INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Systementwicklungsprojekt

Interoperabilität von CORBA-ORBs in Mobilen Agentensystemen

Bearbeiter:	Raimund Brandt
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Boris Gruschke Helmut Reiser
Abgabetermin:	April 1999

Zusammenfassung

Während CORBA 1.1 die Implementierung von Programmen für CORBA-compliant ORBs unterschiedlicher Hersteller garantieren sollte, hatte sich die *Object Managemant Group (OMG)* mit CORBA 2.0 ein weit höheres Ziel gesteckt: Die Interoperabilität zwischen ORBs verschiedener Hersteller. Die praktische Realisierbarkeit dieses gesteckten Zieles sollte im Einsatz verschiedener CORBA-ORBs mit der hier am Lehrstuhl entwickelten *Mobile Agent System Architecture (MASA)* [Kemp 98] untersucht werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Interoperabilitäts Schnittstellen

1.1 Einleitung

Während CORBA 1.1 die Implementierung von Programmen für CORBA-compliant ORBs unterschiedlicher Hersteller garantieren sollte, hatte sich die *Object Managemant Group (OMG)* mit CORBA 2.0 ein weit höheres Ziel gesteckt: Die Interoperabilität zwischen ORBs verschiedener Hersteller. Die praktische Realisierbarkeit dieses gesteckten Zieles sollte im Einsatz verschiedener CORBA-ORBs mit der hier am Lehrstuhl entwickelten *Mobile Agent System Architecture (MASA)* [Kemp 98] untersucht werden.

1.2 Aufgabenstellung

Es sollte aber nicht nur die in CORBA 2.0 spezifizierte Interoperabilität zwischen verschiedenen CORBA-ORBs sondern auch eine eventuelle Kommunikation zwischen verschiedenen Mobilen Agentensystem Architekturen geprüft werden, so daß an den in Abbildung 1.1 gezeigten Schnittstellen eine mögliche Interoperabilität von Interesse war:

- an der Schnittstelle zwischen einer Agentensystemarchitektur und der Kommunikationsarchitektur CORBA, die Agentensystem-ORB-Schnittstelle
- zwischen zwei Agentensystemarchitekturen, die sogenannte Agentensystemschnittstelle
- zwischen verschiedenen CORBA ORBs, die Inter-ORB-Schnittstelle

Für die Interoperabilitätstests an der Agentensystem-ORB-Schnittstelle wurde als Agentensystemarchitektur, die eingangs erwähnte *Mobile Agent System Architecture (MASA)* [Kemp 98] verwendet. Als CORBA ORBs wurden VisiGenics VisiBroker 3.0 und das in JavaSofts JDK 1.2 enthaltene Java IDL benutzt. Die Auswahl fiel aus unterschiedlichen Gründen auf diese Produkte: VisiBroker 3.0 wird im bestehenden MASA als Kommunikationsarchitektur eingesetzt, JavaSofts Java IDL ist im Paket mit JDK 1.2 als freier CORBA ORB erhältlich und somit als Alternative zum kommerziellen VisiBroker interessant. Die gerade erwähnten CORBA ORBs wurden ebenfalls zum Testen der Interoperabilität zwischen verschieden ORBs eingesetzt. Als alternative Agentensystemarchitektur wurde das kommerzielle Agentenframework Voyager 2.0 der Firma Objectspace verwandt, an dessen IIOP-Schnittstelle eine mögliche Kommunikation mit VisiBroker und JavaSofts Java IDL zu überprüfen war. An den beschriebenen Schnittstellen wurde die Interoperabilität zunächst durch kleine Programme im "Hello World"-Stil getestet, bevor das Verhalten der Komponenten in umfangreicheren Systemen untersucht wurde. Dadurch konnten mögliche Fehlerquellen eingegrenzt werden.

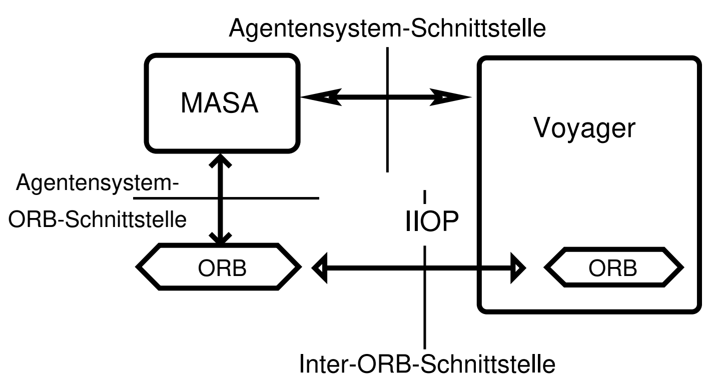

Abbildung 1.1: Interoperabilitäts Schnittstellen

Kapitel 2

MASA-ORB Schnittstelle

Eine dieser komplexeren Konfigurationen stellt die Verwendung verschiedener CORBA-ORBs mit MASA dar. An der aus dieser Konfiguration resultierenden Agentensystem-ORB Schnittstelle sollte ein Austausch des in MASA verwendeten VisiBroker 3.0 gegen JavaSofts Java IDL getestet werden, wobei JavaSofts Java IDL nur als Beispiel für einen alternativen CORBA ORB diente. Andere CORBA ORBs, die durch ihren erweiterten Leistungsumfang, einen Austausch rechtfertigen würden, wären zum Beispiel: OrbixWeb 3.0 von Iona [OrHa 98], dessen Property Service derzeit im Rahmen eines Systementwicklungsprojekts von Thomas Waibel zur Realisierung einer dynamischen Konfiguration in MASA benutzt wird, ORBacus (Object Oriented Concepts, Inc.), dessen IIOP über SSL für die laufenden Diplomarbeit von Harald Roelle [Röl 99] von Interesse ist, oder auch der JacORB 0.9f von G. Brose (Freie Universität Berlin) [Bros], dessen Naming Service eine graphische Benutzeroberfläche beinhaltet.

2.1 Object Adapter

Ein Problembereich bei der Interoperabilität zwischen Agentensystem und verschiedenen CORBA ORBs ist der Object Adapter. Der Object Adapter ist eine Komponente auf Serverseite zwischen ORB und Agentensystem. Seine Aufgabe besteht in der Aktivierung und Deaktivierung der Objekte auf dem Server. Dadurch wird die Transparenz bezüglich des Aktivierungszustandes eines Objekts gegenüber dem Client realisiert und die Anforderungen an die Ressourcen des Servers reduziert, indem von Clients nicht referenzierte Objekte deaktiviert werden.

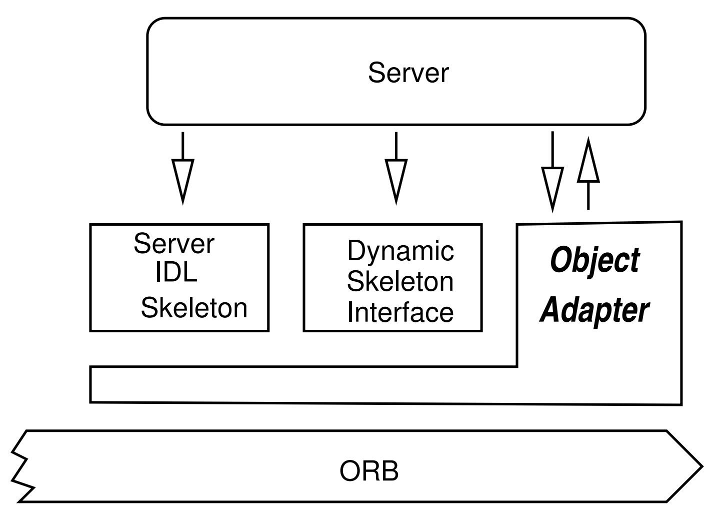

Abbildung 2.1: Object Adapter

2.1.1 Basic Object Adapter

In CORBA 2.0 wurde als Implementierung des Object Adapters der Basic Object Adapter (BOA) spezifiziert. Da dieser original BOA unterspezifiziert war([OrHa 98], s.S.289), führten viele Hersteller proprietäre Erweiterungen ein, um die Schwachstellen des BOA zu beheben. Dies führte in folgenden Bereichen zu Inkonsistenzen auf Serverseite zwischen CORBA ORBs verschiedener Hersteller:

- Objekt Aktivierung/Deaktivierung
- Namensgebung der Basis-Skeleton-Klassen
- Registrierung der Objekte am ORB

2.1.2 Portable Object Adapter

In CORBA 3.0 wird dieses Problem durch den Portable Object Adapter (POA) gelöst[OMG b]. Der POA besitzt die gleiche Funktionalität wie der BOA, enthält aber auch neue Konzepte, wie zum Beispiel den Servant Manager, der als Callback-Objekt benutzt werden kann, um dem Client die Aktivierung und Deaktivierung von Servant-Objekten zu ermöglichen. Der POA wird den BOA zwar ablösen, doch soll der BOA eigentlich für einige Jahre zusammen mit dem POA unterstützt werden, so daß bestehenden Programme, die den BOA benutzen, nicht modifiziert werden müssen. JavaSofts Java IDL unterstützt jedoch nicht mehr den BOA, sondern nur noch den POA, wobei auch dessen Implementation eine Minimallösung darstellt. Diese Minimallösung besteht nur aus zwei Methoden (`connect()`) zur Registrierung bzw. (`disconnect()`) zur Deaktivierung der Objekte am ORB. Diese Implementierung des POA bietet auch VisiBroker 3.0, der aber nach CORBA 2.0 als Object Adapter im vollem Umfang den BOA anbietet.

2.1.3 Object Adapter in MASA

Im Zusammenhang mit MASA und dem Austausch des VisiBroker 3.0 durch JavaSofts Java IDL wird die Ersetzung des BOAs durch den POA zum Problem. Bei der Entwicklung von MASA war die Verwendung des VisiBroker 3.0 mit dem Einsatz des BOA zwangsläufig verbunden. Eine Selektormethode (`getBOA()`) an der MASA-API bindet den BOA als festen Bestandteil in MASA ein und es muß davon ausgegangen werden, daß diese Methode und damit auch der BOA von vielen für MASA implementierten Agenten an der MASA-API vorausgesetzt wird.

2.1.4 Lösungsansatz

Bei einem Austausch des von MASA verwendeten VisiBroker ORBs durch JavaSofts Java IDL müßte die MASA-API angepaßt werden, indem die Methode getBOA() ersatzlos gestrichen würde und alle bestehenden Agenten, die `getBOA()` verwenden, modifiziert würden. Bei einem Austausch durch einen CORBA 3.0 compliant ORB, der den POA in vollem Umfang realisiert, könnte der Modifizierungsaufwand reduziert werden, da die Methode `getBOA()` durch eine äquivalente Methode `getPOA()` ersetzt werden könnte und so die Verwendung des Object Adapters als Pseudo-Objekt in bestehenden Agenten möglich wäre.

2.2 Tie delegation

Ein weiteres Problem beim Austausch des CORBA ORBs in MASA entsteht im Zusammenhang mit *tie delegation*, das zunächst kurz beschrieben werden soll. Bei *tie delegation* handelt es sich um IDL-to-Java Mapping auf Serverseite, d.h. die Abbildung der IDL-Schnittstellenbeschreibung eines Objekts in seine Java-Klassen auf Serverseite, die Skeletons. Das Standardverfahren wird als *base inheritance* bezeichnet. Dabei wird die Klasse, die vom Programmierer als Implementation eines Objekts realisiert wird, als Unterklasse der vom IDL-to-Java-Compiler generierten Klasse `_interface_nameImplBase`. Deshalb auch die Bezeichnung *base inheritance*. *tie delegation* ist eine Alternative zur *base inheritance* ([OrHa 98],s.S.460), ermöglicht die Abbildung von Mehrfachvererbung in IDL nach Java. Dies wird in Abbildung 2.2 als Beispiel der drei IDL Interfaces `A`, `B`, `C` und der Implementierung von `C` in Java dargestellt. Das Interface `C` erbt von `A` und `B` deren Attribute und Methoden und deklariert selbst ein eigenes Attribut und eine Methode. Vom IDL-to-Java Compiler werden die Klassen `_CImplBase` und `_tie_C`, sowie die Interfaces `C` und `COperations` in Java erzeugt. Die Klasse `_CImplBase` implementiert das Interface `C`, dessen Methoden in `_CImplBase` nicht ausprogrammiert sind. Bei *base inheritance* fungiert `_CImplBase` als abstrakte Oberklasse, für die Klasse, in der das CORBA Objekt implementiert wird. Bei *tie delegation* dient die abstrakte Klasse `_CImplBase` nur als Oberklasse der Klasse `_tie_C`, in der die vereinte Menge aller Attribute und Methoden von `A`, `B` und `C` implementiert ist. `_tie_C` besitzt ein ein Attribut `_delegate`, das durch das Interface `COperations` spezifiziert ist und an das alle Aufrufe an Attribute und Methoden von `A`, `B` und `C` weitergeleitet werden. Dieses Interface `COperations` muß von einer Klasse implementiert werden um das CORBA Objekt `C` zu realisieren. Durch die Indirektheit,

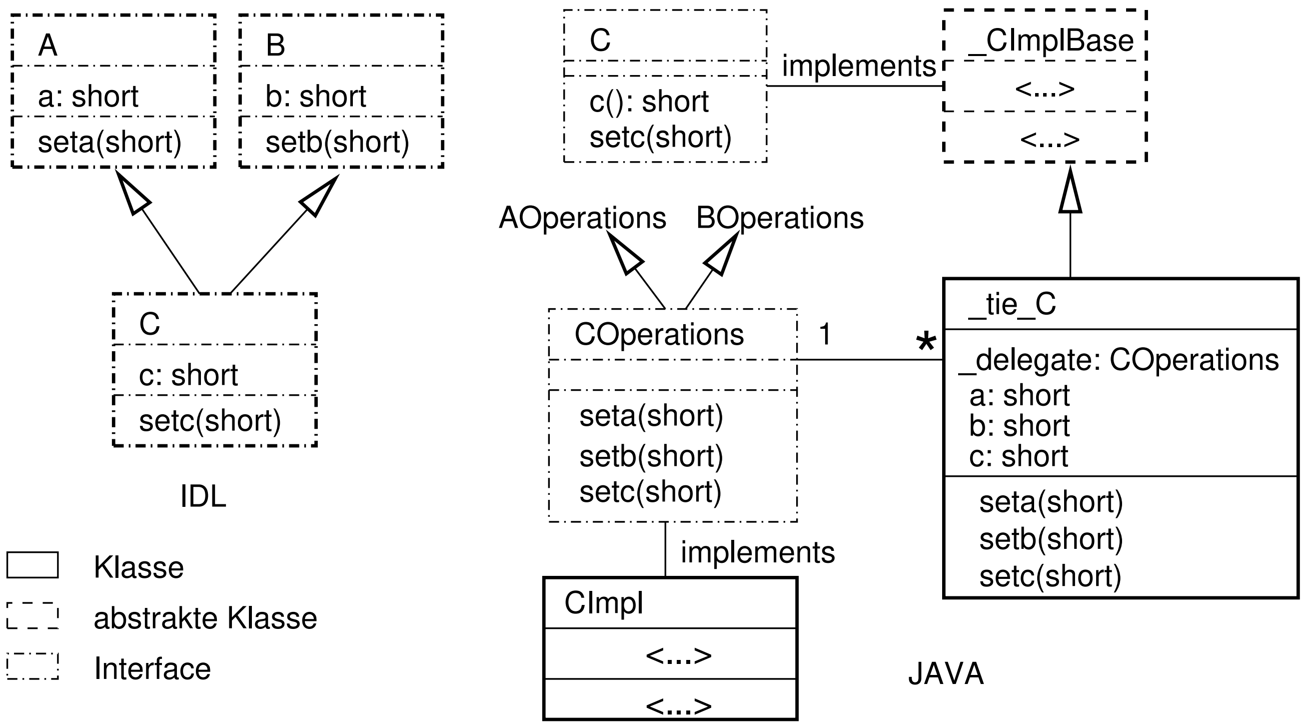

Abbildung 2.2: Beispiele für *tie delegation*

auf der *tie delegation* basiert, ist dieser Mechnismus für den Programmierer weniger intuitiv als das Standardverfahren mit *base inheritance* zur Implementierung von IDL-Interfaces und erfordert Performanceeinbußen auf Serverseite, so daß *tie delegation* nur eingestzt werden sollte, wenn es wirklich nötig ist, d.h. Mehrfachvererbung abgebildet werden muß oder Indirektheit erwünscht ist.

2.2.1 Verwendung und Probleme in MASA

Die Klassenhierarchie ist in MASA durch die MASIF-Spezifikation [Kemp 98] vorgegeben. Darin ist keine Mehrfachvererbung beinhaltet und erfordert eigentlich nicht den Einsatz von *tie delegation*. Die Implementierung von MASA erfordert allerdings eigene Zusatzklassen, so daß ein IDL-Szenario mit Mehrfachvererbung entsteht und *tie delegation* in MASA eingesetzt werden muß. Ein Austausch von VisiBroker 3.0 durch JavaSofts Java IDL ist im Zusammenhang mit *tie delegation* problematisch, da JavaSofts Java IDL sich nicht an der bei der OMG eingereichten Submission [OMG b] bezüglich der

Namenskonvention der Tie- bzw. Operation-Klassen orientiert. Abbildung 2.3 zeigt die unterschiedlichen Benennungen bei VisiBroker 3.0 und JavaSofts Java IDL.

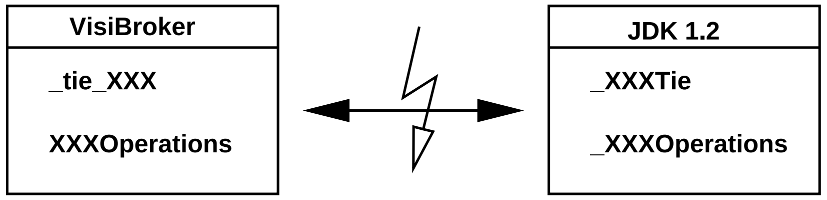

Abbildung 2.3: *tie delegation* Namensunterschiede

2.2.2 Lösungsansatz

Um in MASA einen Austausch VisiBroker 3.0 und JavaSofts Java IDL zu ermöglichen, müßte eine beiden Namenskonventionen genügende Parametriesierung das MASA-Quellcodes erfolgen, der vor der jeweiligen Compilierung durch eine Präprozessorbearbeitung auf die entsprechende Konvention vorbereitet werden müßte. Dieser recht umfangreiche Lösungsansatz soll hier nicht weiter ausgeführt werden, da das beschriebene Verfahren nur bei einem Austausch durch JavaSofts Java IDL nötig wäre. Bei Verwendung eines CORBA ORB (z.B. OrbixWeb), dessen IDL-to-Java Compiler sich an der Namenskonvention gemäß VisiBroker orientiert, ist die Problematik nicht gegeben.

Kapitel 3

ORB-ORB Schnittstelle

Die in den vorhergehenden Kapiteln beschriebenen Probleme und deren Lösungsansätze hätten einen vollkommenen Wechsel des CORBA-ORBs, der Kommunikationsschicht in MASA, bedeutet, was jedoch angesichts der Komplexität vorerst zurückgestellt wurde und somit die Lösungsansätze nicht realisert wurden. Anders verhält es sich an der ORB-ORB-Schnittstelle. Hier konnte ein Konzept ohne größere Modifikation des bestehenden Systems realisiert werden. Die Kommunikation zwischen CORBA-Client und CORBA-Server über ORB-Grenzen hinweg ist durch das Internet Inter-ORB Protokoll (IIOP) ausreichend spezifiziert und relativ unproblematisch. Die Interoperabilität von CORBA-Services ist bisher nicht ausreichend spezifiziert.

3.1 Naming Service

Ein elementarer CORBA-Service ist der Naming Service, der die Auflösung von CORBA-Objektreferenzen über Bezeichner ermöglicht. Er besteht aus zwei Objekten: NamingContext und BindingIterator(s. Abb. 3.1). Im NamingContext können Bezeichnern CORBA-Objektreferenzen zugeord-

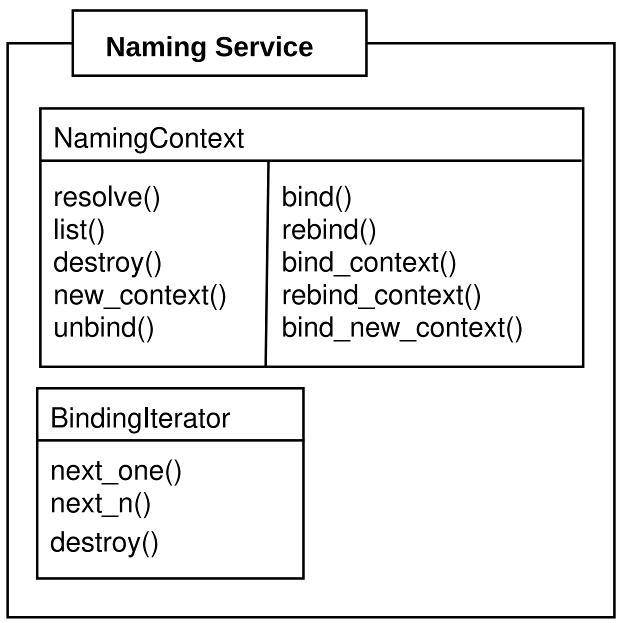

Abbildung 3.1: Komponenten des Naming Service

net werden. Der BindingIterator dient gemäß seinem Namen als Iterator, um die Liste aller Bindungen, Paare von Bezeichnern und Referenzen, im NamingContext zu durchlaufen([OrHa 98], s.S.115). Um über den NamingContext eine Verbindung zum Naming Service eines CORBA-ORBs herzustellen, stellt der ORB die Funktion `resolve_initial_references()` zur Verfügung, mit der die Objekte der Implementierungen aller aktiven CORBA-Services, wie zum Beispiel der NamingContext des Naming Services, referenziert werden können. Dieser Mechanismus funktioniert allerdings nicht zwischen ORBs unterschiedlicher Hersteller.

