementierung der neuen Klasse benutzt wird. Diese kann, wie gewünscht, nach Belieben um zusätzliche Schnittstellen, Attribute und Methoden angereichert werden.

Hierdurch geht nun aber innerhalb der Implementierung auf syntaktischer Ebene der zweifelsohne vorhandene semantische Zusammenhang der internen Klasse mit dem ursprünglichen Ergebnis aus dem IDL-Konstrukt verloren.

Dieses Schema teilt sich mit den vorher vorgestellten Ansätzen zwar den Nachteil, daß eine Änderung der IDL-Definition manuelle Änderungen an den internen Klassen nach sich zieht, zumindest wird aber das Überschreiben des Quelltexts durch den Übersetzer vermieden.

Als weitere Folge bleibt, neben der drastischen Verschlechterung der Wartbarkeit eines Projekts mit allen Folgen für die Qualitätssicherung, ein mögliches Aufblähen des Codes durch zusätzlich Methoden.

Beispielsweise muß bei der hier vorgeschlagenen Technik für eine Methode, die sowohl interne als auch die Originalklasse als Parameter Akzeptieren soll, zusätzlicher Aufwand getrieben werden. So muß für eine erfolgreich Überprüfung durch den Übersetzer für jede Klasse eine eigene Signatur implementiert werden:

```
public void dummyMethod( <name> inParam)
{
  return dummyMethod( new <name>_Internal( inParam));
}

public void dummyMethod( <name>_Internal inParam)
{
  …
}
```

Code-Fragment 4.15

Betrachten wir nun unseren Vorschlag für eine systematische Umsetzung in eine interne Klasse. Basierend auf der eingangs dargestellten IDL-Definition und der definierten Umsetzung ergibt sich folgender schematischer Aufbau:

```
1    class <name>_Internal implements ExtendedIDLStruct
2    {
3                     <type_1>  <attrib_1>;
4                        :          :
5                     <type_n>  <attrib_n>;
6
7      public         <name>_Internal()
8      public         <name>_Internal( <type_1> in<attrib_1>, …,
9                                      <type_n> in<attrib_n>)
10     public         <name>_Internal( <name> inObj)
11
12     public         <name>_Internal clone()
13     public         <name>          cloneToIDLSuperClass()
14     public static  <name>          cloneToIDLSuperClass( <name> inObj)
15
16     public         boolean         equals( <name>_Internal inObj)
17     public         boolean         equals( <name> inObj)
18     public static  boolean         equals( <name> inObj1, <name> inObj2)
19
20     public         Class           getIDLSuperClass()
21   }
```

Code-Fragment 4.16

Dieses Fragment sollte als Pflichtumfang für die interne Implementierung eines IDL-Struktur-Datentyps gelten, die dann nach Bedarf noch beliebig erweitert werden kann.

4.2.2.1 Attribute

Neben den Attributen wie sie in Code-Fragment 4.13 dargestellt sind, können hier nun weitere Attribute der Klasse stehen. Dabei ist auf die Verwendung angemessener Visibilitäts-Attribute (`private`, `public`, …) zu achten.

4.2.2.2 Konstruktoren

In Code-Fragment 4.16 sind die Zeilen 7 und 8/9 die direkten Pendants der Konstruktoren aus Code-Fragment 4.13, Zeilen 7 und 11. Ihre Implementierung ist vollkommen analog zu denen aus Code-Fragment 4.13:

```
public <name>_Internal()
{
  …
}
```

Code-Fragment 4.17: Standard-Konstruktor

```
public <name>_Internal( <type_1> in<attrib_1>, …,<type_n> in<attrib_n>)
{
  <attrib_1> = in<attrib_1>;
       :             :
  <attrib_n> = in<attrib_n>;
}
```

Code-Fragment 4.18: Attribut-Konstruktor

Für Code-Fragment 4.17 ist es denkbar und wünschenswert, daß alle Attribute mit sinnvollen Standardwerten initialisiert werden.

Weiterhin bietet es sich an, einen Konstruktor bereitzustellen, der es ermöglicht aus der Originalklasse eine interne Repräsentation zu gewinnen:

```
public <name>_Internal( <name> inObj)
{
  <attrib_1> = inObj.<attrib_1>;
       :             :
  <attrib_n> = inObj.<attrib_n>;
}
```

Code-Fragment 4.19: Superklassen-Konstruktor

Man beachte, daß alle Konstruktoren im Falle von Klassen mit Referenzen arbeiten. Es werden keine Instanzen geklont, nur die Referenzen übernommen!

4.2.2.3 Methoden

Um die in Kapitel 4.1 aufgezeigt Problematik behandeln zu können, empfiehlt es sich einige Methoden verpflichtend vorzuschreiben.