3.1.1 Konzept zur Interoperabilität

Aus diesem Grund muß ein Konzept entwickelt werden, damit ein Zugriff auf einen Naming Service über ORB-Grenzen hinweg ermöglicht wird. Beim Einsatz verschiedener CORBA-ORBs mit jeweils eigenen Naming Services, in diesem Fall VisiBroker 3.0 und JavaSofts Java IDL, wird durch Konfiguration ein Naming Service ausgewählt und allen beteiligten Prozessen zur Verfügung gestellt. Damit ein Prozeß, Client oder Server, einen Naming Service benutzen kann, genügt es, ihm den Zugriff auf den NamingContext des entsprechenden Naming Services über IIOP zu ermöglichen. Dazu benötigt das Programm die entsprechende Objekt-Referenz des NamingContexts, die nicht über den NamingService aufgelöst werden kann, da der Zugriff auf diesen noch nicht möglich ist ("Henne-Ei-Problem"). Eine CORBA-Objektreferenz kann als ein String, die sogenannte *Interoperable Object Reference (IOR)*, abgespeichert werden. Wenn diese IOR allgemein zugänglich ist, kann jedes Programm beim Bootstrapping die Referenz auf den NamingContext erhalten, mit dieser Referenz über IIOP auf den NamingContext zugreifen und damit den Naming Service benutzen.

3.1.2 Implementierung

Das eben beschriebene Konzept sollte für MASA je nach Konfiguration mit dem Naming Service von VisiBroker 3.0 oder JavaSofts Java IDL realisiert werden. Die Referenz des NamingContexts sollte auf einem WWW-Server abgelegt und beim Start von MASA über HTTP abfragbar sein. Die Entscheidung für einen WWW-Server mit Anfrage über HTTP läßt sich durch die relativ unkomplizierte Programmierung von WWW-Server und WWW-Clients in Java, sowie die Einfachheit des HTTP Protokoll rechtfertigen. Mögliche Alternativen wären die Übertragung der IOR via DHCP (komplexer) oder die Abspeicherung der IOR in einer Datei (unflexibler). Problematisch bei dem gewählten Konzept via HTTP ist allerdings die Konfiguration des Ports, an dem der WWW-Server Requests erwartet. Bei der Benutzung mehrerer unabhängiger MASA-Systeme auf einem Rechner, z.B. bei der Nutzung eines Rechners durch zwei User, dürfen nicht identische Portnummern verwendet werden.

NamingWebServer

Der beschriebene WWW-Server wurde in einer Java-Klasse als `NamingWebServer` (s. Abb. 3.2) realisiert. Seine Funktionalität umfaßt den Start des konfigurierten Naming Services (entweder VisiBroker 3.0 oder JavaSofts Java IDL) und die Bereitstellung der IOR des NamingContexts als WWW-Server.

main() Der Start des `NamingWebServers` erfolgt als Kommandozeilenaufruf bzw. in MASA durch ein Skript. An der Kommandozeile werden zwei Parameter erwartet, die zum Start des Naming Services von VisiBroker 3.0 nötig sind: Name der NamingFactory und Dateiname des Logfiles. Die `main()`-Methode erzeugt eine Instanz der Klasse `NamingWebServer`, initialisiert den WWW-Server durch `initWebServer()` und aktiviert den WWW-Server durch `waitForRequests()`(s. Abb. 3.3).

initWebServer() Zur Initialisierung des WWW-Servers wird der entsprechende Naming Service gestartet, je nach Konfiguration des Java-Propertys *de.unimuenchen.informatik.mnm.masa.-namingwebserver.orbtype*. Die Naming Services von VisiBroker und JavaSofts Java IDL unterscheiden sich in ihrer Implementierung erheblich: Der Naming Service von JavaSoft ist als native executable File realisiert, während der Naming Service des VisiBroker als Java *main()*-Methode implementiert ist. Auf dementsprechend unterschiedliche Weisen werden die Naming Services gestartet: Für den Naming

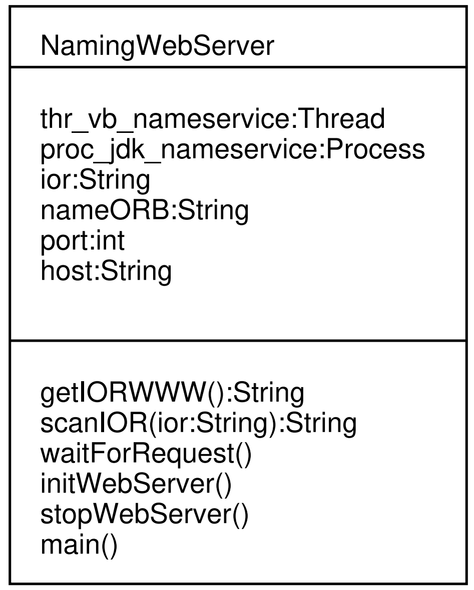

Abbildung 3.2: Komponenten des NamingWebServers

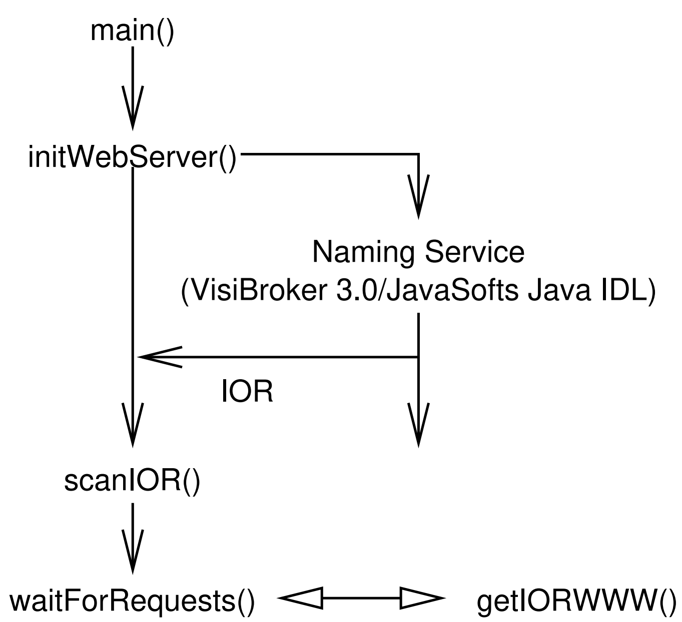

Abbildung 3.3: Funktionsweise des NamingWebServers (allgemein)

Service von JavaSoft muß ein eigener Prozeß kreiert werden, während für den Naming Service von VisiBroker ein eigener Java Thread genügt. Nach dessen Start wartet `initWebServer()` auf die Ausgabe der IOR des NamingContexts auf der Standardausgabe. Aus dem eingelesene String wird der exakte, Sonderzeichen freie IOR String durch `scanIOR()` gelesen. Bei Verwendung des VisiBroker Naming Service in der Standardkonfiguration von MASA liegt nicht die IOR des NamingContexts vor, sondern die der *Extended Naming Context Factory*, aus der der NamingContext aufgelöst wird. Dessen IOR wird anschliesend ermittelt und als IOR-String des Naming Context für den Web-Server abgespeichert.

scanIOR() Wie bereits oben erwähnt dient `scanIOR()` dazu aus dem eingelesenen IOR-String jegliche Sonderzeichen, die bei der Ausgabe der IOR des NamingContexts auf der Standardausgabe zur besseren Lesbarkeit (z.B. Zeilenumbruch) durch den Naming Service mitausgegeben werden, zu entfernen. Seine Funktionsweise wird in Abbildung 3.4 beschrieben.

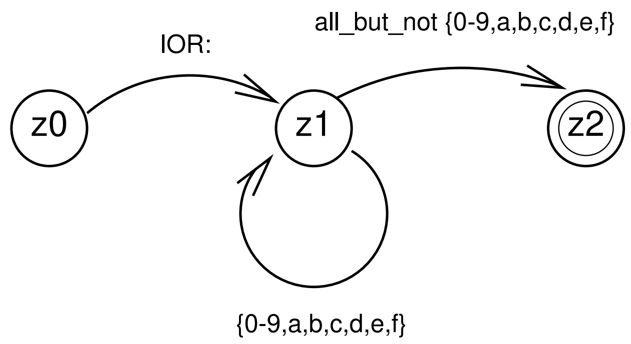

Abbildung 3.4: Scanner für IOR

waitForRequests() Nach der Initialisierung des WWW-Servers wird mit `waitForRequest()` ein Server Socket mit der in den Java-Propertys *namingwebserver.host* und *namingwebserver.port* konfigurierten Adresse erstellt und bei Anfragen an diese Adresse die IOR als Plaintext über HTTP zurückgegeben.

getIORWWW() Anfragen an den NamingWebServer können einfach durch die static-Methode `getIORWWW()` ausgeführt werden. Diese liefert als Rückgabewert die IOR. Voraussetzung ist, daß die Java-Propertys *namingwebserver.host* und *namingwebserver.port* entsprechend gesetzt sind. Bei einer Benutzung in MASA, d.h. in einer JVM, die mit einem MASA-Skript gestartet wurde ist dies der Fall.

Bug in VisiBroker 3.0 Ein allgemeine Benutzung des NamingWebServers mit dem NamingConext von VisiBroker und JavaSofts Java IDL ist aufgrund eines Bugs in VisiBroker 3.0 nicht möglich. In allen Helperklassen von VisiBroker 3.0, die zur Benutzung von Objekten über CORBA benötigt werden, ist ein Teil der Repository ID der Objekte falsch kodiert: statt "omg.org" "org.omg". Dies bedeutet mit einem nicht gepatchten VisiBroker 3.0 kann der NamingContext von JavaSofts Java IDL nicht benutzt werden, da die Helperklasse aufrund der falschen Repository ID die CORBA-Referenz des NamingContexts mit der narrow()-Methode nicht in ein NamingContext-Objekt umwandeln kann. Dies bedeutet eine Verwendung eines fremden Naming Services ist mit dem ORB von VisiBroker 3.0 nicht möglich, ebenso eine Verwendung des NamingServices von VisiBroker 3.0 mit einem fremden ORB, wie zum Beispiel JavaSofts JavaIDL. Um dies trotzdem zu realisieren, wurde als "Quick-Hack" eine Beta Version des *JDK 1.2* dem Fehler der Final Release von VisiBroker 3.0 angepaßt, da die Quellen von VisiBroker 3.0 nicht zugänglich waren. In der Version 3.3 des VisiBroker ist dieser Fehler behoben.

3.1.3 Einsatz in MASA

Der Einsatz des `NamingWebServers` erfolgt auf folgende Weise (s. Abb. 3.5):

1. Der NamingWebServer startet den konfigurierten NamingService.
2. Der Naming WebServer liest die IOR aus und wartet auf Anfragen
3. MASA stellt beim Bootstrapping Anfrage an NamingWebServer nach IOR
4. MASA löst IOR-Referenz über IIOP auf und hat Zugriff auf Naming Service

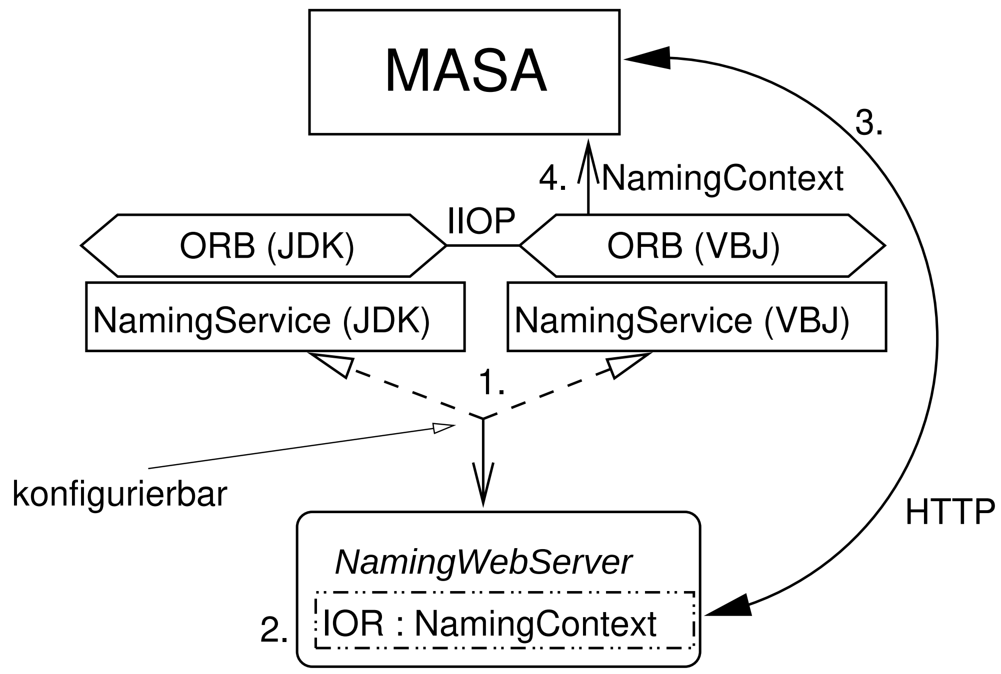

Abbildung 3.5: Funktionsweise des NamingWebServers (in MASA)

3.1.4 Konfiguration und Benutzung des NamingWebServers

Die Konfiguration des Namingwebservers erfolgt über Java Propertys: Welcher Naming Service gestartet wird, muß in *de.unimuenchen.informatik.mnm.masa.namingwebserver.orbtype* mit “JDK1.2” oder “VisiBroker” angegeben werden; die Adresse(Hostname, Portnummer) des WWW-Server wird in *de.unimuenchen.informatik.mnm.masa.namingwebserver.host* und *de.unimuenchen.informatik.mnm.-masa.namingwebserver.port* eingestellt. An der Kommandozeile muß der Name der NamingFactory und ein Dateiname für das Logfile des Naming Services übergeben werden. Die Benutzung in MASA erfolgt über den Befehl “masasSkript start naming” mit dem alle nötigen Paramter gesetzt und übergeben werden. Bei einem fehlerfreien Start wird die IOR des NamingContexts auf der Standardausgabe ausgegeben, sowie der Rechnername, für den der NamingWeb-Server in MASAs Makefile.DEF konfiguriert wurde, und der Name des Localhost zum Vergleich. Die Standardkonfiguration ist “localhost”, d.h. der NamingWeb-Server wird für den Rechner konfiguriert, auf dem das Skript *namingScript* erzeugt wurde. Nach diesen Angaben meldet der NamingWeb-Server seine Bereitschaft zusammen mit der Angabe, auf welchem Port er die HTTP-Requests erwartet.

Kapitel 4

MASA-Voyager Schnittstelle

Bei der Betrachtung der Agentensystemschnittstelle zwischen MASA und Voyager 2.0 war eine mögliche Anzeige und Steuerung der Voyager Agenten durch MASA Ausgangspunkt der Untersuchung. Dabei galt es die "kleinsten gemeinsamen Nenner" der beiden sehr unterschiedlichen Konzepte zu finden und darauf basierend ein Gateway zwischen beiden Welten zu realisieren.

4.1 Beschreibung der Voyager Technologie

Voyager 2.0 ist ein auf Java basierendes Agentenframework der Firma Objectspace[Obje b]. Es ermöglicht die Erstellung von Remoteobjekten und deren Migration als Agenten auf andere Server. Mit den von Voyager erstellten Remoteobjekten kann entweder über die proprietäre Voyager-Architektur oder auch über IIOP kommuniziert werden. Außerdem sind in Voyager CORBA ähnliche Services realisiert: Naming Service, Event Service. Auffallend in Voyager ist die Form seiner Java-Schnittstellen: Sie bestehen stellenweise hauptsächlich aus Static-Methodenaufrufen, was das Ergebnis inkonsequenter OO-Modellierung zu sein scheint. Über die Voyager zugrunde liegende Architektur ist leider kaum Information verfügbar.

4.1.1 Remote Objekte

Die Erzeugung von Remoteobjekte erfolgt in Voyager durch eine Factory-Klasse, die nach Erzeugung des Remoteobjekts ein Proxy Objekt zurückgeliefert, über das der Client mit dem Remoteobjekt kommuniziert. Als Interface für Remoteobjekte dient entweder ein entsprechendes Java-Interface oder aber nur das implementierte und in Bytecode übersetzte Java-Classfile, aus dem durch ein Voyager-Utility das nötige Interface des Objekts erzeugt wird. Eine eventuelle Migration des Remoteobjekts bleibt für den Client transparent. Die Nachrichten werden über das Proxy Objekt an das migrierte Objekt weitergeleitet (s. Abb. 4.1). Das Leben eines Objekts wird durch einen Distributed Garbage Collector (DGC) beendet, falls keine Referenzen mehr auf das Objekt bestehen. Der DGC wird standardmäßig alle 2 Minuten ausgeführt ([Obje b],s.S.12).

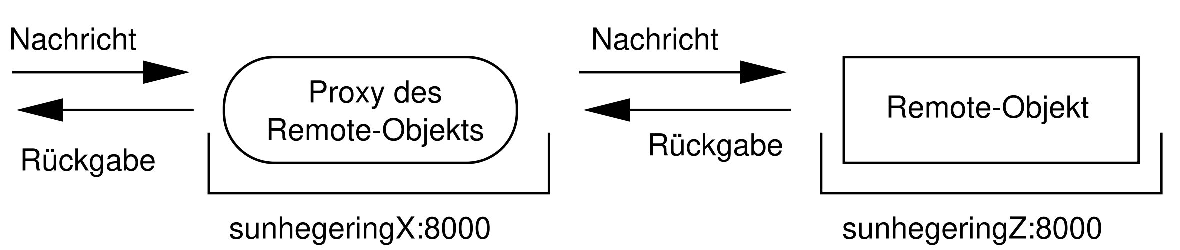

Abbildung 4.1: Voyager Remote Objekt

4.1.2 CORBA (IIOP)

Voyager ermöglicht eine Kommunikation über IIOP. CORBA Objekte können von Voyager allerdings nur über ihre IOR-Referenz ex- bzw. importiert werden, der in CORBA spezifizierte Naming Service wird von Voyager nicht unterstützt. Entsprechende Stubs und Skeletons für CORBA-Objekte werden in Voyager durch einen IDL-to-java-Compiler erstellt. Für diesen existieren keine Parameter zur Abbildung der IDL-Module in Java-Packages, die wiederum in einer Package-Hierarchie eingeordnet sind.

4.1.3 Naming Service

Der Name des Naming Service täuscht eine Ähnlichkeit mit dem gleichnamigen CORBA-Service nur vor. Zur Speicherung der Objektreferenzen und ihrer Bezeichnung wird ein Objekt der in Voyager enthaltenen Schnittstelle `IDirectory` ([Obje b], s.S. 47). Dieser Aufbau entspricht nicht der CORBA-Konvention (s. Kapitel 3.1), wodurch eine Benutzung in fremden CORBA ORBs unmöglich ist. Selbst sein Leistungsumfang innerhalb Voyagers zeigt im Vergleich zum CORBA-Service erhebliche Mängel: Es gibt keinen Mechanismus, um alle im Naming Service gebundenen Objekte auf zu listen [TE]. Diese Funktionalität wurde auch erst auf Anfrage bei Objectspace für eine spätere Version Voyagers in Erwägung gezogen.

4.1.4 Mobile Agenten

Remoteobjekte können auch vollkommen selbständig, ohne remote oder lokal referenziert zu sein, als autonome Agenten Aufgaben durchführen, ohne von der Garbage Collection gelöscht zu werden. Ein Voyager Agent bietet die Methode `setAutonomous()` an, um ihn als einen autonomen Agenten zu markieren. Wie bereits erwähnt bietet Voyager auch die Funktionalität, jedes Objekt bzw. Agenten auf einen anderen Server zu migrieren und dort seine Methoden bzw. Aufgaben ausführen zu lassen.

4.2 Unterschiede zwischen MASA und Voyager

Ein grundlegender Unterschied ist die Auslegung Voyagers als Framework für ein Agentensystem im Gegensatz zu MASA, das an sich ein Agentensystem darstellt. Voyager stellt nur die Komponenten zur Entwicklung eines solchen Agentensystems zur Verfügung. Dadurch erscheint die Entwicklung von Agenten mit Voyager Komponenten flexibler, da nicht die Integration in ein komplexes System erfolgen muß. Die Kontrolle der Agenten ist in MASA jedoch effizienter:

- Terminierung: Eine Terminierung des Agenten ist in Voyager nur durch den oben erwähnten Garbage Collection Mechanismus möglich. In MASA wird ein Agententhread explizit durch den AgentManager beendet.
- Migration: Ein Agent muß in Voyager vor seiner Migration selbstständig alle Methoden beenden. In MASA wird dies zentral vom AgentManager übernommen.

Die proprietäre Kommunikationsarchitektur der Voyager Technologie ermöglicht zwar die unkomplizierte Entwicklung von Agenten, wirkt aber der Verknüpfung mit fremden System entgegen, während MASA durch die CORBA Spezifikation eine standardisierte Schnittstelle auf der Kommunikationsebene bietet.

4.3 Voyager-MASA-Gateway

Trotz der unterschiedlichen Konzepte und Technologien bei Voyager und MASA bieten die Agenten beider Systeme grobgranular betrachtet ähnlichen Funktionalität. Man kann in beiden Systemen Agenten, die über einen Naming Service referenziert werden können, erstellen, migrieren und terminieren.

Auf Basis dieser Gemeinsamkeiten sollte eine Anbindung der Voyager Technologie an das MASA-Agentensystem in Form des Voyager-MASA-Gateways realisiert werden.

4.3.1 Aufgabe

Das Gateway zwischen MASA und Voyager soll folgende Funktionen in MASA zur Benutzung von Voyager-Agenten erfüllen:

- Monitoring: Anzeige aller Voyager Agenten die auf einem Voyager Server aktiv sind im MASA Naming Service.
- Steuerung: Erzeugung, Migration und Terminierung von Voyager-Agenten durch das MASA-System

4.3.2 Lösungskonzept

Das Voyager-MASA-Gateway soll durch einen *stationären* und *globalen* MASA Agenten realisiert werden, um ein zentrales Gateway zwischen MASA und Voyager zu schaffen, deshalb *global*, und um mögliche Seiteneffekte die durch Migration entstehen könnten zu vermeiden, deshalb *stationär*. Dieses *VoyagerMasaGateway* kontrolliert als MASA Agent die Komponenten zum Monitoring und zur Steuerung von Voyager Agenten.

Monitoring von Voyager Agenten

Das geforderte Monitoring erfolgt durch Abbilden des Voyager Naming Service in den MASA Naming Service, indem in MASA für jeden Voyager Agenten als Repräsentant ein mobiler MASA Agent, der *VoyagerProxyAgent*, erstellt wird und bei Terminierung des Voyager Agenten ebenfalls terminiert wird. Diese Abbildung wird durch ein eigenes Java-Programm, das *NamingGateway*, unter Kontrolle des *VoyagerMasaGateways* realisiert. Die Entscheidung für ein eigenes Programm beruht auf der Anforderung eine von MASA unabhängige Komponente zur Abbildung des Voyager *Namespaces* in einen CORBA Naming Service zu schaffen. Beim Start des *NamingGateways*, der bei Erzeugung des *VoyagerMasaGateway* erfolgt, schickt das *NamingGateway* eine Liste aller aktiven Voyager Agenten an das *VoyagerMasaGateway*. Danach werden nur immer die Listen der auf den Voyager Servern neu erstellten bzw. terminierten Agenten an das *VoyagerMasaGateway* geschickt. Um die *VoyagerProxyAgenten* zu erstellen bzw. zu terminieren, benutzt das *VoyagerMasaGateway* Methoden des *VoyagerAgentManagers* (die Funktionsweise des *VoyagerAgentManagers* wird in Abschnitt 4.3.2 näher erläutert). Die Voyager Server, die das *NamingGateway* überwacht, können vor Start des *NamingGateways* bereits aktiv sein und Agenten geladen haben oder noch gar nicht gestartet sein. Falls ein Voyager Server während der Laufzeit des *VoyagerNamingGateways* terminiert wird, meldet das *NamingGateway* alle Agenten dieser Servers dem *VoyagerMasaGateway* als terminiert. Das Monitoring der Voyager Agenten erfolgt in folgenden Schritten (s.Abb.4.2):

1. Ein Voyager Agent wird auf Voyager Server kreiert bzw. terminiert
2. Das *NamingGateway* registriert die Veränderung im Voyager Namespace und schickt die entsprechende Nachricht an das *VoyagerMasaGateway* in MASA
3. Das *VoyagerMasaGateway* reicht die Nachricht an den *VoyagerAgentManager* weiter
4. Der *VoyagerAgentManager* erzeugt über die MASA-Agentensystemschnittstelle einen *VoyagerProxyAgenten*
5. Das MASA *AgentManager* erzeugt einen *VoyagerProxyAgenten*
6. Der Name des *VoyagerProxyAgenten* wird im MASA Namensverzeichnis in einem speziellen Voyager-Verzeichnis eingetragen.

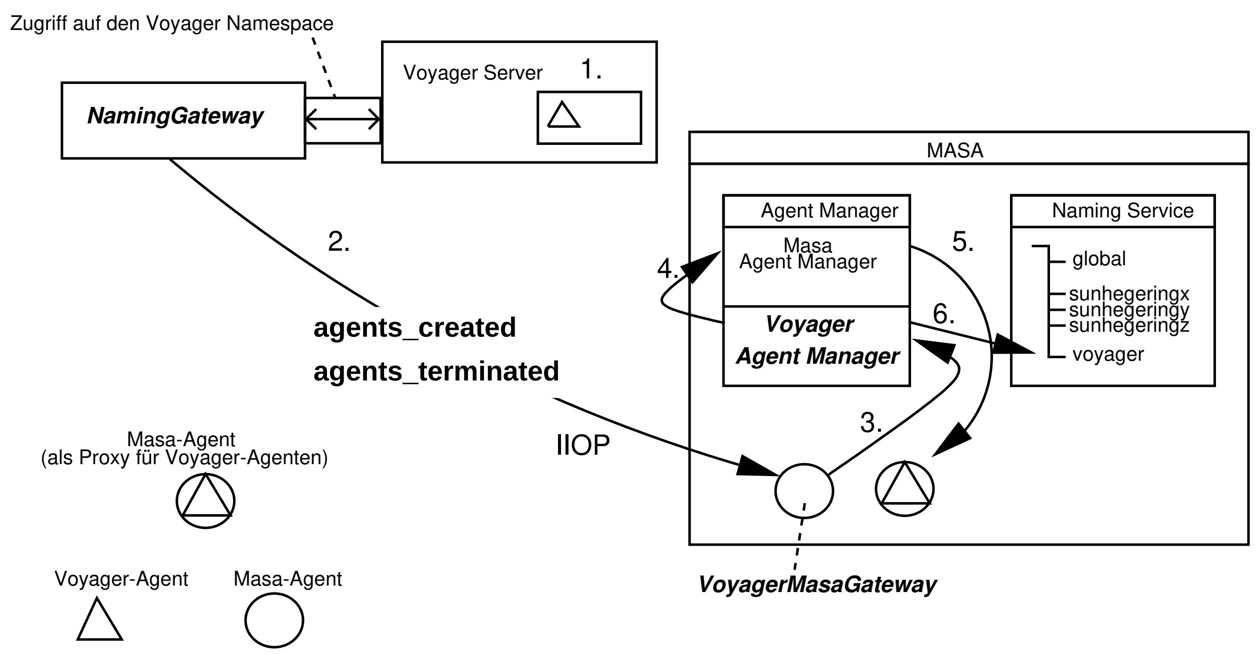

Abbildung 4.2: Monitoring der Voyager Agenten in MASA

Steuerung der Voyager Agenten

Zur Kontrolle der Voyager Agenten wird MASA erweitert, so daß ein Agent Manager für ein fremdes Agentensystem, wie zum Beispiel Voyager, von MASA zur Steuerung von Agenten des fremden Agentensystems verwandt wird. Dazu muß zum einen in MASA die Möglichkeit zur Registrierung eines alternativen Agent Managers gegeben sein als auch die Klassifizierung durch das MASA-System der Aufrufe nach dem jeweiligen Agentensystem für die Zuordnung des entsprechenden Agent Managers.

Für die Steuerung der Voyager Agenten von MASA aus stellt das *VoyagerMasaGateway* als MASA Agent [Kemp 98] ein Java-Applet zur Verfügung. Über dieses Applet können Voyager Agenten

- erzeugt,
- migriert,
- terminiert,
- und deren Methoden ausgeführt werden.

Die Ausführung der jeweiligen Aktionen werden vom *VoyagerMasaGateway* entweder an die MASA-Agentensystemschnittstelle weitergeleitet oder direkt dem *VoyagerAgentManager* übergeben. Dies ist davon abhängig, ob die entsprechenden Voyager Aktionen äquivalent an der MASA-Agentensystemschnittstelle vorhanden sind (Erzeugung,Terminierung von Agenten) oder nur direkt vom *VoyagerAgentManager* durchgeführt werden können (Ausführung von Methoden, Migration)). Der *VoyagerAgentManager* muß dazu beim MASA System vom *VoyagerMasaGateway* registriert werden, damit Aufrufe an der MASA-Agentensystemschnittstelle für Voyager Agenten behandelt werden können. Die Steuerung der Voyager Agenten von MASA aus über das Applet des *VoyagerMasaGateways* geschieht folgendermaßen(s.Abb.4.3):

1. *Das VoyagerMasaGatewayApplet* schickt die Benutzereingaben zur Erzeugung, Migration und Terminierung der Voyager Agenten an das *VoyagerMasaGateway*

2. Das *VoyagerMasaGateway* leitet diese Angaben an die MASA-Agentensystemschnittstelle weiter, bzw. (2a.) direkt an den *VoyagerAgentManager*, falls die Aufrufe aufgrund der unterschiedlichen

Konzeption von MASA und Voyager an der MASA-Agentensystemschnittstelle nicht definiert sind. Auf diese Aufrufe wird in der Implementationsbeschreibung näher eingegangen.

3. Der *VoyagerAgentManager* bildet die entsprechenden Operationen der Voyager Agenten auf die *VoyagerProxyAgenten* in MASA über die MASA Systemschnittstelle ab, d.h. reicht die Nachrichten über Erzeugung und Terminierung an die MASA-Systemschnittstelle weiter. Die Migration eines Voyager Agenten hat keine Effekt für seinen *VoyagerProxyAgenten* in MASA und wird deshalb nicht weitergereicht.

4. Das MASA-System führt die gewünschten Steuerungsoperationen auf den *VoyagerProxyAgenten* aus.

5. Der *VoyagerAgentManager* nimmt die erforderlichen Veränderungen im Voyager-Verzeichnis des MASA-Namendienstes durch: Eintragung eines Agenten bei Erzeugung, Löschen bei Terminierung

6. Nach der erfolgreichen Beendigung der Aktionen in MASA, führt der *VoyagerAgentManager* die gewünschten Steuerungsoperationen auf den Voyager Agenten selbst aus. Fehler während der Ausführunf werden dem Benutzer im Applet des *VoyagerMasaGateways* angezeigt.

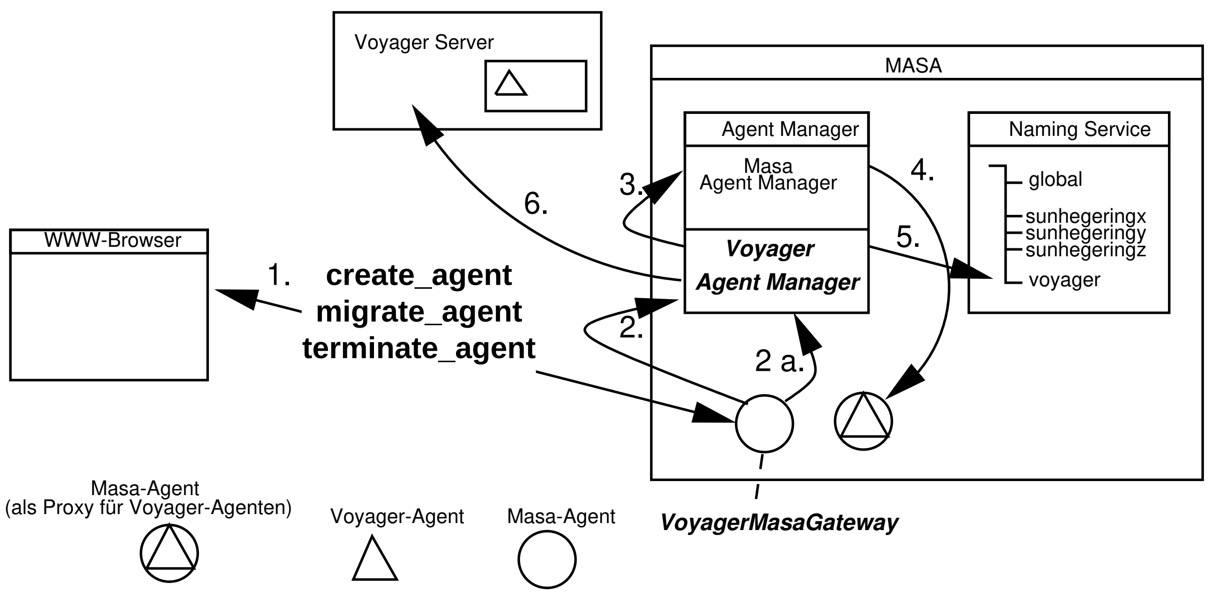

Abbildung 4.3: Steuerung der Voyager Agenten durch MASA

Veränderungen an MASA

Neben der Erweiterung MASAs um die oben genannten Komponenten sind Modifikationen am MASA-System selbst notwendig. Zur Realisierung des Konzepts, Voyager Agenten in MASA durch *VoyagerProxyAgents* darzustellen, muß es möglich sein, Agenten mit unterschiedlichen Namen, aber gleichnamiger Java-Klasse als Implementierung zu erstellen. Da im bestehenden MASA aber der Klassenname eines Agenten durch seinen Agentennamen bestimmt wird, kann das oben erwähntes Konzept nicht realisiert werden. Die Lösung liegt in der Modifikation MASAs, so daß Agentenname und Klassenname seiner Implementierung von einander unabhängig sind und mehrere Agenten als Instanzen der gleichen Klasse in einem MASA-Agentensystem verwendet werden können. Außerdem muß in MASA die Integration des VoyagerAgentManagers als Alternative zum vorhandenen *AgentManager* ermöglicht werden, um eine Steuerung der Voyager Agenten über die MASA-Agentensystemschnittstelle zu realisieren.

NamingGateway

Das NamingGateway verwaltet die Liste der Voyager-Agenten und meldet entsprechende Veränderungen im Voyager-Namespace (Agentenerzeugung, Terminierung) an das VoyagerMasaGateway. Der Voyager-Namespace ist dem CORBA Naming Service ähnlich, bietet jedoch nicht den gleichen Funktionsumfang (s. oben).

VoyagerMasaGateway

Bei dem *VoyagerMasaGateway* handelt es sich um einen globalen (= exklusiven), stationären Agenten in MASA. Nach seiner Erstellung, startet er das *NamingGateway* in einem eigenen Thread und erzeugt einen *VoyagerAgentManager*, den er im MASA-System als Agent Manager für Voyager Agenten registriert. Außerdem stellt er ein Java-Applet zur Steuerung der Voyager-Agenten zur Verfügung. Benutzereingaben für die Verwaltung des Agenten im *VoyagerMasaGatewayApplet* werden an das *VoyagerMasaGateway* weitergeleitet und von ihm über die Agentensystemschnittstelle von MASA an den *VoyagerAgentManager* weitergereicht bzw. direkt an den *VoyagerAgentManager* gerichtet, falls die Aufrufe aufgrund der Unterschiede zwischen Voyager und MASA an der MASA-Agentensystemschnittstelle nicht definiert sind.

VoyagerMasaGatewayApplet

Das *VoyagerMasaGatewayApplet* realisiert als Java-Applet die Steuerung, d.h. Erzeugung, Migration und Terminierung von Voyager Agenten über Benutzereingaben, die an das *VoyagerMasaGateway* weitergeleitet werden. Die Implementierung der Oberflächenelemente erfolgt mit Java-Swing-Komponenten.

VoyagerAgentManager

Der *VoyagerAgentManager* wird vom *VoyagerMasaGateway* über die MASA-Agentensystemschnittstelle als AgentManager für Voyager Agenten registriert und verarbeitet alle Systemaufrufe, die die Steuerung von Voyager Agenten betreffen. Der *VoyagerAgentManager* soll folgende Funktionen erfüllen: Erzeugung, Migration und Terminierung von Voyager Agenten, Erzeugung und Terminierung von Voyager Proxy Agenten, Vermittlung von Information über Voyager Agenten und Server.

VoyagerProxyAgent

Um einen Voyager Agenten in MASA darzustellen, wird ein VoyagerProxyAgent für jeden Voyager Agenten erzeugt. Seine gesamte Funktionalität beschränkt sich auf die Umleitung der URL seines Applets zur URL des VoyagerMasaGatewayApplets.

FOOVoyagerAgent

Anhand eines beispielhaften Voyager Agenten soll das VoyagerMasaGateway getestet werden. Dieser soll die nötige Palette an Methodenaufrufen bieten, um einen realistischen Agenten zu repräsentieren.

4.4 Implementierung

Die Implementierung zerfällt in die notwendigen Modifikationen an MASA und die Erstellung des VoyagerMasaGateways als Stationärer Agent in MASA.

4.4.1 Modifikationen an MASA

AgentSystem

Erweiterungen und Anpassungen an der Klasse `AgentSystem` und dessen IDL-Schnittstelle `AgentSystemService` sind für die Realisierung zur Nutzung verschiedener Agent Manager, je nach Agentensystemtyp, nötig. Im Folgenden werden nur die modifizierten Methoden der Klasse `AgentSystem` und die Veränderungen an den Attributen beschrieben.

Neues Attribut _agentManagerTable Zur Speicherung alternativer Agent Manager erhält `AgentSystem` ein zusätzliches Attribut, die `_agentManagerTable`. Die `_agentManagerTable` ist eine Hashtable die als Key den Agentensystemtyp und als Value den Agent Manager enthält, der eine Subklasse der abstrakten Klasse `GenericAgentManager` ist.

void registerAgentManager(CfMAF.Name name, GenericAgentManagerService agentManager) `registerAgentManager()` ist eine neue Methode in der IDL-Schnittstelle `AgentSystemService` und in der Implementierungsklasse `AgentSystem`. Mit `registerAgentManager()` werden neue Agent Manager zur `_agentManagerTable` hinzugefügt. Der Parameter `name` ist nur für die Festlegung des Agentensystemtyp notwendig. Er ist in `name.agent_system_typ` abgelegt.

getAgentManager(CfMAF.Name agent_name) Das Attribut `_agentManager` wird mit Einführung der `_agentManagerTable` überflüssig. Für den korrekten Zugriff auf die `_agentManagerTable` sorgt `getAgentManager()`, wobei der Agentensystemtyp in `agent_name` analog zu `registerAgentManager()` enthalten sein muß. Falls nur der Standard MASA `AgentManager` benutzt wird, genügt der Aufruf `getAgentManager(`*`null`*`)`.

Ersetzung _agentManager durch getAgentManager() Die Ersetzung des `_agentManager` erfolgt durch Aufruf der Methode `getAgentManager()`, wobei als Parameter der Agentensystemtyp in einem `CfMAF.Name` verpackt übergeben werden muß. Falls diese Information nicht zur Verfügung steht, wie zum Beispiel bei `list_all_agents()`, wird der Standard MASA AgentManager verwandt (s. oben). Die Ersetzung wird bei folgenden Methoden (zu ihrer genauen Beschreibung s. [Kemp 98]S.52) durchgeführt:

- `org.omg.CORBA.Object connectToAgent(CfMAF.Name agentName)`
- `java.lang.String getURL()`
- `Name create_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, byte[] arguments, ClassName[] class_names, String code_base, MAFAgentSystem class_provider)`
- `AgentStatus get_agent_status(Name agent_name)`
- `AuthInfo get_authinfo(Name agent_name)`
- `CfMAF.Name[] list_all_agents()`
- `CfMAF.Name[] list_all_agents_of_authority(byte[] authority)`
- `void receive_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, ClassName[] class_names, String code_base, MAFAgentSystem agent_sender)`
- `void resume_agent(Name agent_name)`

- `void suspend_agent(Name agent_name)`
- `void terminate_agent(Name agent_name)`
- `void terminate_agent_system(boolean nameService)`
- `void setAgentSystemService(AgentSystemService ass)`
- `void writeAgentToFile(CfMAF.Name agentName, java.lang.String file)`
- `CfMAF.Name readAgentFromFile( java.lang.String file)`

GenericAgentManager

Die neue Oberklasse für alle AgentManager, d.h. dem Standard MASA AgentManager und dem VoyagerAgentManager, wird durch die abstrakte Klasse GenericAgentManager realisiert. Die Verwendung eines Java-Interfaces statt einer abstrakten Klasse ist nicht möglich, da nicht, wie bei einem Interface möglich, nur public Methoden spezifiziert werden müssen. Der GenericAgentManager enthält folgende abstrakte Methoden:

- `org.omg.CORBA.Object connectToAgent(Name agent_name)`
- `String getAgentSystemURL()`
- `Name create_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, byte[] arguments, ClassName[] class_names, String code_base, MAFAgentSystem class_provider)`
- `AgentStatus get_agent_status(Name agent_name)`
- `AuthInfo get_authinfo(Name agent_name)`
- `CfMAF.Name[] list_all_agents()`
- `CfMAF.Name[] list_all_agents_of_authority(byte [] authority)`
- `void receive_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, ClassName[] class_names, String code_base, MAFAgentSystem agent_sender)`
- `void resume_agent(Name agent_name)`
- `void suspend_agent(Name agent_name)`
- `void terminate_agent(Name agent_name)`
- `void terminateAgentManager()`
- `void setAgentSystemService(AgentSystemService ass)`
- `void writeAgentToFile(CfMAF.Name agentName, java.lang.String file)`
- `CfMAF.Name readAgentFromFile(java.lang.String file)`

Da der `GenericAgentManager` eigentlich nur als erweitertes Interface fungiert enthält die Klasse keinerlei konkrete Implementierung.