Zunächst seien hier die `equals`-Methoden für den Identitätsvergleich zweier Instanzen genannt.

Code-Fragment 4.16, Zeile 16, vergleicht auf Identität der Instanz gegen eine Instanz des gleichen Typs. Die Methode überlädt jene, die im Java Wurzelobjekt `java.lang.Object` eingeführt wird. Für die Implementierung einer solchen Methode sind vor allen Dingen die Anmerkungen aus Kapitel 4.1.3 zu beachten.

Code-Fragment 4.16, Zeile 15, ist analog, nur wird hier gegen eine Instanz der Originalklasse verglichen. Eine aus Geschwindigkeits- und Speichererwägungen ungünstige, aber nicht redundante, Implementierung könnte lauten:

```
public boolean equals( <name> inObj)
{
  return equals( new <name>_Internal( inObj));
}
```

Code-Fragment 4.20

Code-Fragment 4.16, Zeile 18 ist als statische Methode für den Identitäts-Vergleich zweier Instanzen von Originalklassen zuständig. Die Implementierung kann sich ebenfalls auf die "native" `equals`-Methode stützen:

```
public static boolean equals( <name> inObj1, <name> inObj2)
{
  return (new <name>_Internal( inObj1)).equals( inObj2);
}
```

Code-Fragment 4.21

Die grundlegende Semantik der `clone`-Methoden ist die, daß eine Kopie der Instanz als Rückgabewert an den Aufrufer gegeben wird, d.h. daß die Rückgabe by-value erfolgt. Bei der Implementierung der Methoden ist besonders darauf zu achten, daß bei Strukturelementen, die Referenzen auf Objektinstanzen sind, diese selbst wiederum durch ihre `clone`-Methoden dupliziert werden.

Zeile 12 in Code-Fragment 4.16 überlädt jene Methode, die im Java Wurzelobjekt `java.lang.Object` definiert wird.

Die Methode `cloneToIDLSuperClass()` (Code-Fragment 4.16, Zeile 13) stellt die Verbindung zur Superklasse her, indem eine Instanz vom Typ des Originalobjekts erzeugt wird.

Code-Fragment 4.16, Zeile 14 schließlich implementiert eine statische Methode zur Duplizierung eines Originalobjekts. Implementiert wird sie beispielsweise mit:

```
public static <name> cloneToIDLSuperClass( <name> inObj)
{
  return (new <name>_Internal( inObj)).cloneToIDLSuperClass();
}
```

Code-Fragment 4.22: statische `clone`-Methode

4.2.2.4 Schnittstelle „ExtendedIDLStruct”

Um den angesprochenen Nachteil des fehlenden syntaktischen Zusammenhangs auf Java-Ebene von <name> und <name>_Internal etwas zu mildern, sollte eine Java Schnittstelle implementiert werden, die hilft den Zusammenhang etwas deutlicher zu machen.

```
interface ExtendedIDLStruct
{
  public Class getIDLSuperClass()
    throws ClassNotFoundException;
}
```

Code-Fragment 4.23: Java-Schnittstelle „ExtendedIDLStruct”

Code-Fragment 4.23 erzwingt die Implementierung der Methode `getIDLSuperClass()`. Diese sollte dann als Rückgabewert die Originalklasse, die semantische Superklasse, liefern:

```
public Class getIDLSuperClass()
  throws ClassNotFoundException
{
  return Class.forName( "<name>");
}
```

Code-Fragment 4.24: Zugriff auf semantische Superklasse

Zwar bringt diese Konstruktion keine Vorteile auf der syntaktischen Ebene, wenigstens wird aber der semantischen Zusammenhang zwischen den Klassen auch im Quelltext hinterlegt.

Dadurch ist es beispielsweise zumindest möglich über den Klassenbaum zu erkennen, welche Klassen eines Projekts erweiterte Implementierungen eines IDL-Struktur-Datentyps darstellen. Weiterhin gestattet es ein Blick auf die konkrete Implementierung von `getIDLSuperClass()`,den semantischen Zusammenhang zur Originalklasse nachzuvollziehen.

4.2.3 Anmerkungen und Ausblicke

Eine Umsetzung des gemachten Vorschlags anhand eines Beispiels ist in Kapitel 4.1.4 angegeben.

Festzustellen bleibt, daß dieser Vorschlag nur eine Teil-Lösung auf Basis einer Konvention darstellt. Dem Projektmanagement und speziell der Qualitätssicherung eines Projekts obliegt die Durchsetzung dieser Konvention und damit die Erlangung der gewünschten Strukturierung mit all ihren Vorteilen.