AgentManager

Neben der Benutzung alternativer Agent Manager muß in MASA die Einschränkung bezüglich der Namensgleichheit von Agent und seiner IDL-Schnittstelle aufgehoben werden, so daß die Erstellung mehrerer Agenten des gleichen Interfaces in einem MASA Namensverzeichnis möglich wird. Dazu muß die Verwendung der Parameter der `create_agent()`-Methode des `AgentManager` gemäß der MASIF-Spezifikation angepaßt werden (s. unten 4.4.1). Für die Benutzung alternativer Agent Manager muß AgentManager eine Subklasse des abstrakten GenericAgentManager werden (s. Abschnitt 4.4.1). Dies hat jedoch keine Auswirkungen, da im `GenericAgentManager` nur eine echte Untermenge der Methoden des `AgentManager` deklariert ist.

Name create_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, byte[] arguments, ClassName[] class_names, String code_base, MAFAgent-System class_provider) Bei der Erzeugung eines Agenten wird nun nicht mehr der Klassenname seiner Implementierung durch seinen eigenen Namen angegeben, sondern im Parameter `class_names`. Dieser wurde bisher dazu verwandt die Typen der Konstruktorargumente zu übergeben. Da jedoch diese beim Aufruf selbst vorhanden sind, können ihre Typen durch Introspektion ermittelt werden. Von `class_names` wird nur das erste Element des Arrays benutzt. Falls `class_names` die Länge 0 hat, wird der Klassenname wie bisher aus dem Agentennamen berechnet.

AgentSystemApplet

Durch eine veränderte Benutzung des Parameters `class_names` in `AgentManager.create_agent()` (s. oben 4.4.1) müssen die Parameter für den Aufruf der Methode im `AgentSystemApplet` angepaßt werden. Die Angabe der Typen aus der im Applet angezeigten Tabelle der Konstruktorenargumente wird ignoriert und die Länge von `class_names` auf 0 gesetzt, d.h. der Agentenname entspricht dem Klassenname der Implementierung.

AgentSecurityManager

Zur Erstellung von Remote Objekten auf Voyager Servern durch den `VoyagerAgentManager` muß die Überprüfung des *MemberAccess* im `AgentSecurityManager` deaktiviert werden, um die Erzeugung einer *RMISecurityException* durch den Voyager Client zu vermeiden. Dies erfolgt durch Überschreiben der Methode `CheckMemeberAccess()`. Die dadurch verminderte Sicherheit im MASA-System wird durch ein verändertes Sicherheitskonzept im Rahmen der oben erwähnten Diplomarbeit von Harrald Rölle wiederhergestellt werden.

4.4.2 Zusatzkomponenten für MASA

Neben den beschriebenen Modifikationen ist das Gateway zwischen MASA und Voyager durch Zusatzkomponenten für MASA realisiert:

- *VoyagerMasaGateway* als `VoyagerMasaGatewayStationaryAgent` mit `NamingGateway`, `VoyagerAgentManager` und `VoyagerMasaGatewayApplet`
- *VoyagerProxyAgent* als `VoyagerProxyAgentMobileAgent`

Die Aufgaben des *VoyagerMasaGateways*, Monitoring und Steuerung der Voyager Agenten, wird grundsätzlich jeweils durch eine eigene Komponente realisiert. Das Monitoring wird durch ein Java-Programm, das `NamingGateway`, realisiert, die Steuerung der Voyager Agenten durch den `VoyagerAgentManager`, der als Agent Manager in MASA registriert wird. Die Steuerung und Koordination dieser beiden Komponenten erfolgt durch den `VoyagerMasaGatewayStationaryAgent`. Um die

Namensauflösung für Voyager Agenten zu vereinfachen, wird als Randbedingung ein globaler Namensraum für Voyager Agenten vorgegeben. Die Namensauflösung für gleichnamige Agenten wird im Absatz "VoyagerAgentManager" in 4.4.2 beschrieben.

VoyagerMasaGatewayStationaryAgent

Der `VoyagerMasaGatewayStationaryAgent` ist die Implementierung eines exklusiven globalen MASA Agenten. Die Klasse `VoyagerMasaGatewayStationaryAgent` ist eine Unterklasse der Klasse `Agent`, die als Oberklasse für die Erstellung eines MASA Agenten dient [Kemp 98]. Nach seiner Erstellung startet er das `NamingGateway` und registriert beim MASA System (AgentSystem) den `VoyagerAgentManager` als Agent Manager für Voyager Agenten. Dazu wird die Methode `registerAgentManager()` verwandt, um die die MASA Klasse AgentSystem erweitert werden muß. Seinem Java-Applet stellt der `VoyagerMasaGatewayStationaryAgent` alle zur Steuerung von Voyager Agenten notwendigen Methoden zur Verfügung. Die Steuerung von Voyager Agenten umfaßt nicht nur die Erzeugung, Migration und Terminierung, sondern zusätzlich die Ausführung von Methoden der Voyager Agenten. Die Aufrufe der Steuerungsmethoden vom Applet leitet er über die Schnittstelle des MASA-Systems an den in MASA registrierten `VoyagerAgentManager` weiter bzw. direkt an den `VoyagerAgentManager`, falls an der MASA Schnittstelle die für das Voyager Konzept notwendigen Methoden, z.B. Ausführung einer Methode eines Voyager Agenten, nicht vorgesehen sind.

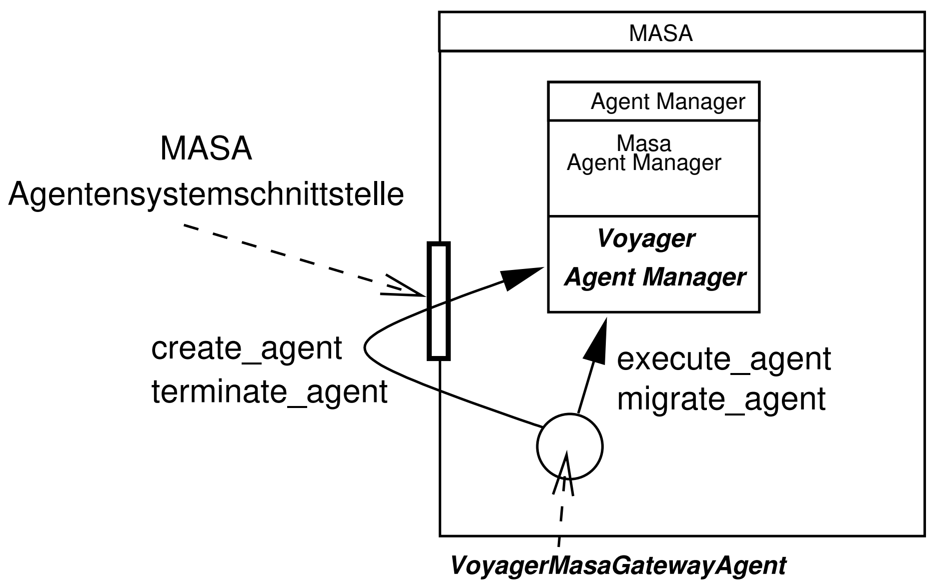

Abbildung 4.4: MASA Agentensystemschnittstelle

Konstruktor Im Konstruktor wird die Liste der Voyager Server, die bei der Erzeugung des `VoyagerMasaGatewayStationaryAgent` bekannt sein müssen, aus dem Property *de.unimuenchen.-informatik.mnm.masa.voyager.server=host:port[,host:port,...]*, das in Makefile.DEF gesetzt wird, ausgelesen. Die einzelnen Voyager Server müssen jeweils mit Rechnernamen und Portnummer durch einen Doppelpunkt getrennt angegeben werden. Die einzelnen Voyager Server werden in der Liste durch Kommata getrennt. Nachdem an die Rechnernamen ihre Domain angehängt ist, wird die Liste der Voyager Server in dem Attribut `_voyagerServer` abgelegt.

void run Die `run()`-Methode wird von der Oberklasse `Agent` geerbt und überschrieben. Sie wird beim Start des `VoyagerMasaGatewayStationaryAgent` automatisch vom MASA System gestartet. Zunächst wird überprüft, ob der `VoyagerMasaGatewayStationaryAgent` als *global* definiert ist. Ist er nicht als *global* deklariert wird er in den *suspend*-Status versetzt und bricht die `run()`-Methode ab. Damit ist der `VoyagerMasaGatewayStationaryAgent als` unbrauchbar markiert und muß nach einer anschließenden Terminierung neu gestartet werden. Nach einer erfolgreichen Kontrolle seiner Eigenschaft bezüglich der Globalität wird im MASA Namensverzeichnis auf Ebene der globalen Agenten ein neues Namensverzeichnis für Voyager Agenten angelegt (s.Abb.4.5). Anschließend wird eine Instanz des

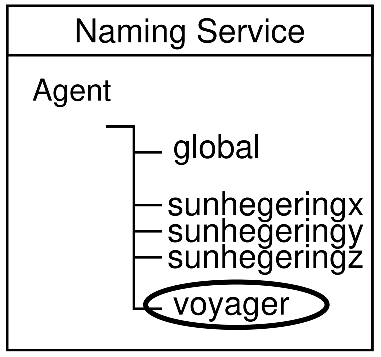

Abbildung 4.5: Namensverzeichnis "Voyager" in MASA

`VoyagerAgentManager` erstellt und beim MASA System registriert. Eine Referenz auf das MASA System Objekt wird mit `_agentManager.getAgentSystemService()` ermittelt, wobei `_agentManager` als Attribut in jedem MASA Agenten den MASA `AgentManager` referenziert. An das MASA System Objekt kann nun die Nachricht `registerAgentManager()` mit dem instanzierten `VoyagerAgentManager` und dem Agentensystemtyp für Voyager geschickt werden. Der Agentsystemtyp ist als short-Variable definiert und wird im Makefile.DEF von MASA für Voyager mit *de.unimuenchen.informatik.mnm.-masa.voyager.type* = "Agentensystemtyp" gesetzt und kann dann aus der gleichnamigen Java-Property ausgewertet werden . Der Standardwert für MASA ist 4, der für Voyager ist mit 5 festgelegt. Eine Referenz auf den im MASA registrierten `VoyagerAgentManager` wird in `_voyagerAgentManager` gespeichert. Nach der erfolgreichen Registrierung des `VoyagerAgentManagers` wird die `main()`-Methode des `NamingGateways` in einem eigenen Thread gestartet. Dazu wird die Liste der Voyager Server, die im Attribut `_voyagerServer` (s. oben "Konstruktor" in 4.4.2) gespeichert sind, übergeben.

String create_agent(String agentname, String classname, String packagename, String place_name) Diese Methode dient zur Erzeugung von Voyager Agenten und wird vom Applet des `VoyagerMasaGatewayStationaryAgent`, dem `VoyagerMasaGatewayApplet`, über CORBA aufgerufen. Beim Aufruf muß der Name des Agenten, der Klassenname seiner Implementierung, falls er sich vom Agentennamen unterscheidet, und der Name des Voyager Servers angegeben werden. Das Konzept der Erstellung eines Agenten ist bezüglich der Parameter in MASA und Voyager ähnlich und kann deshalb über die MASA-Agentensystemschnittstelle abgewickelt werden. Dazu werden in `create_agent()` zunächst die notwendigen Parameter zusammengestellt. Dabei sind für den Voyager Agenten hauptsächlich der Agentenname, der Klassenname des Agenten und die Bezeichnung seines Voyager Servers, als Place bezeichnet, von Bedeutung. Die restlichen Parameter, die bei einem `create_agent()`-Aufruf an der MASA-Agentensystemschnittstelle benötigt werden, werden mit Standard bzw. Dummy-Werten belegt. Das sind im einzelnen: `agent_profile, arguments, code_base, class_provider`. Zu beachten ist, daß es sich um einen Aufruf über CORBA handelt und somit null-Werte nicht verwendet werden dürfen. Nach der Umwandlung des Agentennamen von String in CfMAF.Name wird der Agentensystemtyp auf Voyager gesetzt (s. oben). Der Rückgabewert der create_agent()-Methode des MASA Systems enthält den Agentennamen der bei Gleichheit des ursprünglichen Agentennamens mit einem bestehenden Voyager Agenten vom VoyagerAgentManager angepaßt wurde (s. "create_agent()" in 4.4.2). Dieser modifizierte Agentename wird als String auch von `create_agent()` weiter zurückgegeben. Vor Beendigung der Methode muß dieser Vorgang durch Ablage des Agentennamens in dem Attribut `_agentsCreatedByApplet` gespeichert werden, damit eine spätere Meldung vom `NamingGateway` über Erstellung eines Agenten differenziert behandelt werden kann, je nachdem ob ein Agent innerhalb MASAs, d.h. durch das Applet, oder außerhalb MASAs auf einem Voyager Server erstellt wurde.

String execute_agent(String agent_name, String method, String arguments) Bei der Möglichkeit Methoden eines Agenten auszuführen, handelt es sich um ein Voyager Konzept, das keine Äquivalenz in MASA besitzt, weshalb nicht die MASA-Agentensystemschnittstelle benutzt werden kann, sondern ein Aufruf direkt am `VoyagerAgentManager` erforderlich ist. Wie `create_agent()` ist die Methode `execute_agent()`[1] für einen Aufruf durch das `VoyagerMasaGatewayApplet` vorgesehen. Die Benutzereingaben am Applet, werden von diesem als Strings direkt an `execute_agent()` weitergegeben. Für einen Aufruf der `execute_agent()`-Methode des `VoyagerAgentManager` muß der Parameter `arguments` in einen Object-Array umgeformt werden. Die Liste der Argumente für die Methode des Voyager Agenten werden am Applet folgendermaßen eingegeben (Beschreibung in BNF):

$$
\begin{aligned}
\text{Liste der Argumente} &::= Argument[',' Argument] \\
Argument &::= Package.Classname \sqcup Wert \\
Package.Classname &::= \text{Klassenname mit Packageangabe des Arguments} \\
Wert &::= \text{Wert des Arguments}
\end{aligned}
$$

Beispiel: “`java.lang.String Hello World!,java.lang.Integer 3`”
Durch diese Syntax ist die Verwendung von Kommata in Werten nicht möglich, z.B. ist nicht erlaubt: “`java.lang.String ein Komma, ein kleines`”

Der Argumentenstring wird aufgelöst und durch den Reflection-Mechanismus von Java werden die Klassen der einzelnen Argumente geladen und instanziert. Nach erfolgreicher Umwandlung der Argumente in einen Object-Array wird die `execute_agent()`-Methode des `VoyagerAgentManagers` aufgerufen, die den Rückgabewert der Methode als String bzw. den String “`void`”, falls die Methode des Agenten mit Rückgabewert *void* definiert ist, zurück liefert. Dieser vom `VoyagerAgentManager` zurückgelieferte Wert wird an das Applet weitergereicht.

void migrate_agent(String agent_name, String target, String execute, String arguments) Wie die zuvor beschriebene `execute_agent()`-Methode muß auch `migrate_agent()` sich direkt an den `VoyagerAgentManager` wenden, um einen Voyager Agenten zu migrieren. Neben der Angabe des Agentennamen und des Voyager Servers, hier `target` genannt, wird ein Methodenname - `execute` - und die nötigen Argumente - `arguments` - für die Methode übergeben. Voyager bietet die Möglichkeit nach einer Migration eine Methode auszuführen, wobei ein Rückgabewert dieser Methode nicht verwertet werden kann. Die Umwandlung der Argumente geschieht analog wie oben beschrieben bei `execute_agent()`. Zur Migration und Ausführung des Agenten wird die gleichnamige Methode `migrate_agent()` des `VoyagerAgentManagers` aufgerufen. Nach erfolgreicher Migration muß der Agentenname in die Vektoren `_agentsCreatedByApplet` und `_agentsTerminatedByApplet` eingetragen werden, da der Voyager Agent vom VoyagerAgentManager aus dem Namensverzeichnis des Voyager Servers, von dem wegmigriert wurde, ausgetragen und in des Namenverzeichnis des neuen Voyager Servers eingetragen wird. Dieser Vorgang vom `NamingGateway` als Terminierung und Erzeugung eines Agenten registriert, darf jedoch nicht beachtet werden (s. Absatz “migrate_agent()” in 4.4.2).

[1] Die Bezeichnung der Methode `execute_agent()` soll die Ausführung eines Voyager Agenten als Äquivalenz zur standardmäßigen Ausführung der `run()`-Methode eines MASA Agenten betonen

void terminate_agent(String agent_name) Im Gegensatz zur Ausführung und Migration kann die Terminierung eines Agenten wieder über die Agentensystemschnittstelle MASAs abgewickelt werden. Zur Terminierung genügt die Angabe des Agentennamen. Dieser String wird in ein Datum des Typs `CfMAF.Name` umgewandelt, wobei der Agentensystemtyp auf Voyager gesetzt wird, damit das MASA System die Behandlung der Agententerminierung dem registrierten VoyagerAgentManager überläßt und nicht dem MASA `AgentManager`. Nach erfolgreicher Terminierung durch MASA muß dieser Vorgang analog zur Erstellung eines Agenten durch Ablage des Agentennamen in `_agentsTerminatedByApplet` protokolliert werden, um eine Meldung des `NamingGateway` über terminierte Agenten klassifizieren zu können.

void agents_created(String[] agent_names, String place_name) Diese Methode ist zur Verwendung durch das `NamingGateway` gedacht. Sie dient als Nachricht des `NamingGateway`, um die Erstellung von Voyager Agenten zu melden. In den Parametern werden zum einen die Namen der neu erstellten Agenten - `agent_names` - gemeldet, sowie deren Voyager Server - `place_name`. Falls diese Agenten durch das `VoyagerMasaGatewayApplet` erstellt wurden, ist dies im Attribut `_agentsCreatedByApplet` gespeichert. Der Name des Agenten wird aus `_agentsCreatedByApplet` ausgetragen und die Methode `agents_created()` ohne weiter Aktionen beendet. Ist der Voyager Agent außerhalb MASAs erstellt worden, dann wird der `VoyagerAgentManager` beauftragt einen `VoyagerProxyAgenten` zu erstellen, um die erstellten Voyager Agenten in MASA darzustellen.

void agents_terminated(String[] agent_names) Bei einer Terminierung von Agenten, die durch das `NamingGateway` registriert wird, wird die Nachricht `agents_terminated()` geschickt. Ist eine Agent in der als Parameter übergebenen Liste durch das `VoyagerMasaGatewayApplet` terminiert worden, wird sein Name aus `_agentsCreatedByApplet` gelöscht und ihm bezüglich keine weiteren Aktionen durchgeführt. Wenn er jedoch außerhalb MASAs terminiert wurde, wird der `VoyagerAgentManager` damit beauftragt, den passenden `VoyagerProxyAgent` in MASA zu terminieren.

String [] list_all_agents() Um eine Liste aller Voyager Agenten zu erhalten, kann nicht die MASA Schnittstelle bemüht werden, da die Information über den Agentensystemtyp nicht übergeben werden kann. Der `VoyagerAgentManager` erbt jedoch die Methode `list_all_agents` vom `GenericAgentManager` und kann so eine Liste aller Voyager Agenten liefern. Diese `CfMAF.Name` Liste wird in einen String Array umgewandelt und kann dann an das `VoyagerMasaGatewayApplet` zurückgegeben werden.

String [] list_all_voyagerserver() Die Information über die konfigurierten Voyager Server erhält das `VoyagerMasaGatewayApplet` wiederum über diese Methode des `VoyagerMasaGatewayStationaryAgents`, der sich die Liste der Voyager Server vom `VoyagerAgentManager` beschafft.

String get_server(String agent_name) Die Methode `get_server()` leitet den Aufruf direkt an den `VoyagerAgentManager` weiter, der die Information bereithält, welcher Voyager Agent auf welchem Server aktiv ist.

void cleanUp() Zur Terminierung des `VoyagerMasaGatewayStationaryAgent` wird der `VoyagerAgentManager` angewiesen, seine Terminierung vorzubereiten und das Namenverzeichnis für Voyager Agenten in MASA gelöscht.

NamingGateway

Das NamingGateway, das durch den VoyagerMasaGatewayStationaryAgent gestartet wird, durchsucht den Namespace von Voyager nach Voyager-Agenten (d.h. es wird eigentlich nach Objekten gesucht, s.

Kurzbeschreibung Voyager Technologie). Falls ein neuer Agent entdeckt wurde, wird eine Nachricht an den VoyagerMasaGatewayStationaryAgent geschickt. Die Durchsuchung des Voyager Namespaces geschieht auf eine von Objectspace für Voyager nicht dokumentierte Weise, da Anfragen [TE] bei Objectspace nach einer Möglichkeit (s. Absatz "getVoyagerDirectory"), eine Liste aller im Namespace eingetragenen Objekte zu erhalten, negativ beantwortet wurden.

static String[] getDifference(String[] a, String []b) Bei `getDifference()` handelt es sich um eine kleine Hilfsmethode um die Differenz zweier Mengen A und B in Form zweier String Arrays zu berechnen. Das Ergebnis ist A/B falls B eine echte Teilmenge von A ist, ansonsten wird eine null-Referenz zurückgegeben.

IDirectory getVoyagerDirectory(String vurl) Mit `getVoyagerDirectory()` wird das Verzeichnis des in `vurl` angegebenen Voyager Server zurückgegeben. Dieses `IDirectory` Verzeichnis bildet den Kern des Voyager `Namespaces` und ermöglicht eine Auflistung aller enthaltenen Eintragungen.

String[] listAllVoyagerAgents(String vurl) Um den Zugriff auf das Voyager Verzeichnis abzuschirmen, wird die Methode `listAllVoyagerAgents()` verwendet. Sie startet einen Voyager Client und beendet diesen vor Aufruf von `getVoyagerDirectory()` ([Obje b], s.S.6). Die Liste der gefunden Voyager Agenten wird als String-Array zrückgeliefert.

VoyagerMasaGateway getVoyagerMasaGatewayStationaryAgent() Dieser Selektor dient als Hilfe bei der Initialisierung des `NamingGateways`. Um eine Referenz des `VoyagerMasaGatewayStationaryAgents` zu erhalten, wird mit Hilfe des im Rahmen der Interoperabilitätstest an der Orb-Orb-Schnittstelle erstellten `NamingWebServers` der von MASA verwandte `NamingContext` über IIOP referenziert und aus dem Verzeichnis der globalen Agenten der `VoyagerMasaGatewayStationaryAgent` ermittelt.

void start() In dieser Methode ist der eigentliche Mechanismus des `NamingGateways` implementiert. Durch einen Polling-Algorithmus wird der `Namespace` des Voyager alle fünf Sekunden [2] durchsucht, das jeweils aktuelle Suchergebnis mit dem vorhergehenden durch die Hilfsmethode `getDifference()` verglichen und Veränderungen dem `VoyagerMasaGatewayStationaryAgent` über dessen Methoden `agents_created()` und `agents_terminated()` mitgeteilt (s.Abb.4.6). Die Suche nach Voyager Agenten wird nicht nur für *einen* Voyager Server, sondern für die in `_vurls` gespeicherten Voyager Server durchgeführt.

static void main(String[] voyager_server) Um das `NamingGateway` zu starten, muß die `main()`-Methode der `NamingGateway`-Klasse aufgerufen werden und die Liste der Voyager Server, die überwacht werden sollen, übergeben werden. `main()` erzeugt zunächst eine Instanz des `NamingGateways` und startet es anschließend mit der `start()`-Methode. Die etwas umständliche Implementierung des `NamingGateways` via `main()`-Methode entstand aus dem Konzept, das `NamingGateway` unabhängig vom `VoyagerMasaGatewayStationaryAgent` zu realisieren und zu benutzen. Zur Steigerung der Benutzerfreundlichkeit wurde dies jedoch nicht vollzogen, sondern die Aktivierung des `NamingGateways` mit der Erzeugung des `VoyagerMasaGatewayStationaryAgents` vereint.

VoyagerAgentManager

Der `VoyagerAgentManager` ist eine Subklasse der abstrakten Klasse `GenericAgentManager`. `GenericAgentManager` enthält die kleinste Menge aller Methoden, die nötig ist, um eine Klasse als

[2] Der Wert wird im Quellcode durch eine Klassenkonstante gesetzt. Der verwendete Wert von 5 Sekunden beruht auf Erfahrung bezüglich einem verträglichen Maß zwischen Rechnerzeitverbrauch und temporärer Konsistenz der Daten über aktive Voyager Agenten

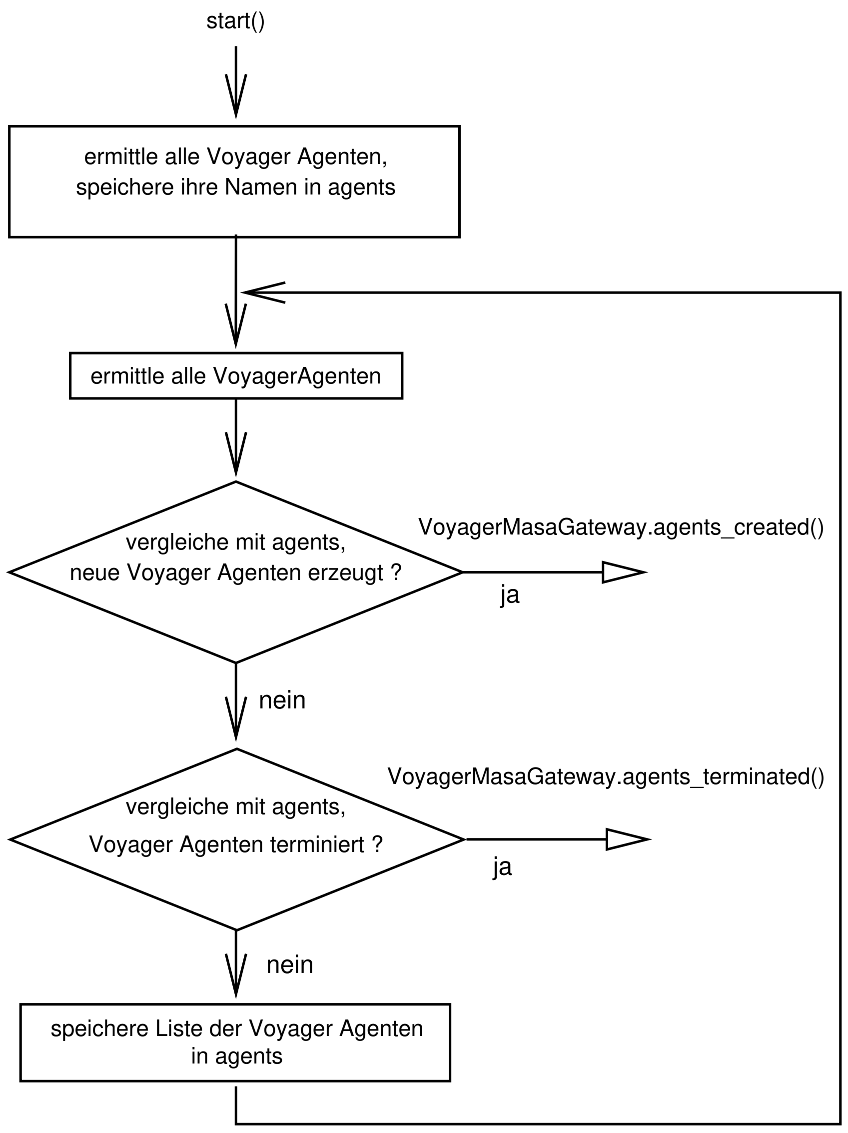

Abbildung 4.6: Funktionsweise des NamingGateways

Agent Manager in der Klasse `AgentSystem` zu verwenden. Davon sind für den `VoyagerAgentManager` hauptsächlich die Methoden `create_agent` und `terminate_agent` von Bedeutung, um eine Steuerung der Voyager-Agenten zu ermöglichen. Folgende Methoden, die der `VoyagerAgentManager` von `GenericAgentManager` erbt, werden zur Verwaltung und Steuerung der Voyager Agenten nicht benötigt und sind deshalb nicht implementiert (zu ihrer genaueren Beschreibung s. [Kemp 98]):

- `org.omg.CORBA.Object connectToAgent(Name agent_name)`
- `String getAgentSystemURL()`
- `AgentStatus get_agent_status(Name agent_name)`
- `AuthInfo get_authinfo(Name agent_name)`
- `CfMAF.Name[] list_all_agents_of_authority(byte [] authority)`
- `void receive_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, ClassName[] class_names, String code_base, MAFAgentSystem`

agent_sender)

- void resume_agent(Name agent_name)
- void suspend_agent(Name agent_name)
- void terminateAgentManager()
- void setAgentSystemService(AgentSystemService ass)
- void writeAgentToFile(CfMAF.Name agentName, java.lang.String file)
- CfMAF.Name readAgentFromFile(java.lang.String file)

Name create_agent(Name agent_name, AgentProfile agent_profile, byte[] agent, String place_name, byte[] arguments, ClassName[] class_names, String code_base, MAFAgentSystem class_provider) Die Methode `create_agent()` wird vom MASA System zur Erstellung eines Voyager Agenten aufgerufen. Von den übergebenen Parametern sind zur Erstellung eines Voyager Agenten folgende von Interesse: `agent_name`, `agent`, `place_name`, `class_names`. Der Name des Voyager Agenten soll die in `agent_name` übergebene Bezeichnung erhalten. Unter Beachtung der Randbedingung, daß für alle Voyager Agenten nur ein Namensraum existiert, wird zur Vermeidung der Namensgleichheit der gewünschte Name des Voyager Agenten folgendermaßen modifiziert: Ist ein Agent des gleichen Namens schon vorhanden wird dem Agentennamen das Postfix `<n>` hinzugefügt, wobei n standardmäßig mit 1 belegt ist und für jeden weiteren namensgleichen Agenten um 1 inkrementiert wird. Der Voyager Server, auf dem der Agent erstellt werden soll, wird in `place_name` angegeben. Die Java-Klasse, die die Implementation des Agenten realisiert, wird in `class_names` angegeben. Hat `class_names` die Länge 0, so wird die in `agent_name` angegebene Bezeichnung verwandt. Im ersten Schritt zur Erstellung eines Agenten wird der Packagename aus dem Parameter `agent` ausgelesen. Danach der zuvor beschriebene Mechanismus zur Ermittlung des Klassennamens durchgeführt und anschließend der Name des Agenten auf die ebenfalls zuvor beschriebene Art und Weise berechnet. Nun kann in MASA ein `VoyagerProxyAgent` zur Repräsentation des Voyager Agenten erstellt werden. Schlägt dies fehl, wird die `create_agent()`-Methode ohne Erstellung eines Voyager Agenten abgebrochen. Im fehlerfreien Fall wird mit der Erstellung des Voyager Agenten fortgesetzt. Dazu wird zunächst ein Voyager Client gestartet. Da parallel dazu in derselben Java-Virtual-Machine (JVM) im Thread des `NamingGateways` ein Voyager Client gestartet sein kann, kann der Versuch, durch den `VoyagerAgentManager` hier einen Voyager Client zu starten, mit dem Wurf einer `StartupException` fehlschlagen. Dies stellt jedoch keinen Fehler dar und wird nur durch die Ausgabe einer Warnmeldung angezeigt. Nach dem Start des Voyager Clients wird der Voyager Agent erzeugt und dann zum Voyager `Namespace` mit seinem Namen und dem Voyager Server samt Portnummer als Präfix gebunden. Die Eintragung in den Voyager `Namespace` kann zwar auch zusammen mit der Erstellung des Agenten mit einer Methode durchgeführt werden, doch durch die Aufteilung ist eine differenziertere Fehlererkennung möglich. Nach erfolgreicher Erstellung wird der Voyager Client wieder angehalten. Im Fehlerfall muß zunächst ein Rollback bezüglich der Erstellung des `VoyagerProxyAgents` erfolgen. Die jedoch einfache und konsistente Terminierung eines MASA Agenten rechtfertigt auch die Reihenfolge, in der zunächst der `VoyagerProxyAgent` und dann erst der Voyager Agent selbst erstellt werden. Als Rückgabewert wird der zur Vermeidung von Namensgleichkeit modifizierte Agentenname zruückgeliefert.

String execute_agent(String agent_name, String method, Object[] args) Die Möglichkeit eine Methode eines Agenten auszuführen, ist in MASA nicht realisiert und deshalb erbt der `VoyagerAgentManager` keine entsprechende Methode vom `GenericAgentManager`, sondern führt mit `execute_agent()` selbst eine neue Methode ein. Die Parameter sind bis auf die Liste der Argument dieselben wie in der `execute_agent()`-Methode des `VoyagerMasaGatewayStationaryAgents`. Die Argumente liegen jetzt schon als Objekte vor. Der Aufruf der Methode des Voyager Agenten erfolgt synchron, d.h. `execute_agent` blockiert, bis die Methode ausgeführt wurde. Die Blockierung der Programmflusses wirkt sich auf das `VoyagerMasaGatewayApplet` aus, das keine weiteren Benutzereingaben verarbeiten kann, bis ein Rückgabewert der Methode des Voyager Agenten zur Verfügung steht. Der Rückgabewert der Methode des Voyager Agenten wird in einen `String` umgewandelt und zrückgegeben.

void migrate_agent(String agent_name, String target, String execute, Object [] arguments) Die Migration eines Agenten erfolgt wie `execute_agent()` in einer neu eingeführten Methode. Die Parameter entsprechen bis auf die Liste der Argumente den in der `migrate_agent()`-Methode des `VoyagerMasaGatewayStationaryAgents` beschriebenen Parametern. Bei der Angabe des Voyager Servers auf den migriert wird muß beachtet werden, daß der Hostname mit der gesamten Domainbezeichnung und der Portnummer angegeben werden muß. Nach der Migration der Voyager Agenten wird die Lokaltransparenz von Voyager aufgehoben, indem der migrierte Agent dem `Namespace` des neuen Voyager Servers zugeordnet wird. Da bei einer Terminierung des Voyager Servers von dem wegmigriert wurde, die Referenz auf den Agenten verloren gehe würde, wird durch die Abbildung der Migration auf den `Namespace` verhindert.

void terminate_agent(Name agent_name) Bei der `terminate_agent()`-Methode handelt es sich um eine vom `GenericAgentManager` geerbte Methode und dient der Terminierung eines Voyager Agenten. Da die Terminierung eins Voyager Agenten durch den Garbage Collection Mechanismus erfolgt, wird in terminate_agent lediglich die Referenz im Namespace auf einen Voyager Agenten gelöscht, um dem Distributed Garbage Collector die Möglichkeit zu geben, den Voyager Agenten zu entfernen ([Obje b],s.S.12). Dazu ist der Start eines Voyager Clients nötig (zur damit verbundenen Problematik siehe Absatz "create_agent()").

void create_voyagerproxyagent(String agent_name, String place_name) Ein wichtiges Bestandteil des Konzepts Voyager Agenten in MASA abzubilden, ist die Darstellung der Voyager Agenten durch `VoyagerProxyAgenten`. Diese werden vom `VoyagerAgentManager` mit der Methode `create_voyagerproxyagent()` erzeugt. Zur Erzeugung eines `VoyagerProxyAgenten` müssen zunächst die zur Erzeugung eines MASA Agenten nötigen Parameter zusammengestellt werden. Da als Agentennamen mit `agent_name` auch Agentennamen übergeben werden, die vom `NamingGateway` als neu erstellte Voyager Agenten gemeldet wurden, müssen die Namen dahin überprüft werden, ob ihr Name auch Punkte enthält. Da Punkte in Namen von MASA Agenten nicht erlaubt sind, werden in den Namen der Voyager Agenten Punkte durch Unterstriche ersetzt, bevor diese als Namen für die `VoyagerProxyAgenten` dienen können. Nach Erzeugung eines `VoyagerProxyAgenten` wird dessen Name in das Namensverzeichnis für `VoyagerProxyAgenten` in MASA eingetragen. In MASA sind die `VoyagerProxyAgenten` dann sowohl im Namensverzeichnis ihres Agentensytems als auch im Namenverzeichnis für Voyager Agenten eingetragen. Nach erfolgreicher Erstellung des Voyager Agenten samt `VoyagerProxyAgent` in MASA wird der Name des Agenten in zwei Hashtables abgelegt: in `_agentVoyagerName`, in der jedem Namen eines Voyager Agenten die korrekte Namensbezeichnung nach Voyager zugeordnet wird, d.h. mit Servernamen und Portnummer als Präfix und mit dem korrekten Namen des Agenten inklusive eventueller Punkte als Postfix, und in `_voyagerProxy`, in der jedem Voyager Agenten der Name als `CfMAF.Name` des zugehörigen `VoyagerProxyAgenten` in MASA zugeordnet wird.

void terminate_voyagerproxyagent(String agent_name) Um einen VoyagerProxyAgent zu terminieren, wird zunächst die Nachricht `readyForTermination` an den Agenten geschickt, um anzuzeigen, daß er von dem `VoyagerAgentManager` terminiert wird. Ein Terminierung durch andere Komponenten, z.B. das AgentSystemApplet, ist nicht erlaubt. Nach der Terminierung durch das MASA System wird der Name des Agenten aus dem Voyager Namensverzeichnis in MASA ausgetragen und die Einträge in den Hashtables `_agentsVoyagerName` und `_voyagerProxy` gelöscht.

void cleanUp() Diese Nachricht wird vom `VoyagerMasaGatewayStationaryAgent` an den `VoyagerAgentManager` geschickt, um eine anschließende Terminierung anzukündigen. Der `VoyagerAgentManager` terminiert mit der `cleanUp()`-Methode alle `VoyagerProxyAgenten`.

CfMAF.Name[] list_all_agents() Die Liste aller Voyager Agenten wird aus der Hashtable `_voyagerProxy` ausgelesen.

String get_server(String agent_name) Um den Voyager Server eines Voyager Agenten zu ermitteln wird aus der Tabelle `_agentVoyagerName` der volle Name, der als Präfix den Voyager Server enthält, eines Voyager Agenten ermittelt und das beschriebene Präfix zurückgegeben.

VoyagerProxyAgentMobileAgent

Die Aufgabe des `VoyagerProxyAgenten`, der als mobiler MASA Agent realisiert ist, besteht darin, Voyager Agenten in MASA zu repräsentieren und Anfragen an ihr zugehöriges Java-Applet an das Applet des `VoyagerMasaGatewayStationaryAgent` um zu leiten.

void run() Die Umlenkung des Applets wird durch Modifizierung des Attributs `_url` realisiert, das von der Methode `getURL()` als Adresse der HTML-Seite zurückgegeben wird, die das Applet enthält. Diese Modifikation erfolgt in der `run()-Methode` des `VoyagerProxyAgentMobileAgents`.Dazu wird der letzte Teil von `_url`, "VoyagerProxyAgent.html", durch "VoyagerMasaGateway.html" ersetzt.

VoyagerMasaGatewayApplet

Das `VoyagerMasaGatewayApplet` dient zur Steuerung der Voyager Agenten:

- Erzeugung von Voyager Agenten
- Ausführung von Methoden eines Voyager Agenten
- Migrierung eines Voyager Agenten
- Terminierung eines Voyager Agenten

Durch das Applet werden die nötigen Benutzereingaben ermöglicht und an den `VoyagerMasaGatewayStationaryAgent` geschickt. Die Oberfläche ist als Java-Applet, das aus Komponenten des Swing-Packages besteht, in eine HTML-Seite eingebettet. Die Klasse `VoyagerMasaGatewayApplet` ist eine Unterklasse von `AgentApplet`, die als Oberklasse für die Applets aller MASA Agenten fungiert und selbst eine Unterklasse von `JApplet` ist [Kemp 98].

void init() Diese Methode wird beim Laden des Applets durch den Browser aufgerufen. Sie beschafft eine Referenz des `VoyagerMasaGatewayStationaryAgenten` über CORBA, die in `_voyagerMasaGateway` gespeichert wird, und initialisiert die Oberfläche mit den Methoden `init_gui()` und `update_gui()`.

void init_gui() Mit der Methode `init_gui()` werden die Elemente, wie Buttons, ComboBoxes, Panels, etc. zur Oberfläche des Applets dazu gefügt und ihr Layout initialisiert.

void update_gui() Die Belegung der Elemente, hauptsächlich der ComboBoxen, mit Werten, wird mit `update_gui()` durchgeführt, das die Liste der Voyager Agenten bzw. Voyager Server von `_voyagerMasaGateway` erhält. Außerdem wird die Methode `set_buttons()` ausgeführt.

void set_buttons() Regelt die Aktivierung bzw. Deaktivierung der Buttons in Abhängigkeit der Eingabebedingungen. So wird zum Beispiel der Button zur Erstellung eines Voyager Agenten deaktiviert, falls der Agentenname nicht eingegeben ist.