Allenfalls durch Erstellung und Einsatz eines Co-Übersetzters zum „IDL nach Java"-Übersetzers, der der Konvention entsprechende Quelltext-Skelette erstellt, könnte eine weitere Automatisierungsstufe erreicht werden. Auf den ersten Blick scheint die Erstellung eines solchen Co-Übersetzers aus technischer Sicht nicht ausgeschlossen, Hinweise hierfür liefert dieser Vorschlag selbst. Ob allerdings, im Licht der fortschreitenden Entwicklung von CORBA/IDL betrachtet, der notwendige Aufwand für Entwicklung und Pflege gerechtfertigt ist, bleibt anzuzweifeln.

Eine durchwegs zufriedenstellende Lösung ließe sich deshalb nur erreichen, indem entsprechende Veränderungen und Erweiterungen der diversen Spezifikationen der CORBA/IDL Umsetzung nach Java durch die zuständigen Gremien vorgenommen würden.

5 Installation

Mit dem vorliegenden Installationspaket kann der TopologyService sowohl als allein lauffähige Variante (Option `STANDALONE`) als auch als Masa-Agent (Option `MASA_AGENT`) erzeugt werden.

Die unterstützten Plattformen sind Solaris und Linux. Die Variante `MASA_AGENT` kann allerdings nur für Solaris erzeugt werden. HP-UX wird derzeit nicht unterstützt, da Sun JDK hier nur in der Version 1.1.3 vorliegt, welche für Swing nicht ausreichend ist.

Voraussetzungen für die erfolgreiche Installation des Pakets sind:

	Solaris	Linux
Allgemein	GNU make[4] GNU find GNU awk GNU cpp (Teil des gcc Pakets)	
Java	Sun JDK 1.1.6 oder höher	
`STANDALONE`	Visibroker 3.0 oder ORBacus 3.0	ORBacus 3.0
`MASA_AGENT`	Visibroker 3.0	Nicht unterstützt
Applet	Swing 1.1beta2 oder höher	

Tabelle 5.1: Installationsvoraussetzungen

Details zur Installation finden sich in der `README` und `INSTALL`-Datei im Wurzelverzeichnis des Installationspakets.

[4] Die Solaris beiliegenden Fassungen sind nicht ausreichend.

6 Abschließende Bemerkungen

6.1 Eindruck von der Spezifikation

Betrachtet man rückblickend die Spezifikation [1], so bleibt festzustellen, daß diese in Teilen noch von sehr vorläufigem Charakter ist.

Neben trivialen Fehlern in den Schnittstellenbeschreibung finden sich auch inhaltliche Unzulänglichkeiten. Beispielsweise ist für die Methode `get_entity_of_type` für den Fall daß zwar der angefragte Typ existiert, aber keine *Entity* mit passendem Typ Teil der *AggregateEntity* ist, keine passende Ausnahmebehandlung spezifiziert.

Aber auch gewisse Eigenschaften der Spezifikation werden nicht so explizit dargestellt, wie es für eine zügige Implementierung wünschenswert wäre. So wird beispielsweise die Tatsache, daß *AggregateEntities* durch teilnehmende *Entities* identifiziert werden und daraus resultierend, daß eine *Entity* immer nur höchsten Teil einer *AggregateEntities* sein kann, an keiner Stelle explizit erwähnt.

Weiterhin schränken, nach Meinung des Autors, die starken Restriktionen bei der Bildung von *AggregateEntities* die Nutzbarkeit des TopologyService unnötig stark ein. In diesem Zusammenhang wäre eine ausführliche Beschreibung von Richtlinien und Schemata für die Modellierung anhand konkreter Beispiele sehr hilfreich.

Mittlerweile wurde die Weiterentwicklung des TopologyService von der OpenGroup übernommen.

6.2 Stand der Implementierung

Die vom Autor vorgelegte Implementierung ist in der vorliegenden Form nicht als vollständig und abgeschlossen zu bezeichnen. Nachfolgend ein Überblick über die noch zu ergänzenden Teile:

- Verwendung eines DBMS (Kapitel 3.1)
- Transaktionsmanagement, lokal und methodenübergreifend (Kapitel 3.3)
- Implementierung von Enforcerers (Kapitel 3.3)
- Implementierung von Notifications (Kapitel 3.3)
- Vervollständigung der Implementierung von TQL (Kapitel 3.3), dabei:
- Verwendung von Compiler-Tools zur Erstellung des Scanners/Parsers (Kapitel 3.5.3)
- Parallelisierung und Optimierung der TQL-Anfragebearbeitung (Kapitel 3.5.3)
- Verwendung des in Kapitel 4.2.2 vorgeschlagenen Schemas, hierfür möglicherweise:
- Erstellung eines Co-Übersetzers nach Kapitel 4.2.3

Daneben finden sich in der `RELEASE_NOTES`-Datei des Installationspakets noch weitere Hinweise auf bestehende Probleme der aktuellen Version.