Event-Handling Das Event-Handling wird hauptsächlich durch den `AppletActionListener` durchgeführt. Die jeweiligen Aufrufe zur Steuerung der Agenten werden mit den Benutzereingaben als Parameter an `_voyagerMasaGateway` geschickt und deren Rückgabewerte als Ausgabe in einem Textfeld an der Oberfläche dem Benutzer angezeigt. Bei fehlerhafter Ausführung wird die `Exception` abgefangen und eine Fehlermeldung ebenfalls auf dem zuvor erwähnten Textfeld ausgegeben. Nach jedem Ereignis wird `update_gui()` aufgerufen, um die veränderten Daten auf die Oberfläche zu übertragen. Bei Veränderung der Eingabefelder, die eine Aktivierung bzw. Deaktivierung der Buttons haben sollen, wird der `AppletCaretListener` aktiv und setzt mit `set_buttons()` die Zustände der Buttons neu.

4.4.3 Testagent

Das Interface `FOO` dient als Schnittstelle für den Voyager Agenten `FOOVoyagerAgent`, der als Testagent benutzt wird. Für Voyager Agenten ist es nicht nötig ein IDL-Interface zu erstellen, es genügt ein Java-Interface oder sogar das Classfile der Implementierung, aus dem mit dem Voyager Utility *igen* ein Interface erstellt werden kann ([Obje b], s.S.104).

4.4.4 Implementierung des Testagenten

Als Testagent für das `VoyagerMasaGateway` fungiert der `FOOVoyagerAgent`. Er ermöglicht die Ausführung verschiedener Methodentypen.

void message() Mit dieser Methode wir lediglich die Ausführung einer Agentenmethode getestet. Bei erfolgreicher Ausführung wird auf der Standardausgabe des Voyager Servers, auf dem der Agent läuft, “This is a message!” ausgegeben.

void print_message(String message) Mit dieser Methode kann die Übergabe von Parametern getestet werden. Der übergebene String wird auf der Standardausgabe des Voyager Servers, auf dem der Agent läuft, ausgegeben.

void print_number_message(Short number, String message) Der `print_number_message()`-Methode kann zusätzlich eine Zahl vom Typ `Short` übergeben werden. Beide Parameter werden wie zuvor beschrieben ausgegeben.

String get_time() Die Rückgabe von Parametern wird mit `get_time()` getestet. `get_time()` liefert den Hostname und die Systemzeit des Voyager Servers seines Agenten.

4.5 Installation des VoyagerMasaGateways

Das VoyagerMasaGateway ist im cvs-Repository unter Masa/VoyagerMasaGateway abgelegt und wird mit dem Befehl “cvs -d *cvsroot* checkout Masa/VoyagerMasaGateway” ausgecheckt (zur Konfiguration s. unten). Die Kompilierung und Installation in MASA erfolgt mit den Befehlen:

1. `gmake proddir`
2. `gmake`
3. `gmake install`

4.6 Benutzung des VoyagerMasaGateways

Zur Nutzung des VoyagerMasaGateways muß der VoyagerMasaGatewayStationaryAgent erzeugt werden und die konfigurierten Voyager Server gestartet werden.

4.6.1 Konfiguration des VoyagerMasaGatewayStationaryAgent

Zur Konfiguration des VoyagerMasaGatewayStationaryAgent sind in der Datei "Makefile.DEF" des VoyagerMasaGateways folgende Parameter zu setzen:

- de.unimuenchen.informatik.mnm.masa.voyager.server=*host:port[,host:port,...]* Dieser Parameter bestimmt die Voyager Server, zu denen eine Verbindung über das *VoyagerMasaGateway* aufgebaut werden soll. Standardwert:"sunhegering10:8000,sunhegering2:8000"

- de.unimuenchen.informatik.mnm.masa.voyager.type=*nummer* Dieser Parameter legt den Agentensystemtyp für Voyager in MASA fest. In MASA wird ein Agentensystem anhand seines Agentensystemtyps identifiziert. Der Agentensystemtyp ist als Nummer kodiert.Standardwert für Voyager: "5"

4.6.2 Starten des VoyagerMasaGatewayStationaryAgent

Der VoyagerMasaGatewayStationaryAgent wird vom Benutzer durch das AgentSystemApplet eines MASA Agentensystems gestartet. Der Paramater *exclusive* muß auf "true" gesetzt werden, ansonsten wird der VoyagerMasaGatewayStationaryAgent nach seinem Start "suspended" gesetzt. Für den Konstruktor des VoyagerMasaGatewayStationaryAgent sind keine Parameter nötig.

4.6.3 Starten eines Voyager Servers

Zum Starten eines Voyager Servers steht im `Masa/system/install/bin`-Verzeichnis das Skript "voyagerScript.*os* zur Verfügung. Als Parameter muß die Portnummer des gewünschten Servers übergeben werden.

4.6.4 Steuerung der Voyager Agenten durch das VoyagerMasaGatewayApplet

Das VoyagerMasaGatewayApplet als Hompage des VoyagerMasaGatewayStationaryAgent (dieser Agent muß, wie oben beschrieben, zuerst erzeugt werden) kann vom Benutzer durch das AgentSystemApplet geladen werden. Die vier Aktionen zur Steuerung eines Voyager Agenten (Erzeugung, Ausführung einer Methode, Migration, Terminierung) werden im Applet auf vier Karteikarten abgebildet. Auf einer zusätzlichen Karteikarte wird der Voyager Server eines Agenten angezeigt.

Erzeugung eines Voyager Agenten (create) (s. Abb.4.7) Im Feld Agentenname wird der Name des Agenten angegeben. Im Feld Classname der Klassenname der Implementierungsklasse des Agenten. Wurde im Feld Agentenname der Klassenname bereits als Agentenname eingetragen, kann das Feld Klassenname leer bleiben. Im Feld Package wird das Package der Agentenklasse angegeben. Der Voyager Server für den Agenten kann über die ComboBox Voyager Server ausgewählt werden. Der Button "create", der zur Ausführung angeklickt werden muß, wird erst aktiviert, wenn ein Agentenname eingegeben wurde. Im Ausgabefeld "output" wird die erfolgreiche Erzeugung des Agenten und die Modifikation seines Namens bei Namensgleichheit mit bestehenden Agenten angezeigt, ebenso ein eventuelle Fehlermeldung.

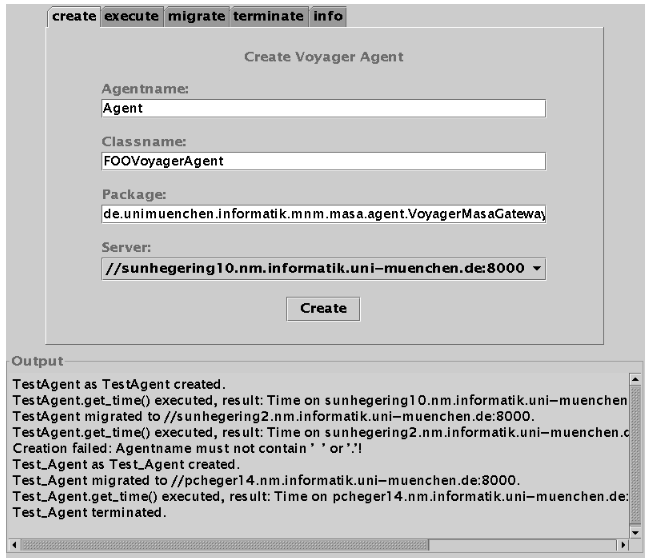

Abbildung 4.7: Karteikarte zur Erzeugung eines Voyager Agenten

Ausführung einer Voyager Agenten Methode (execute) (s. Abb.4.8) Zur Ausführung einer Methode eines Agenten muß zunächst ein aktiver Agent ausgewählt werden. Im Feld Method muß der Methodenname eingetragen werden und im Feld Arguments die Parameter bestimmt werden, wobei deren Typ als Java-Klasse mitangegeben werden muß (s. Absatz "VoyagerMasaGatewayStationaryAgent", Paragraph "execute_agent()"):

$$
\begin{array}{rcl}
\text{Liste der Argumente} & ::= & Argument[',' Argument] \\
Argument & ::= & Package.Classname \sqcup Wert \\
Package.Classname & ::= & \text{Klassenname mit Packageangabe des Arguments} \\
Wert & ::= & \text{Wert des Arguments}
\end{array}
$$

Beispiel: "java.lang.String Hello World!,java.lang.Integer 3"
Bestätigt zur Ausführung werden die Angeben durch klicken des Knopfes "execute". Der Rückgabewert der Methode wird als String in "output" angezeigt; liefert die Methode null oder ist als *void* deklariert, dann wird "void" ausgegeben.

Migration eines Voyager Agenten (migrate) (s. Abb.4.9) Zur Migration eines Agenten muß sein Name und der Voyager Server, auf den der Agent migriert werden soll, ausgewählt werden und mit Klicken des Knopfes "migrate" die Durchführung veranlaßt werden. In Voyager ist es möglich eine Methode zur Ausführung nach der Migration anzugeben. Zur Beschreibung der nötigen Angaben siehe oben: "Ausführung einer Agenten Methode". Zu beachten ist allerdings, daß nach einer Ausführung hier kein Rückgabewert der Methode vorliegt. Die Bestätigung bzw. Fehleranzeige erfolgt im Feld "output".

Terminierung eines Voyager Agenten (terminate) (s. Abb.4.10) Die Terminierung eines Voyager Agenten wird durch Auswahl des Agentennamen und Klicken des Knopfes "terminate". Die Bestätigung bzw. Fehleranzeige erfolgt im Feld "output".

Auf welchem Voyager Server läuft ein Agent? (info) (s. Abb.4.11) Die Information erhält man auf der letzten Karteikarte. Dort wird nach Auswahl eines Agenten unter der Bezeichnung *is running*

Abbildung 4.8: Karteikarte zur Ausführung einer Methode des Agenten

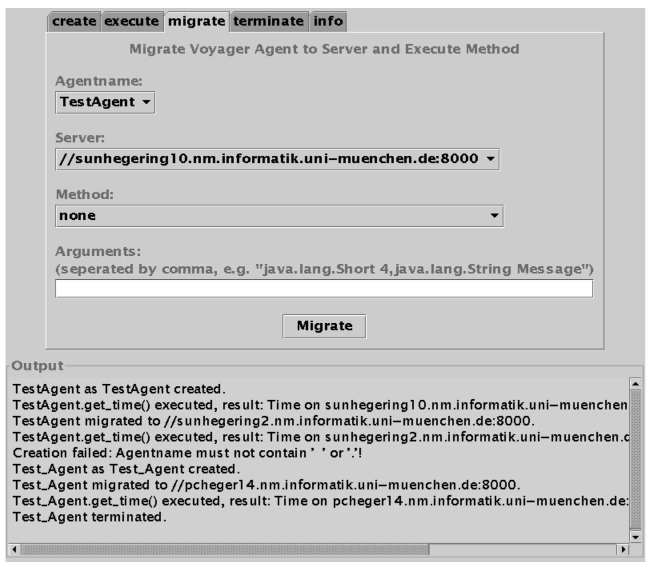

Abbildung 4.9: Karteikarte zur Migrierung eines Agenten

on der Name des Voyager Servers angegeben, auf dem der Agent aktiv ist. Im Feld “output” erfolgt diesmal keine Ausgabe.

4.7 Zusammenfassung

Aus der Beschäftigung mit MASA und der Voyager Technologie im Vergleich ergeben sich knapp gefaßt folgende Bewertungen:

- Die Erstellung von Agenten mit der Voyager Technologie ist äußerst effizient, da sogar aus bestehenden Java-Klasse, die nur im Byetcode vorliegen, Agenten erstellt werden können.
- Die Steuerung und Verwaltung von Agenten ist aufgrund der MASA zugrunde liegenden *MASIF*-Spezifikation jedoch konsequenter durchdacht.

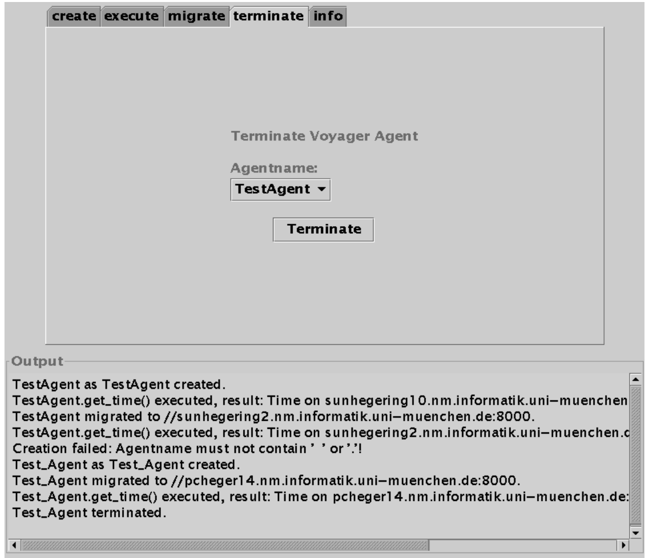

Abbildung 4.10: Karteikarte zur Terminierung eines Agenten

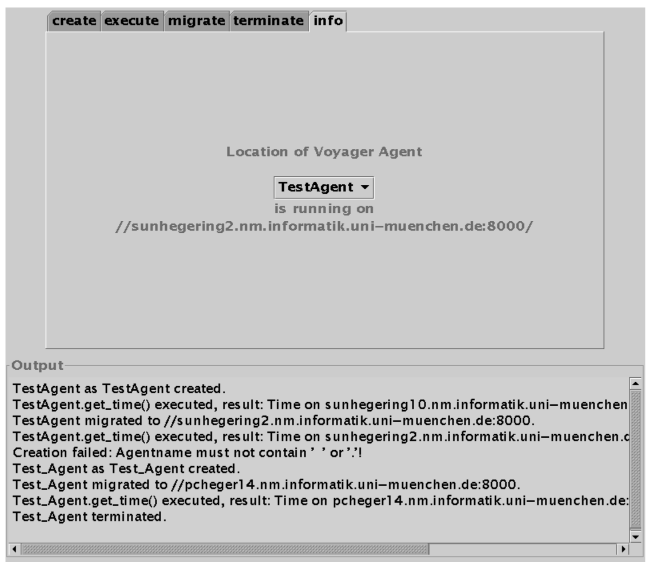

Abbildung 4.11: Karteikarte zur Anzeige des Voyager Servers eines Agenten

- Die Erweiterbarkeit des Systems gestaltet sich in MASA einfacher, da als Kommunikationsarchitektur keine proprietäre Entwicklung wie in Voyager verwendet wurde, sondern die grundsätzlich standardisiert CORBA Technologie.

- Das Sicherheitskonzept von Voyager beruth auf dem Java-SecutiyManager Konzept ([Obje b]s.S.59) und führt bei der Voyager Standardinstallation keine Authentifzierung der Remote Objekte bezüglich

- Ein weiterer Kritikpunkt bezüglich Voyager ist die magere Dokumentation, die keinerlei Beschreibung der Voyager zugrunde liegenden Architektur enthält.

Bei dieser Bewertung darf jedoch nicht die unterschiedliche Auslegung MASAs, als Agentensystem, im Vergleich zu Voyager, als Agentenframework, vergessen werden. Im folgenden Ausblick sollen Möglichkeiten aufgezeigt werden zur Ausnutzung der Vorteile von MASA als Agentensystem und Voyager als "Baukasten" für Agenten durch das *VoyagerMasaGateway*.

Kapitel 5

Ausblick

5.1 Interoperabilität des CORBA Naming Service

Im Rahmen der derzeit laufenden Diplomarbeit von Harald Rölle wird MASA um die Einsatzmöglichkeit mit dem CORBA-ORB Orbacus [Obje a] erweitert. Diese Erweiterung betrifft auch den NamingWebServer (s. Abschnitt 3.1.2), der nun zusätzlich, neben den Naming Services von VisiBroker und JavaSofts Java IDL, den Naming Service von Orbacus unterstützen soll. Das bedeutet eine weitere Parametrisierung des Codes bezüglich des verwendeten NamingContexts und macht eine Präprozessor gestützte Parametrisierung des Codes sinnvoll. Zur Interoperabilität des NamingContexts liegt bei der OMG eine Submission vor [OMG a], in der Bootstrappingverfahren zur Benutzung eines gemeinsamen NamingServices beschrieben werden. Dazu werden neben der Refenzierung durch den IOR-String alternative URL Formate (iioploc, iiopname) vorgeschlagen, so daß die Referenz auf das NamingService Objekt vom Programm selbst berechnet werden kann und auf die Abfrage der IOR verzichtet werden kann.

5.2 Erweiterungen des Voyager-MASA-Gateways

Viel umfangreicher Möglichkeiten zur Optimierung und Erweiterung als im Bereich des interoperablen Naming Service bietet die Voyager Technologie im Zusammenhang mit MASA.

5.2.1 Optimierung des VoyagerMasaGateways durch Einsatz des Notification Service

In der aktuellen Implementierung des *VoyagerMasaGateways* führt die Terminierung eines *VoyagerProxyAgents* durch das *AgentSystemApplet* zu einer Instabilisierung des *VoyagerMasaGateways*, da diesem die Terminierung eines *VoyagerProxyAgents* nicht gemeldet wird. Durch Einsatz des derzeit im Rahmen eines Systementwicklungsprojekts in Entwicklung befindlichen *Notification Services* könnte diese Schwachstelle beseitigt werden.

5.2.2 MASA Agentensystem auf Basis der Voyager Technologie

Neben der Optimierung sind Erweiterungen für das *VoyagerMasaGateway* denkbar. Das oben beschriebene *VoyagerMasaGateway* realisiert nur eine Anbindung von Voyager Agenten an MASA. Die Möglichkeiten zur Ausnutzung ihrer Funktionalitäten ist mit der Ausführung ihrer Methoden über das zuvor beschriebene Applet zwar möglich aber relativ beschränkt. Eine sinnvolle Erweiterung wäre, neben der Implementierung zusätzlicher funktional sinnvoller Voyager Agenten, die Referenzierung eigener Applets für Voyager Agenten über seinen jeweiligen *VoyagerProxyAgenten* in MASA, wodurch die Funktionalität von implemenetierten Voyager Agenten ausgenutzt werden könnte. (s. Abb. 5.1)

Im nächsten Schritt könnte die Realisierung eines Agentensystems unter Einsatz der Voyager Technologie mit den Schnittstellenspezifikationen von MASA stehen. (s. Abb. 5.2)

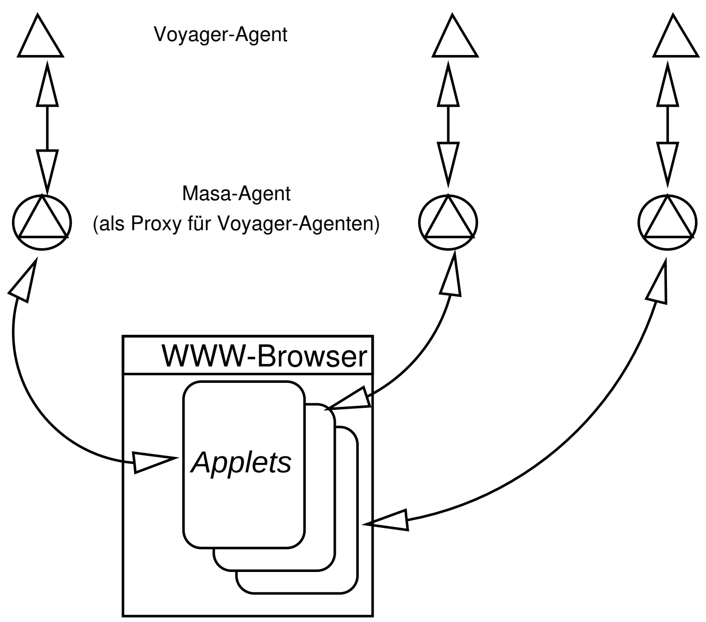

Abbildung 5.1: Eigenes Applet für jeden Voyager Agenten

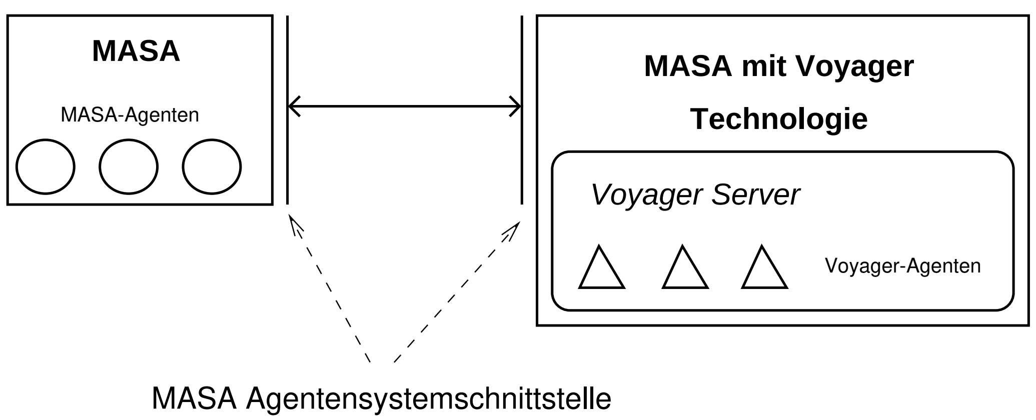

Abbildung 5.2: Agentensystem basierend auf Voyager Technologie

Anhang A

E-Mail Korrespondez mit Objectspace

A.1 E-Mail von Objectspace über list()-Funktion in Voyager

```
...
To answer your question:
Is there a method at the Namespace's api to get a list of all objects
bound to the Namespace or how can I get the list by some other way ?

I have been informed by our technical support department
that, unfortunately, there is no API that provides a list
of objects bound to the namespace.  To do this you will be
required to write your own code that keeps track of objects
added to and removed from the namespace.
...
```