7 Literatur

[1] Hewlett-Packard Company; „Response to Topology Service RFP“; Revised Submission; OMG TC Document Telecom/97-10-02; 16 October, 1997

[2] Flanagan, David: „Java in a Nutshell“: A Desktop Quick Reference; second Edition; O´Reilly & Associates; Mai 1997

[3] Newman, Alexander, u.a.: „Java: Referenz und Anwendungen“; Verlag que; Haar bei München; 1997

[4] Object-Oriented Concepts; „ORBacus For C++ and Java, Version 3.0“; 1998

[5] Visigenic; „Visibroker for Java 3.0: Reference Manual“; 1997

[6] Stallmann, Richard M. und McGrath, Roland; „GNU Make: A Program for Directing Recompilation“; Edition 0.48 for make Version 3.73 Beta; Free Software Foundation; April 1995

[7] Kempter, Bernhard: „Entwurf eines Java/CORBA-basierten Mobilen Agenten“; Diplomarbeit; Technische Universität München; 1998

[8] Sun Microsystems; „Java Platform 1.2 Beta 4 API Specification“; Palo Alto; 1998

[9] OMG; „The Common Object Request Broker: Architecture and Specification, Revision 2.0“; OMG Document 97–02–25

[10] OMG; „IDL/Java Language Mapping“; OMG document 97-03-01

[11] Gosling, James u.a.; „The Java Language Specification“; Edition 1.0; Addison-Wesley; August 1996

[12] Arnold, Ken und Gosling, James; „The Java Programming Language“; fourth printing; Addison-Wesly; 1996

[13] Sun Microsystems; „JDK 1.1.6 Specification“; Palo Alto; 1998

8 Anhang A: Verzeichnisbaum des Installationspakets

```
Makefile
Makefile.DEF.home
Makefile.DEF.mnm.linux.masa
Makefile.DEF.mnm.linux.standalone
Makefile.DEF.mnm.sunos.masa
INSTALL
README
RELEASE_NOTES
PRODUCTION.default/
    Makefile
    ControlServer.properties
    EntityAEServer.properties
    QueryServer.properties
    TypeServer.properties
    topo.db/
Testapp/
    Makefile
    AggregFunctions.java
    AssocFunctions.java
    EntityFunctions.java
    ExtendedFunctions.java
    QueryFunctions.java
    RuleFunctions.java
    TestApplication.java
    TypeFunctions.java
    Util.java
TopoApplet/
    Makefile
    MainPanel.java
    TopologyServiceApplet.java
    component/
        Makefile
        AssocPartField.java
        IndefProgressWatch.java
        PartIdentField.java
        ResultWindow.java
        RulePartField.java
        ScrollShowTablePane.java
        StringField.java
        TableHeadClickSortListener.java
        TypeJComboBox.java
    misc/
        Makefile
        Const.java
        Messages.java
        Util.java
    sorter/
        Makefile
        AssocsSorter.java
        EntitiesSorter.java
        QSortAbstract.java
        RulesSorter.java
        TiesSorter.java
        TypesSorter.java
    tabPanel/
        Makefile
        AggregPanel.java
        AssocPanel.java
        ControlPanel.java
        EntitiesPanel.java
        InfoPanel.java
        QueryPanel.java
        RulesPanel.java
        TypesPanel.java
    tableModel/
        Makefile
        AssocsTableModel.java
        EntitiesTableModel.java
        RulesTableModel.java
        SortTableModel.java
        TiesTableModel.java
        TypesTableModel.java
TopologyData/
    Makefile
    AggregCache.java
    AggregationTieInternal.java
    AssocPart.java
    AssociationInternal.java
    EntityStorage.java
    PartitionedIdentifierInternal.java
    TiePart.java
    TopologicalEntityInternal.java
    TopologyEntityAEServerStationaryAgent.java
TopologyMetaData/
    Makefile
    RuleStorage.java
    TopoTypeEntry.java
    TopologyTypeServerStationaryAgent.java
    TypeHierarchyInternal.java
    TypeStorage.java
TopologyPrivate/
    Makefile
    API.java
    API_Internal.java
    Constants.java
    PropertyNames.java
    Tools.java
    TopologyControlServerStationaryAgent.java
TopologyQuery/
    Makefile
    QCmd.java
    QExecStack.java
    QParser.java
    QToken.java
    QueryExecMachine.java
    QueryExecutionImpl.java
    TopologyQueryServerStationaryAgent.java
bin/
    add_serializable.linux
    buildAppletMASAHtml
    buildAppletStandaloneHtml
    getJavaStrConst
doc/
    Ausarbeitung.pdf
    Ausarbeitung.ps
    TopoService-RevSub971002.pdf
    Vortrag_Folien.pdf
    Vortrag_Folien.ps
idl/
    Makefile
    Topology.idl
    TopologyControlServer.idl
    TopologyData.idl
    TopologyEntityAEServer.idl
    TopologyMetaData.idl
    TopologyQuery.idl
    TopologyQueryServer.idl
    TopologyTypeServer.idl
```

9 Anhang B: Shell-Script „add_serializable.linux“

```
#!/bin/sh
#
# shell/awk script to add 'Serializable' interface to classes generated
# by the ORBacus idl2java compiler.
#
# This one is really a hack!
# Feel free to write a better and safer one!
#
# Author:  Harald Roelle
# Version: 0.1.0
#

JSOURCES=$(find $ROOT_PATH/$PROD_DIR/$SERVER_PACKPATH -name "*.java" -printf "%p ")
JIDLSOURCES=$(grep -l "// Generated by the ORBacus IDL to Java Translator" $JSOURCES)

for THE_FILE in $JIDLSOURCES; do
  mv $THE_FILE $THE_FILE.tmp
  gawk -- ' /^final public class/{ if ( index( $0, "implements") == 0 )
                                      printf "%s implements java.io.Serializable\n",$0;
                                    else
                                      printf "%s, java.io.Serializable\n",$0;
                                  }
           !/^final public class/{ print $0
                                  }' $THE_FILE.tmp > $THE_FILE
  rm $THE_FILE.tmp
done
```

10 Anhang C: IDL-Quelltexte nach Spezifikation der OMG

10.1 IDL-Quelltext Modul „Topology“

```
module Topology {

    struct PartitionedIdentifier {
        string                  id_context;
        string                  id_base;
    };

    typedef sequence<PartitionedIdentifier> IdList;

    struct TopologicalEntity {
        PartitionedIdentifier   id;
        Object                  obj;
    };

    typedef sequence <TopologicalEntity>    TopologicalEntityList;
    typedef string                          TopologicalTypeName;
    typedef sequence <TopologicalTypeName>  TopologicalTypeNameList;

    struct Association {
        TopologicalEntity       entity1;
        string                  role1;
        TopologicalEntity       entity2;
        string                  role2;
    };

    typedef sequence <Association>          AssociationList;

    struct NotificationValue {
        string                  kind;
        TopologicalTypeName     typo_name;
        PartitionedIdentifier   entity;
    };

    exception SchemaViolation {};
    exception AggregationConstraintViolation{};
    exception NotManaged{};
    exception ServiceNotAvailable{};
    exception TransactionWillRollback{};
    exception NoSuchType{};
    exception InconsistentData{};
    exception InvalidRoleName{};
    exception InvalidTypeName{};

};
```