A.2 E-Mail von Objectspace über Implementierung einer list()-Funktion in Voyager

```
...
To answer your question:
Does objectspace plan to enhance the voyager's api in the next version
of voyager for a method to get a list of all objects bound to the
namespace ?

I have consulted with the Voyager developers and they
have informed me that there are no immediate plans to
include this option, but that they will add it to their
list of requested features for consideration.  If the
decision is made to include it in the next release, I
will be sure to send you and email notification.
...
Regards,
Tina Eerligh
Business Development Executive
ObjectSpace, Inc.
www.ObjectSpace.com
Phone: 1.972.726.4206
Fax: 1.972.715.9006
```

Anhang B

Source-Code des Voyager-MASA-Gateways

B.1 NamingWebServer

```
package de.unimuenchen.informatik.mnm.masa.agentSystem.nameservice;

import java.io.*;
import java.net.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

/**
 * Class NamingWebServer implements a startroutine for
 * a CORBA-NameService and WWW-server to replay the IOR
 * of the NamingContext-Object of the CORBA-NameService.
 * So a the NameService of a special ORB can be used by another ORB.
 *
 */

public class NamingWebServer{
private String factory_name;          // Factory's name, necessary for VisiBroker's Naming Service
private String logfile;               // Name of the logfile, necessary for VisiBroker's Naming Service
private Thread thr_vb_nameservice;    // Thread of the NameService (VisiBroker)
private Process proc_jdk_nameservice; // Process of the NameService (JDK1.2)
private String ior;                   // IOR-String
private String nameORB;               // ORB name
private int port;                     // Port of WWW-Server
private String host;                  // Hostname


/**
 * Method to get the IOR from the NamingWebServer
 *
 * @return IOR of the NamingContext belonging to the NamingService started by the NamingWebServer
 *
 */
public static String getIORWWW() throws Exception
```

```
    {
String ior = null;
try{
// get host
String host = System.getProperty("namingwebserver.host");
// get portnumber
System.out.print("NamingWebServer.getIORWWW() get port...");
int port = new Integer (System.getProperty("namingwebserver.port")).intValue();
System.out.println("done.");

// Request server for IOR
URL url_name = new URL ("http",host,port,"/");
URLConnection con_name = url_name.openConnection();
InputStream in = con_name.getInputStream();
byte [] buffer = new byte[in.available()];
in.read(buffer);
ior = new String(buffer);
in.close();

// Extract IOR from HTTP-File
ior = ior.substring(ior.indexOf("IOR:"),ior.length());
System.out.println("NamingWebServer.getIORWWW(): " + ior);
} catch (Exception e){
e.printStackTrace();
throw e;
}
return ior;
    }


/**
 *  Function to extract the pur IOR from a string
 *
 *  @param ior the string sequenze containing the IOR
 *  @return the pur IOR
 */

private static String scanIOR(String ior){
int begin = ior.indexOf("IOR:");
String result = "IOR:";
int i=begin+4;
while(i<ior.length() &&
((47<ior.charAt(i) && ior.charAt(i)<58) ||
 (96<ior.charAt(i) && ior.charAt(i)<103))){ //is hex number
result += ior.substring(i,i+1);
i++;
}//while
return result;
}
```

```
/**
 * Constructor
 */
private NamingWebServer(String factory_name, String logfile) throws Exception
    {

// Check property
nameORB = System.getProperty("orb.name");
host = System.getProperty("namingwebserver.host");
port = new Integer(System.getProperty("namingwebserver.port")).intValue();
if (nameORB == null || (!nameORB.equals("VisiBroker") &
!nameORB.equals("JDK1.2")))
throw new Exception ("No or wrong ORB name specified.\n " +
 "Valid ORB's: VisiBroker, JDK1.2");
// Check arguments
if(nameORB.equals("VisiBroker")){
if (factory_name==null)
throw new Exception("No factory name specified!");
else this.factory_name = factory_name;
if(logfile==null)
throw new Exception("No logfile specified!");
else this.logfile = logfile;
}

System.out.println("NamingWebServer.<init>: ORB name=" + nameORB);

    }

/**
 * Wait for request and send as answer of HTTP-requests
 * IOR of the NamingContext
 */
private void waitForRequests() throws Exception {
// check if localhost host for namingwebserver
System.out.println("NamingWebServer.waitForRequests(): localhost:" +
 InetAddress.getLocalHost() + "\n"+
 "                                  configured host: " +
 host);
if(!InetAddress.getLocalHost().equals(InetAddress.getByName(host))){
throw new Exception("NamingWebServer must be started on " + host +
" or the scripts must be re-compiled on localhost/"+
InetAddress.getLocalHost() + "!");
}


// Create Serversocket
ServerSocket acceptSocket = new ServerSocket(port);
while (true){
System.out.println("NamingWebServer.waitForRequests(): Wait for requests, listening at port " + port + ".");
// Accept client, wait for requests
```

```
Socket socket = acceptSocket.accept();
// Create inputstream
InputStream in = socket.getInputStream();
// Wait for request-input
java.lang.Object a = new java.lang.Object();
synchronized (a) {while(in.available() <= 0) a.wait(10);}
System.out.println("NamingWebServer.waitForRequests(): Request:");
byte [] buffer = new byte[in.available()];
in.read(buffer);
System.out.print(new String(buffer));

// Test whether nameservice still alive
if (thr_vb_nameservice != null && !thr_vb_nameservice.isAlive())
throw new  Exception("NameService down!");
// Create outputstream, send ior over http
OutputStream out = socket.getOutputStream();
String answer = new String ("HTTP/1.0 200 OK \r\n" +
"Server: NamingWebServer \r\r" +
"Content_type: text/plain \r\n \r\n" + ior );
out.write(answer.getBytes());
out.flush();
in.close();
out.close();
socket.close();
}
}


/**
 * initWebServer() starts the NameService and gets the IOR from it
 */
private void initWebServer()throws Exception{
OutputStream out = System.out;   // dummy initialisation, just to satisfy the compiler
InputStream in = System.in;
PrintStream stdout = System.out;;

if (nameORB.equals("VisiBroker")){
///////////////////////////////////////////
// VisiBroker specific BEGIN
// Create PipedInput- and OutputStreams for threadcommunication
out = new PipedOutputStream();
in = new PipedInputStream((PipedOutputStream) out);
// Redirect stdout to Outputstream, stdout saved
PrintStream ptrout = new PrintStream(out,true);
System.setOut(ptrout);
// create thread for nameservice
thr_vb_nameservice = new Thread(new ThreadNameService());
// Start nameService
thr_vb_nameservice.start();
//VisiBroker specific END
///////////////////////////////////////////
```

```
} else if (nameORB.equals("JDK1.2")){

//////////////////////////////////////////
// JDK specific BEGIN
// start NameService
System.out.println("NamingWebServer.initWebServer(): Start tnameserv in  runtime.");
Runtime runt_nameservice = Runtime.getRuntime();
String [] cmd = {"/proj/java/jdk1.2-solaris/jdk1.2beta4/bin/tnameserv",
 "-ORBInitialPort","14050"};
proc_jdk_nameservice = runt_nameservice.exec(cmd);
// Get I/O-Stream of NameService-Process
out = proc_jdk_nameservice.getOutputStream();
in = proc_jdk_nameservice.getInputStream();
System.out.println("NamingWebServer.initWebServer(): NameService starting.");
// JDK specific END
//////////////////////////////////////////
}

// Wait for Nameservice output (=IOR-String)
while(in.available() <=0) {
java.lang.Object a = new java.lang.Object();
// artifical deadlock for waiting,
// i.e. nameservice has chance to write in stream
synchronized (a) {a.wait(1);
}
}
// Read IOR-String
byte [] buffer = new byte[in.available()];
in.read(buffer);
// Close streams
out.close();
in.close();

// extract the IOR from the read string
String iorExtFact = scanIOR(new String(buffer));

if (nameORB.equals("VisiBroker")){
//////////////////////////////////////////
// VisiBroker specific BEGIN
// Reset stdout
System.setOut(stdout);
System.out.println("NamingWebServerinitWebServer(): " + iorExtFact);
// Get IOR of NameService,
// The IOR returned by the NameService is
// the IOR of the ExtendedNamingContextFactory
ORB orb = ORB.init();
org.omg.CORBA.Object refNameContFact =
orb.string_to_object(iorExtFact);
ExtendedNamingContextFactory nameContFact =
ExtendedNamingContextFactoryHelper.narrow(refNameContFact);
NamingContext nameService = nameContFact.root_context();
```

```
ior = orb.object_to_string(nameService);
System.out.println("NamingWebServerinitWebServer(): ior length" + ior.length());
//VisiBroker specific END
///////////////////////////////////////////
}else if(nameORB.equals("JDK1.2")){
///////////////////////////////////////////
// JDK specific BEGIN
System.out.println("NamingWebServerinitWebServer(): " + iorExtFact);
ior  = iorExtFact;
System.out.println("NamingWebServerinitWebServer(): ior length" + ior.length());
// JDK specific END
///////////////////////////////////////////
}

}

/**
 * Stops the NamingWebServer
 */
private void stopWebServer()
    {
// Stop NameService
if (thr_vb_nameservice != null && thr_vb_nameservice.isAlive()) {
System.out.println("NamingWebServer.stopWebServer(): Stop NameService (VisiBorker).");
thr_vb_nameservice.interrupt();
} else if (proc_jdk_nameservice != null){
System.out.println("NamingWebServer.stopWebServer(): Stop NameService (JDK1.2).");
proc_jdk_nameservice.destroy();
}
    }

/**
 * Destruktor
 */
protected void finalize()
    {
stopWebServer();
    }

//////////////////////////////////////////////
// Main
/////////////////////////////////////////////
public static void main (String[] args) throws Exception {

NamingWebServer nws = null;
try {
nws = new NamingWebServer(args[0],args[1]);
nws.initWebServer();
nws.waitForRequests();
} catch (Exception e){
if (nws != null) nws.stopWebServer();
```

```
e.printStackTrace();
}
}

/**
 * Thread of the NamingService of VisiBroker 3.0
 */
private class ThreadNameService  implements Runnable {

public void run(){

////////////////////////////////////////
// VisiBroker specific BEGIN
// Prepare arguments for namegservice
String[] args_naming = {factory_name,logfile};
// nameservice VisiBroker in an own thread
com.visigenic.vbroker.services.CosNaming.ExtFactory.main(args_naming);
// VisiBroker specific END
///////////////////////////////////////

}
}
}
```

B.2 VoyagerMasaGatewayStationaryAgent

```
package de.unimuenchen.informatik.mnm.masa.agent.voyagermasagateway;

// MASA
import de.unimuenchen.informatik.mnm.masa.agentSystem.voyagerintegration.*;
import de.unimuenchen.informatik.mnm.masa.agentSystem.*;
import de.unimuenchen.informatik.mnm.masa.agent.*;
import CfMAF.*;
import de.unimuenchen.informatik.mnm.masa.tools.*;

// Voyager
import com.objectspace.voyager.*;

// VisiBroker
import org.omg.CosNaming.*;

// JDK
import java.lang.reflect.*;
import java.net.*;
import java.util.*;

/**
 *  Gateway between Voyager-Agentsystem and MASA-Agentsystem.
 *  Creates and registers a VoyagerAgentManager at startup at the MASA-System
 *  and forwards messages to the MASA-System to create, migrate and terminate
```

```
 *  Voyager-Agents.
 *  Starts the NamingGateway and receives messages from it about
 *  created or terminated agents
 *  Accepts messages from the VoyagerNamingGateway about created or terminated
 *  Voyager-Agents.
 */

public class VoyagerMasaGatewayStationaryAgent extends StationaryAgent
  implements VoyagerMasaGatewayOperations {
/**
 * CfMAF.Type of the Voyager-Agentsystem
 */
private Short VOYAGER_TYPE = new Short (System.getProperty("de.unimuenchen.informatik.mnm.masa.voyager.type","5"));
/**
 * VoyagerAgentManager, registered at MASA AgentSystem
 */
private VoyagerAgentManager _voyagerAgentManager;


/**
 * Voyager agents created by myself,
 * the message agents_created/agents_terminated
 * about this agents will be ignorred
 */
private Vector _agentsCreatedByApplet = new Vector();
private Vector _agentsTerminatedByApplet = new Vector();

/**
 * Voyager server
 */
private String [] _voyagerServer;

/**
 * Thread of the NamingGateway
 */
private Thread _namingGatewayThread;

/**
 * Constructor, initialize ?voyagerServer by exntending hostnames with full domainname
 */
public VoyagerMasaGatewayStationaryAgent(){
// get voyager server
StringTokenizer server = new StringTokenizer(System.getProperty("de.unimuenchen.informatik.mnm.masa.voyager.server","sunhegering10:8000,sunhegering2:8000"),",");
_voyagerServer = new String [server.countTokens()];
for(int i=0; i<_voyagerServer.length && server.hasMoreTokens(); i++){
String vserv = server.nextToken();
// expand name to fulll domain
try{
// get Internet address
InetAddress iAddress = InetAddress.getByName(vserv.substring(0,vserv.indexOf(":")));
// get ip number
```

```
String ip = iAddress.getHostAddress();
// build new internet address with full domain out of ip number
String fullAddress = InetAddress.getByName(ip).toString();
_voyagerServer[i] = "//" + fullAddress.substring(0,fullAddress.indexOf("/"));
_voyagerServer[i] += vserv.substring(vserv.indexOf(":"),vserv.length());
} catch(UnknownHostException unknownHost){
unknownHost.printStackTrace();
Debug.debug("VoyagerMasaGatewayStationaryAgent.<init> - " + vserv.substring(0,vserv.indexOf(":")-1));
_voyagerServer[i] = "";
}
Debug.debug("VoyagerMasaGatewayStationaryAgent.<init> - voyagerServer: " + _voyagerServer[i]);
}
}


/**
 * Stops voyager client via in _voyagerAgentManager and unbinds voyager namingcontext
 * @see Agent.cleanUp
 */
public void cleanUp()
    {
Debug.debug("VoyagerMasaGatewayStationaryAgent.cleanUp()");
try{
// terminate voyager prox agents
_voyagerAgentManager.cleanUp();

// unbind voyager namingcontext in masa
NameComponent [] path_voyager = new NameComponent [] { new NameComponent("Agent",""),
 new NameComponent("voyager", "")};
_initContext.unbind(path_voyager);
} catch (Exception e){
e.printStackTrace();
}

    }

/**
 * @see Agent.checkSerialization
 */
public void checkSerialization() throws CouldNotMigrate
    {
Debug.debug("VoyagerMasaGatewayStationaryAgent.checkSerialization()");
    }

/**
 * Starts the NamingGateway and registers VozagerAgentManager at AgentSystemService,
 * i.e. sends the message registerAgentManager to AgentSystemService
 *
 * @see Agent.run
 */
public void run()
```

```
    {
Debug.debug("\n\n\nVoyagerMasaGatewayStationaryAgent.run()\n\n\n");
// Test whether global
if (!_isGlobal){
try{
Debug.debug("\n\n\nVoyagerMasaGatewayStationaryAgent must be started global/exclusive\n\n\n");
// getAgentManager().terminate_agent(getAgentName());
  setAgentStatus(CfMAF.AgentStatus._CfMAFSuspended);
}catch(Exception e){
e.printStackTrace();
}
return;
}

// create namingcontext in masa for voyager
NamingContext voyagerContext = null;
try{
NameComponent [] path_voyager = new NameComponent [] { new NameComponent("Agent",""),
 new NameComponent("voyager", "")};
voyagerContext = _initContext.bind_new_context(path_voyager);
} catch (org.omg.CosNaming.NamingContextPackage.AlreadyBound alreadyBoundException){
// voyager context already bound
System.err.println("VoyagerMasaGatewayStationaryAgent.run(): WARNING: voyager-NamingContext already bound to masa-NamingContext!");
// get voyager naming-context
NameComponent name_voyager = new NameComponent("voyager", "");
NameComponent path_voyager[] = {name_voyager};
try{
voyagerContext = NamingContextHelper.narrow(_initContext.resolve(path_voyager));
} catch (Exception e){
e.printStackTrace();
}
} catch (Exception e){
e.printStackTrace();
}

// create AgentManager
_voyagerAgentManager = new VoyagerAgentManager(_agentManager.getAgentSystemService(),
 voyagerContext,
 _ownAgentContext);
// create Information about it
CfMAF.Name name = new Name();
name.agent_system_type = VOYAGER_TYPE.shortValue();

// get CORBA - object reference of myself
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
orb.connect(_voyagerAgentManager);
// Register Voyager-AS in Masa AgentSystem for redirection of create,
// terminate, etc. invocation
_agentManager.getAgentSystemService().registerAgentManager(name,_voyagerAgentManager);
Debug.debug("\n\n\nVoyagerAgentManager registered.\n\n\n");
```

```
// create and start NamingGateway
_namingGatewayThread = new Thread(new Runnable () {
/**
 * Implementation of the _namingGatewayThread
 */
public void run(){
try{
NamingGateway.main(_voyagerServer);
}catch(Exception namingGatewayEsception){
namingGatewayEsception.printStackTrace();
}
}
});
try{
_namingGatewayThread.start();
}catch(Exception e){
e.printStackTrace();
}
}


/**
 * This method is called by the NamingGateway,
 * if Voyager agents are terminated. It terminates then the
 * VoyagerProxyAgents according to the terminated Voyager agents
 *
 * @param agent_names Names of the terminated Voyager agents
 *
 */
public void agents_terminated(String[] agent_names){
Debug.debug("VoyagerMasaGatewayStationaryAgent.agents_terminated()");
try{
for (int i=0; i<agent_names.length; i++)
if (_agentsTerminatedByApplet.contains(agent_names[i]))
// check whether agent terminated by applet
_agentsTerminatedByApplet.removeElement(agent_names[i]);
else
_voyagerAgentManager.terminate_voyagerproxyagent(agent_names[i]);
} catch(Exception e){
e.printStackTrace();
}
}


/**
 * This method is called by the NamingGateway,
 * if new Voyager agents are created. It creates then a
 * VoyagerProxyAgent for the created Voyager agents
 *
 * @param agent_names Names of the new created Voyager agents
 *
```

```
 */
public void agents_created(String[] agent_names, String place_name){
Debug.debug("VoyagerMasaGatewayStationaryAgent.agents_created()");
try{
for (int i=0; i<agent_names.length; i++)
// check whether agent not created by applet and proxy already created
if(_agentsCreatedByApplet.contains(agent_names[i])){
Debug.debug("VoyagerMasaGatewaySationaryAgent.agents_created() - contains " + agent_names[i]);
_agentsCreatedByApplet.removeElement(agent_names[i]);
}
else
_voyagerAgentManager.create_voyagerproxyagent(agent_names[i],place_name);
    } catch(Exception e){
e.printStackTrace();
    }
}

/**
 * This method is called by the VoyagerMasaGatewayApplet to create an Voyager agent
 * It creates the necessary parameters and calls the create_agent()-Method of the
 * AgentSystemService
 *
 * @param agentname Name of the agent
 * @param classname Java-Classname of the agent
 * @param packagename Java-Packagename of the agentclass
 * @param place_name Voyager server on which the agent should be created, f.e. //hostname:portnumber
 *
 */
public String create_agent(String agentname,
 String classname,
 String packagename,
 String place_name) throws CfMAF.ArgumentInvalid, CfMAF.MAFExtendedException{

Debug.debug("VoyagerMasaGateway create_agent " + classname);

// check whether classname different from agentname
CfMAF.ClassName className[];
if(classname.length() != 0)
className=
new CfMAF.ClassName[] {new CfMAF.ClassName(classname,new byte[0])};
else
className = new CfMAF.ClassName[0];

// create the parameters for the create_agent() call
java.io.ByteArrayOutputStream agentBytes;
NameWrapper nw;
CfMAF.AgentProfile ap;
java.io.ByteArrayOutputStream constructorArgs;
try{
agentBytes = new java.io.ByteArrayOutputStream();
java.io.ObjectOutputStream outputAgent = new java.io.ObjectOutputStream(agentBytes);
```

```
outputAgent.writeObject(packagename);

// set type for voyager agent system
nw = new NameWrapper(agentname);
nw._name.agent_system_type = VOYAGER_TYPE.shortValue();

// create the profile of the agent
// not supported, just dummy

ap = new CfMAF.AgentProfile();
org.omg.CORBA.Any props[] = new org.omg.CORBA.Any[1];
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
org.omg.CORBA.Any any = orb.create_any();
any.insert_wstring("Property");

props[0]=any;
ap.language_id=1;
ap.serialization=1;
ap.properties= props;
ap.agent_system_description= "Agent System of MNM-Team";

constructorArgs = new java.io.ByteArrayOutputStream();
java.io.ObjectOutputStream constructorObject = new java.io.ObjectOutputStream(constructorArgs);
}catch(Exception argumentException){
argumentException.printStackTrace();
throw new CfMAF.ArgumentInvalid();
}
Debug.debug("\n\nVoyagerMasaGateway.create_agent() - place_name: " + place_name + "\n\n");

// create the agent
CfMAF.Name res = null;
try{
res = getAgentSystemService().create_agent(nw._name,
 ap,
 agentBytes.toByteArray(),
 place_name,
 constructorArgs.toByteArray(),
 className,
 "",
 getAgentSystemService());
// save agentname as agent created by the applet
Debug.debug("VoyagerMasaGateway.create_agent() - save agent name as agent created by applet: " + new String(res.identity));
_agentsCreatedByApplet.addElement(new String(res.identity));
}catch(Exception createException){
createException.printStackTrace();
throw new CfMAF.MAFExtendedException();
}
return new String(res.identity);
}

    /**
```

```
     * Terminates a Voyager agent by calling terminate_agent of the AgentSystemService
     */
public void terminate_agent(String agent_name) throws CfMAF.TerminateFailed, CfMAF.AgentNotFound{
Debug.debug("VoyagerMasaGateway.terminate_agent() - agentname: " + agent_name);
// masa system call: terminate_agent
NameWrapper nw = new NameWrapper(agent_name);
nw._name.agent_system_type = VOYAGER_TYPE.shortValue();
getAgentSystemService().terminate_agent(nw._name);
// save agentname as agent terminated by the applet
_agentsTerminatedByApplet.addElement(agent_name);
}

/**
 * migrates an agent ot the specified target and executes the method there
 *
 * @param agent_name Name of the Agent to migrate
 * @param target Vozager server to which agent to be migrated
 * @param execute Method which will be excuted after migration
 * @param arguments List of arguments that are used by method, e.g. "java.lang.String Hello World!,java.lang.Integer 1"
 */
public void migrate_agent(String agent_name, String target, String execute, String arguments) throws CfMAF.ArgumentInvalid, CfMAF.MAFExtendedException
    {
Debug.debug("VoyagerMasaGateway.migrate_agent() - agentname: " + agent_name);
// build arguments
StringTokenizer declarations = new StringTokenizer(arguments,","); // pairs of type and value
Object [] args = new Object[declarations.countTokens()];

int i=0;
while(declarations.hasMoreTokens()){
String decl = declarations.nextToken();
// get type
String type = decl.substring(0,decl.indexOf(" "));
// get value
String value = decl.substring(decl.indexOf(" ")+1,decl.length());
Debug.debug("VoyagerMasaGateway.migrate_agent() - arguments: type: "
+ type
+ " value: " + value);
// instantiate class of type with value,
// class must have a constructor,
// accepting string as argument
try{
Debug.debug("VoyagerMasaGateway.migrate_agent() - search argument class");
Class typeClass = Class.forName(type);
Debug.debug("VoyagerMasaGateway.migrate_agent() - get constructor");
Constructor typeConstruct =
typeClass.getConstructor(new Class [] {Class.forName("java.lang.String")});
Debug.debug("VoyagerMasaGateway.migrate_agent() - create instance");
args[i] = typeConstruct.newInstance(new Object[]{value});
i++;
}catch(Exception e){
e.printStackTrace();
```

```
throw new CfMAF.ArgumentInvalid();
}
}

// migrate agent
_voyagerAgentManager.migrate_agent(agent_name, target, execute, args);
// save migration as termination and creation by applet
_agentsTerminatedByApplet.addElement(agent_name);
_agentsCreatedByApplet.addElement(agent_name);
    }

/**
 * list all voyager agents
 */
public String [] list_all_agents(){
CfMAF.Name [] agents = _voyagerAgentManager.list_all_agents();
String [] result = new String [agents.length];;
for(int i=0; i<result.length; i++){
result[i] = new String (agents[i].identity);
}
return result;
}

/**
 * Lists all voyager server
 */
public String [] list_all_voyagerserver(){
return _voyagerServer;
}

/**
 * Exceutes a method of a Voyager agent
 *
 * @param agent_name  Name of the Vozager agent
 * @param method      Name of the Method to be executed
 * @param arguments   Arguments for method, e.g. "java.lang.String Hello World!,java.lang.Integer 1"
 *
 */
public String execute_agent(String agent_name, String method, String arguments) throws CfMAF.ArgumentInvalid, CfMAF.MAFExtendedException{
// build arguments
StringTokenizer declarations = new StringTokenizer(arguments,","); // pairs of type and value
Object [] args = new Object[declarations.countTokens()];

int i=0;
while(declarations.hasMoreTokens()){
String decl = declarations.nextToken();
// get type
String type = decl.substring(0,decl.indexOf(" "));
// get value
String value = decl.substring(decl.indexOf(" ")+1,decl.length());
Debug.debug("VoyagerMasaGateway.migrate_agent() - arguments: type: "
```

```
+ type
+ " value: " + value);
// instantiate class of type with value,
// class must have a constructor,
// accepting string as argument
try{
Debug.debug("VoyagerMasaGateway.migrate_agent() - search argument class");
Class typeClass = Class.forName(type);
Debug.debug("VoyagerMasaGateway.migrate_agent() - get constructor");
Constructor typeConstruct =
typeClass.getConstructor(new Class [] {Class.forName("java.lang.String")});
Debug.debug("VoyagerMasaGateway.migrate_agent() - create instance");
args[i] = typeConstruct.newInstance(new Object[]{value});
i++;
}catch(Exception e){
e.printStackTrace();
throw new CfMAF.ArgumentInvalid();
}
}
// execute method
return _voyagerAgentManager.execute_agent(agent_name, method, args);
}

    /**
     * Returns voyager server of agent
 *
 * @param Name of the Voyager agent
     */
    public String get_server(String agent_name){
return _voyagerAgentManager.get_server(agent_name);
    }
}
```