10.2 IDL-Quelltext Modul „TopologyMetaData“

```
#include <Topology.idl>

module TopologyMetaData {

    interface TopologyEnforcer;

    struct TypeInheritance {
        unsigned long   parent;
        unsigned long   child;
    };

    typedef sequence <TypeInheritance> TypeInheritanceList;

    struct TypeHierarchy {
        Topology::TopologicalTypeNameList   types;
        TypeInheritanceList                 relationships;
    };

    enum InvokeCondition {
        ENTITY_ASSOCIATED,
        ENTITY_DISASSOCIATED,
        ENTITY_MANAGED,
        ENTITY_UNMANAGED,
        ENTITY_STATE_CHANGED,
        AGGREGATE_ENTITY_ASSOCIATED,
        AGGREGATE_ENTITY_DISASSOCIATED,
        AGGREGATE_ENTITY_STATE_CHANGED,
        ASSOCIATED_ENTITY_ASSOCIATED,
        ASSOCIATED_ENTITY_DISASSOCIATED,
        ASSOCIATED_ENTITY_STATE_CHANGED,
        ALL
    };

    enum EnforcerType {
        COMMON_ENFORCER,
        NAMED_ENFORCER
    };

    typedef sequence<InvokeCondition>       InvokeConditionList;
    typedef string                          EnforcerName;
```

```
struct EnforcerRegistration {
    InvokeConditionList                 invoke_conditions;

    union EnforcerObjectReference switch(EnforcerType) {

        case NAMED_ENFORCER:
            EnforcerName enforcer_name;

        case COMMON_ENFORCER:
        default:
            TopologyEnforcer enforcer_obj;
    }                                   enforcer;
};

typedef sequence <EnforcerRegistration> EnforcerRegistrationList;
const unsigned long NO_LIMIT = ~0;

struct AssociationRule {
    string          type1;
    string          role1;
    unsigned long   min_cardinality1;
    unsigned long   max_cardinality1;
    string          type2;
    string          role2;
    unsigned long   min_cardinality2;
    unsigned long   max_cardinality2;
};

typedef sequence<AssociationRule> AssociationRuleList;

exception TypeRedefinition{};
exception InvalidCardinality{};
exception ConflictingRule{};
exception NonDistinctRoles{};
exception TypeInUse{};
exception NoSuchEnforcerImplementation{};
exception NonTransactionalOperation{};

interface TopologicalTypeManager {

    boolean create (
            in Topology::TopologicalTypeName        type_name,
            in Topology::TopologicalTypeNameList    parent_types)
        raises (
            Topology::InvalidTypeName,
            Topology::NoSuchType,
            TopologyMetaData::TypeRedefinition,
            Topology::TransactionWillRollback,
            Topology::ServiceNotAvailable);

    boolean dismiss (
            in Topology::TopologicalTypeName        type_name)
        raises (
            Topology::InvalidTypeName,
            TopologyMetaData::NonTransactionalOperation,
            Topology::ServiceNotAvailable);

    boolean exists (
            in Topology::TopologicalTypeName        type_name)
        raises (
            Topology::InvalidTypeName,
            Topology::ServiceNotAvailable);

    Topology::TopologicalTypeNameList get_all_types()
        raises (
            Topology::ServiceNotAvailable);

    TypeHierarchy get_type_hierarchy(
            in Topology::TopologicalTypeName        type_name)
        raises (
            Topology::InvalidTypeName,
            Topology::NoSuchType,
            Topology::ServiceNotAvailable);

    Topology::TopologicalTypeNameList get_parents (
            in Topology::TopologicalTypeName        type_name)
        raises (
            Topology::InvalidTypeName,
            Topology::NoSuchType,
            Topology::ServiceNotAvailable);

    AssociationRuleList get_association_rules (
            in Topology::TopologicalTypeName        type_name)
        raises (
            Topology::InvalidTypeName,
            Topology::NoSuchType,
            Topology::ServiceNotAvailable);

    EnforcerRegistrationList get_enforcers (
            in Topology::TopologicalTypeName        type_name)
        raises (
            Topology::InvalidTypeName,
            Topology::NoSuchType,
            Topology::ServiceNotAvailable);

    boolean add_rule(
            in AssociationRule                      rule)
        raises (
            Topology::InvalidTypeName,
            Topology::InvalidRoleName,
            Topology::NoSuchType,
```

```
                InvalidCardinality,
                ConflictingRule,
                NonDistinctRoles,
                TypeInUse,
                Topology::TransactionWillRollback,
                Topology::ServiceNotAvailable);

        boolean remove_rule(
                in AssociationRule                      rule)
            raises (
                Topology::InvalidTypeName,
                Topology::InvalidRoleName,
                Topology::NoSuchType,
                TypeInUse,
                Topology::TransactionWillRollback,
                Topology::ServiceNotAvailable);

        void define_enforcers(
                in Topology::TopologicalTypeName        type_name,
                in EnforcerRegistrationList             enforcers)
            raises (
                Topology::InvalidTypeName,
                Topology::NoSuchType,
                Topology::TransactionWillRollback,
                TypeInUse,
                NoSuchEnforcerImplementation,
                Topology::ServiceNotAvailable);
    };

    interface TopologyEnforcer {

        boolean validate (
            in Topology::TopologicalEntity affected_entity);

    };
};
```

10.3 IDL-Quelltext Modul „TopologyData“

```
#include <Topology.idl>

module TopologyData {

    struct AggregationTie {
        Topology::TopologicalEntity entity1;
        Topology::TopologicalEntity entity2;
        string                      id;
    };
    typedef sequence <AggregationTie>   AggregationTieList;
    exception TypeChanged {};

    interface EntityManager {
        // Operations which deal with a collection of entities
        boolean manage (
                in Topology::TopologicalEntity      entity,
                in Topology::TopologicalTypeName    topo_type)
            raises (
                Topology::SchemaViolation,
                Topology::InvalidTypeName,
                Topology::NoSuchType,
                Topology::TransactionWillRollback,
                Topology::ServiceNotAvailable,
                TypeChanged);

        boolean unmanage (
                in Topology::PartitionedIdentifier  entity)
            raises (
                Topology::SchemaViolation,
                Topology::ServiceNotAvailable,
                Topology::TransactionWillRollback);

        boolean associate (
                in Topology::Association            assoc)
            raises (
                Topology::InconsistentData,
                Topology::SchemaViolation,
                Topology::InvalidRoleName,
                Topology::NotManaged,
            Topology::TransactionWillRollback,
            Topology::ServiceNotAvailable);

        boolean disassociate (
                in Topology::Association            assoc)
            raises (
                Topology::InconsistentData,
                Topology::SchemaViolation,
                Topology::TransactionWillRollback,
                Topology::NotManaged,
                Topology::InvalidRoleName,
                Topology::ServiceNotAvailable);

        boolean tie_entities (
                in AggregationTie                   tie)
            raises (
                Topology::NotManaged,
                Topology::AggregationConstraintViolation,
                Topology::TransactionWillRollback,
                Topology::ServiceNotAvailable);
```

```
        boolean untie_entities (
                in AggregationTie                   tie)
            raises (
                Topology::NotManaged,
                Topology::TransactionWillRollback,
                Topology::ServiceNotAvailable);

        boolean in_same_aggregate (
                in Topology::IdList                 entity_list)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);

        // Operations for which the EntityManager acts as a proxy
        // for an individual entity.
        boolean is_managed (
                in Topology::PartitionedIdentifier  entity)
            raises (
                Topology::ServiceNotAvailable);

        void state_has_changed (
                in Topology::PartitionedIdentifier  entity)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);

        Topology::AssociationList get_associations (
                in Topology::PartitionedIdentifier  entity)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);

        Topology::TopologicalTypeName get_topological_type (
                in Topology::PartitionedIdentifier  entity)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);

        AggregationTieList get_ties (
                in Topology::PartitionedIdentifier  entity)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);
    };

    interface AEManager {

        // Operations for which the EntityManager acts as a proxy
        // for an individual aggregate-entity
        Topology::TopologicalEntityList get_entities (
                in Topology::PartitionedIdentifier  ae)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);

        Topology::TopologicalTypeNameList get_topological_types (
                in Topology::PartitionedIdentifier  ae)
            raises (
                Topology::ServiceNotAvailable,
                Topology::NotManaged);

        Topology::TopologicalEntity get_entity_of_type (
                in Topology::PartitionedIdentifier  ae,
                in Topology::TopologicalTypeName    topo_type)
            raises (
                Topology::ServiceNotAvailable,
                Topology::InvalidTypeName,
                Topology::NoSuchType,
                Topology::NotManaged);
    };
};
```

10.4 IDL-Quelltext Modul „TopologyQuery“

```
#include <Topology.idl>

module TopologyQuery {

    typedef sequence<string>        StringList;
    typedef sequence<unsigned long> IndexList;

    struct QueryExecutionStatus {
        boolean     truth_of_proposition;
        boolean     partial;
        boolean     finished;
        short       num_results;
    };

    struct QueryResultAssociation {
        unsigned long   pred_aggregate;
        unsigned long   pred_entity;
        unsigned long   pred_role;
        unsigned long   succ_aggregate;
        unsigned long   succ_entity;
        unsigned long   succ_role;
    };
    typedef sequence <QueryResultAssociation> QueryResultAssociationList;
```

```
    struct QueryResultEntity {
        Topology::TopologicalEntity     entity;
        unsigned long                   aggregate;
        unsigned long                   type;
    };
    typedef sequence <QueryResultEntity> QueryResultEntityList;

    struct QueryResultAggregate {
        IndexList   entities;
        IndexList   tags;
    };
    typedef sequence <QueryResultAggregate> QueryResultAggregateList;

    struct QueryResult {
        QueryResultEntityList       entities;
        QueryResultAggregateList    aggregates;
        QueryResultAssociationList  associations;
        StringList                  types;
        StringList                  roles;
        StringList                  tags;
    };
    typedef sequence <QueryResult> QueryResults;

    enum QueryResultFormat {
        COMBINED_RESULT,
        SEPARATE_RESULTS
    };

    enum QueryResultOption {
        QRO_ALL,
        QRO_ANY,
        QRO_TRUTH
    };

    struct ParameterBinding {
        string      parameter_name;
        string      parameter_value;
    };
    typedef sequence <ParameterBinding> ParameterBindings;

    exception CompileError{
        string  reason;
    };

    interface QueryExecution {
        void stop ( )
            raises (
                Topology::ServiceNotAvailable);

        QueryExecutionStatus get_status ( )
            raises (
                Topology::ServiceNotAvailable);

        string get_query()
            raises(
                Topology::ServiceNotAvailable);

        void get_next (
                out QueryExecutionStatus    status,
                out QueryResult             result_data)
            raises (
                Topology::ServiceNotAvailable);

        void get_all (
                in QueryResultFormat        result_format,
                out QueryExecutionStatus    status,
                out QueryResults            result_data)
            raises (
                Topology::ServiceNotAvailable);

        };

    interface QueryExecutionFactory {

        QueryExecution create (
                in string               tql_string,
                in ParameterBindings    parameters,
                in QueryResultOption    result_option)
            raises (
                CompileError,
                Topology::ServiceNotAvailable);
    };
};
```

11 Anhang D: IDL-Quelltexte der CORBA-Server

11.1 IDL-Quelltext „TopologyTypeServer.idl“

```
#ifndef __TopologyTypeServer__
#define __TopologyTypeServer__


#include <TopologyMetaData.idl>

#ifdef MASA_AGENT
#include "Migration.idl"
#endif


module TopologyMetaData {

    interface TopologicalTypeManagerInternal {

        Topology::TopologicalTypeNameList get_parents_hierarchy(
                in Topology::TopologicalTypeName        type_name)
            raises (
                Topology::InvalidTypeName,
                Topology::NoSuchType,
                Topology::ServiceNotAvailable);

        void addInstanceToType(
                in Topology::TopologicalTypeName        inType,
                in Topology::PartitionedIdentifier      inID)
            raises (
                Topology::TransactionWillRollback);

        void removeInstanceFromType(
                in Topology::TopologicalTypeName        inType,
                in Topology::PartitionedIdentifier      inID)
            raises (
                Topology::TransactionWillRollback);

        void storeToDisk(
                in string                               inName )
            raises (
                Topology::ServiceNotAvailable);

        void loadFromDisk(
                in string                               inName )
            raises (
                Topology::ServiceNotAvailable);

        void shutdownServer();
    };

#ifdef MASA_AGENT
    interface TopologyTypeServer :  TopologyMetaData::TopologicalTypeManager,
                                    TopologicalTypeManagerInternal,
                                    agent::Migration {
#else
    interface TopologyTypeServer :  TopologyMetaData::TopologicalTypeManager,
                                    TopologicalTypeManagerInternal {
#endif
    };
};
#endif
```

11.2 IDL-Quelltext „TopologyEntityAEServer.idl“

```
#ifndef __TopologyEntityAEServer__
#define __TopologyEntityAEServer__


#include <TopologyData.idl>

#ifdef MASA_AGENT
#include "Migration.idl"
#endif


module TopologyData {

    interface EntityManagerInternal {

        void storeToDisk(
                in string                           inName )
            raises (
                 Topology::ServiceNotAvailable);

        void loadFromDisk(
                in string                           inName )
            raises (
                 Topology::ServiceNotAvailable);

        void shutdownServer();
```

```
    };

    interface AEManagerInternal {

        Topology::TopologicalEntityList get_all_ae()
            raises (
                Topology::ServiceNotAvailable);
    };

#ifdef MASA_AGENT
    interface TopologyEntityAEServer :  TopologyData::EntityManager, EntityManagerInternal,
                                        TopologyData::AEManager, AEManagerInternal,
                                        agent::Migration {
#else
    interface TopologyEntityAEServer :  TopologyData::EntityManager, EntityManagerInternal,
                                        TopologyData::AEManager, AEManagerInternal {
#endif
    };
};
#endif
```

11.3 IDL-Quelltext „TopologyQueryServer.idl“

```
#ifndef __TopologyQueryServer__
#define __TopologyQueryServer__


#include <TopologyQuery.idl>

#ifdef MASA_AGENT
#include "Migration.idl"
#endif


module TopologyQuery {

    interface QueryExecutionFactoryInternal {
        void shutdownServer();
    };

#ifdef MASA_AGENT
    interface TopologyQueryServer : TopologyQuery::QueryExecutionFactory,
                                    QueryExecutionFactoryInternal,
                                    agent::Migration {
#else
    interface TopologyQueryServer : TopologyQuery::QueryExecutionFactory,
                                    QueryExecutionFactoryInternal {
#endif
    };

};
#endif
```

11.4 IDL-Quelltext „TopologyControlServer.idl“

```
#ifndef __TopologyControlServer__
#define __TopologyControlServer__


#include <Topology.idl>
#include <TopologyData.idl>
#include <TopologyMetaData.idl>
#include <TopologyQuery.idl>

#ifdef MASA_AGENT
#include "Migration.idl"
#endif


module TopologyPrivate {

    interface TopologyControl {

        TopologyData::EntityManager              GetEntityManager();
        TopologyData::AEManager                  GetAEManager();
        TopologyMetaData::TopologicalTypeManager GetTopologicalTypeManager();
        TopologyQuery::QueryExecutionFactory     GetQueryExecutionFactory();

        void storeToDisk (
                            in string inDBName)
            raises (
                Topology::ServiceNotAvailable);

        void loadFromDisk (
                            in string inDBName)
            raises (
                Topology::ServiceNotAvailable);

        void shutdownService();
    };

#ifdef MASA_AGENT
    interface TopologyControlServer :   TopologyControl,
                                        agent::Migration {
#else
```

```
    interface TopologyControlServer : TopologyControl {
#endif
    };
};
#endif
```