B.3 NamingGateway

```
package de.unimuenchen.informatik.mnm.masa.agentSystem.voyagerintegration;

import de.unimuenchen.informatik.mnm.masa.agentSystem.nameservice.*;
import de.unimuenchen.informatik.mnm.masa.agent.voyagermasagateway.*;
import de.unimuenchen.informatik.mnm.masa.tools.*;

import com.objectspace.voyager.directory.*;
import com.objectspace.voyager.reference.*;
import com.objectspace.voyager.corba.*;
import com.objectspace.voyager.orb.*;
import com.objectspace.voyager.*;
import com.objectspace.lib.util.Console;

import org.omg.CosNaming.*;
import org.omg.CORBA.*;
```

```
import java.util.Vector;

/**
 * The NamingGateway scans the Namespaces of Voyager servers and notifies
 * created or terminated agents to the VoyagerMasaGatewayStationaryAgent.
 * The NamingGateway is startet by his main, e.g. in a special thread
 * of the VoyagerMasaGatewayStationaryAgent
 */
public class NamingGateway
{

/**
 * Timerintervall between updates
 */
private static final int POLLINGTIME = 5000;

/**
 * Compares to list of Strings and return the difference
 *
 * @return a-b, if a=b or a<b null
 */
private static String[] getDifference(String[] a, String []b){
Vector difference = new Vector();
int num=0;
boolean contains = false;
for(int i=0;i<a.length;i++){
for(int j=0; j<b.length && !contains; j++)
if(a[i].equals(b[j]))
contains = true;
if (!contains) {
difference.addElement(a[i]);
num++;
}
contains = false;
}
if (num == 0) return null;  // b>=a
String [] result = new String [num];
difference.copyInto(result);
return result;
}

/**
 * URL of the Voyager-Server
 */
private String[] _vurls;

/**
 * List of all voyager agents on all voyager server
 */
private String [][] agents;
```

```
/**
 * Agnet to communicate with masa
 */
private VoyagerMasaGateway gatewayAgent;

/**
 * Constructor
 */
public NamingGateway(String [] vurls){
this._vurls = vurls;
agents = new String[vurls.length][];
gatewayAgent = getVoyagerMasaGatewayStationaryAgent();
}


/**
 *      Gets the Voyager's Directory of the namingservice
 */
private IDirectory getVoyagerDirectory(String vurl) throws Exception
    {

// Get Voyager Directory
Console.setLogLevel(Console.VERBOSE);

// Proxy from Server
IDirectory directory;

RemoteAddress addr =
new RemoteAddress(vurl,Class.forName("com.objectspace.voyager.directory.Directory").getName(),-9); // -9=Directory of Namespace, not documented by Voyager
IProxy proxy = addr.bind(true).newProxy();
directory = (IDirectory) proxy;

return directory;
    }




/**
 * Lists all voyager remote objects bound to the voyager's namespace
 *
 */
private synchronized String[] listAllVoyagerAgents(String vurl){
String[] result=new String[0];
try{
try{
Voyager.startup();
}catch(com.objectspace.voyager.StartupException e){
System.err.println("WARNING: " + e);
}
IDirectory directory = getVoyagerDirectory(vurl);
```

```
java.lang.Object[] keys = directory.getKeys();
result = new String[keys.length];
for(int i=0; i<result.length; i++){
result[i] = keys[i].toString();
}
Voyager.shutdown();
} catch (Exception e){
// Exceptions if there is no Voyager server
if ((e instanceof  com.objectspace.voyager.RuntimeRemoteException) ||
(e instanceof  java.net.ConnectException)){
} else{
e.printStackTrace();
result = null;
}
}

return result;
}

/**
 * Starts scanning the voyager's namespace
 * and reports changes to the VoyagerMasaGatewayStationaryAgent
 */
private void start(){
// check whether agents alread started
for(int i=0; i<_vurls.length; i++){
agents[i] = listAllVoyagerAgents(_vurls[i]);
if(agents[i].length != 0)
gatewayAgent.agents_created(agents[i],_vurls[i]);

}

// start pooling
int count_scanning=0;
while(true){
// check every voyager server for agents
for(int i=0; i<_vurls.length; i++){
// get new actual list of all agents running
String [] agents_running = listAllVoyagerAgents(_vurls[i]);
String [] created_agents = getDifference(agents_running,agents[i]);
if (created_agents != null){
System.out.println("NamingGateway.start() - agents created on " + _vurls[i]);
gatewayAgent.agents_created(created_agents,_vurls[i]);
}
String[] terminated_agents = getDifference(agents[i],agents_running);
if (terminated_agents != null){
System.out.println("NamingGateway.start() - agents terminated on " + _vurls[i]);
gatewayAgent.agents_terminated(terminated_agents);
}
agents[i] = agents_running; // save agent-lis for next polling loop
}
```

```
try{
Thread.sleep(POLLINGTIME);
} catch (Exception e){
e.printStackTrace();
}
count_scanning = count_scanning==255 ? 0 : count_scanning+1;
}
}

/**
 * Gets the VoyagerMasaGatewayStationaryAgent via namingcontext
 * to communicate with masa.
 * Gets the Agent to handle the events of new created or terminated
 * voyager agents
 */

private VoyagerMasaGateway getVoyagerMasaGatewayStationaryAgent(){
try{
// get namingcontext
ORB orb = ORB.init();
String ior = NamingWebServer.getIORWWW();
org.omg.CORBA.Object objRef = orb.string_to_object(ior);
NamingContext namingcontext = NamingContextHelper.narrow(objRef);
NameComponent path_agent[] = {new NameComponent("Agent",""),
new NameComponent("global",""),
new NameComponent("VoyagerMasaGateway","")};
return VoyagerMasaGatewayHelper.narrow(namingcontext.resolve(path_agent));
}catch(Exception e){
e.printStackTrace();
return null;
}
}

/**
 * Start creates a object of the NamingGateway and start it
 */
public static void main(String[] args) throws Exception
    {
NamingGateway naminggateway = new NamingGateway(args);
naminggateway.start();
    }
}
```

B.4 VoyagerAgentManager

```
package de.unimuenchen.informatik.mnm.masa.agentSystem.voyagerintegration;

import com.objectspace.voyager.*;
import com.objectspace.voyager.agent.*;
import com.objectspace.voyager.message.*;
```

```
import com.objectspace.lib.util.*;

import org.omg.CosNaming.*;
import java.util.*;

import CfMAF.*;
import de.unimuenchen.informatik.mnm.masa.tools.*;
import de.unimuenchen.informatik.mnm.masa.agentSystem.*;
import de.unimuenchen.informatik.mnm.masa.agent.voyagerproxyagent.*;

public class VoyagerAgentManager extends GenericAgentManager
{
/**
 * CfMAF.Type of the Voyager-Agentsystem
 */
private Short VOYAGER_TYPE = new Short (System.getProperty("de.unimuenchen.informatik.mnm.masa.voyager.type","5"));

/**
 * running _agentsVoyagerName
 * form: (String agent_name, String //voyagerhost:port/agent_name)
 */
private Hashtable _agentsVoyagerName = new Hashtable();

/**
 * proxy agents in masa
 * form: (String agent_name, CfMAF.Name proxy)
 */
private Hashtable _voyagerProxy = new Hashtable();

/**
 * MASA-AgnetSystem
 */
private AgentSystemService _masaAgentSystemService;

/**
 * voyager naming context in masa
 */
private NamingContext _voyagerContext;

/**
 * context in which VoyagerMasaGateway bound to
 */
private NamingContext _ownContext;


/**
 * Constructor
 */
public VoyagerAgentManager(AgentSystemService masaAgentSystemService, NamingContext voyagerContext, NamingContext ownContext)
    {
Debug.debug("VoyagerAgentManager.<init>");
```

```
// init attirbutes
this._masaAgentSystemService = masaAgentSystemService;
this._voyagerContext = voyagerContext;
this._ownContext = ownContext;
    }


/**
 * @see AgentManager.connectToAgent()
 */
public org.omg.CORBA.Object connectToAgent(Name agent_name)
throws AgentNotFound
    {
return null; // not supported
    }


/**
 * @see AgentManager.getAgentSystemURL()
 */
protected String getAgentSystemURL(){
return null; // not supported
}

/**
 * Creates a voyageragent and a voyagerproxyagent:
 * 1. get agent arguments
 * 2. get agent's classname
 * 3. create proxy agent in MASA
 * 4. create Voyager agent
 * 5. update internal data
 * If creation of Voyager agent fails then the
 * already created voyagerproxyagent will be terminated
 * because its easier to terminate a voyagerproxyagent
 * in MASA than a Voyager agent
 *
 * @param agent_name    Agentname
 * @param agent_profile not used
 * @param agent         not used
 * @param place_name    Voyager server on which remote object will be created
 * @param arguments     not used
 * @param class_names   List of classnames to instantiate agent,
 *                      only necessary if classname different from agentname
 * @param code_base     not used
 * @param class_provider not used
 *
 */

public synchronized Name create_agent(Name agent_name,
AgentProfile agent_profile,
byte[] agent,
```

```
String place_name,
byte[] arguments,
ClassName[] class_names,
String code_base,
MAFAgentSystem class_provider)
throws ClassUnknown, ArgumentInvalid,
 DeserializationFailed, MAFExtendedException{
/*************************
 *  1. get agent arguments
 *************************/
java.util.Vector agentArgs= null;
if ( agent.length > 0 ) {
agentArgs= new java.util.Vector(4);
try {
java.io.ObjectInputStream input=
new java.io.ObjectInputStream(new java.io.ByteArrayInputStream(agent));
while( true ){
agentArgs.addElement(input.readObject());
}
}
catch(java.io.EOFException eofe){
Debug.debug("VoyagerAgentManager.create_agent():Stream fully read!");
//Ok, Stream full readed
}
catch(Throwable e){
e.printStackTrace();
throw new ArgumentInvalid();
}
}

/*************************
 * 2. get agent's classname
 *************************/

// get classname, if classname is different
// from agent_name then it is defined
// in class_names else class_names is empty
String className;
if (class_names.length != 0)
className = class_names[0].name;
else
className = new String(agent_name.identity);
int indexPackageName= 0;
String packageName= (String)agentArgs.elementAt(indexPackageName);
if ( packageName.length() > 0 )
className= packageName + "." + className;

// build agent name,
// if already agent_name exists
// then put <n> to the end of the name (voyager) convention
Debug.debug("VoyagerAgentManager.create_agent() - build agent name");
```

```
String agentName= new String(agent_name.identity);
// check whether already exists
if(_agentsVoyagerName.containsKey(agentName)){
// have to change agent_name
int i= 1;
Debug.debug("VoyagerAgentManager.create_agent() - _agentsVoyagerName: " + _agentsVoyagerName.toString());
while(_agentsVoyagerName.containsKey(new String(agentName + "<"+i+">"))){
Debug.debug("VoyagerAgentManager.create_agent() - existing agent names: " + new String(agentName + "<"+i+">"));
i++;
}
agentName = agentName + "<"+i+">";
}
Debug.debug("VoyagerAgentManager.create_agent() - agentName: " + agentName);


/********************************
 * 3. create proxy agent in MASA
 ********************************/

// create proxy for MASA
try{
create_voyagerproxyagent(agentName,place_name);
}catch (Exception createProxyException){
createProxyException.printStackTrace();
throw new MAFExtendedException();
}


/********************************
 * 4. create Voyager agent
 ********************************/
try{
// start voyager client
try{
Voyager.startup();
}catch(com.objectspace.voyager.StartupException e){
Debug.debug("WARNING: " + e);
}

// create remote object in voyager
java.lang.Object obj = null;
Debug.debug("VoyagerAgentManager.create_agent() - className:" + className);
Debug.debug("VoyagerAgentManager.create_agent() - place_name:" + place_name);
obj = Factory.create(className,place_name);

// bind voyager remote object to voyager-namespace
Debug.debug("VoyagerAgentManager.create_agent() - Namespace.bind: "
+ place_name
+ "/"
+ new String(agent_name.identity));
Namespace.bind(place_name + "/" + agentName,obj);
```

```
// shutdown Voyager client
Voyager.shutdown();
} catch (Exception generalVoyagerException){
generalVoyagerException.printStackTrace();
try{
Voyager.shutdown();
}catch(Exception shutdownVoyagerException){
shutdownVoyagerException.printStackTrace();
}
// could not create voyager agent so terminate voyager proxy agent in MASA
try{
terminate_voyagerproxyagent(agentName);
} catch(CfMAF.TerminateFailed terminateException){
terminateException.printStackTrace();
}
throw new MAFExtendedException();
}

Debug.debug("VoyagerAgentManager.create_agent() - _agentsVoyagerName: " + _agentsVoyagerName.toString());

/************************
 * 5. update internal data
 ************************/

// update agent_name
agent_name.identity = agentName.getBytes();
return agent_name;
}


/**
 * @see AgentManager.get_agent_status()
 */
public AgentStatus get_agent_status(Name agent_name)
throws AgentNotFound{
return null; // not supported
}

/**
 * @see AgentManager.get_authinfo()
 */
public AuthInfo get_authinfo(Name agent_name)
throws AgentNotFound{
return null; // not supported
}

/**
 * Lists all voyager agents
 */
public CfMAF.Name[] list_all_agents(){
```

```
CfMAF.Name [] result = new CfMAF.Name[0];
try{
Debug.debug("VoyagerAgentManager.list_all_agents() - proxys: " + _voyagerProxy.toString());
result = new CfMAF.Name[_voyagerProxy.size()];
Enumeration agents = _voyagerProxy.keys();
Debug.debug("VoyagerAgentManager.list_all_agents() - got "
+ result.length + " keys from table");
for(int i=0; i<result.length; i++){
result[i] = new CfMAF.Name();
result[i].identity = ((String) agents.nextElement()).getBytes();
}
} catch (Exception e){
e.printStackTrace();
}
return result;
}

/**
 * @see AgentManager.list_all_agents_of_authority()
 */
public CfMAF.Name[] list_all_agents_of_authority(byte [] authority){
return null; // not supported
}

/**
 *  @see AgentManager.receive_agent()
 */
public void receive_agent(Name agent_name,
AgentProfile agent_profile,
byte[] agent,
String place_name,
ClassName[] class_names,
String code_base,
MAFAgentSystem agent_sender)
throws ClassUnknown, ArgumentInvalid,
 DeserializationFailed, MAFExtendedException{
// not supported
}

/**
 *  @see AgentManager.resume_agent()
 */
public void resume_agent(Name agent_name)
throws AgentNotFound, ResumeFailed, AgentIsRunning{
// not supported
}

/**
 *  @see AgentManager.suspend_agent()
 */
public void suspend_agent(Name agent_name)
```

```
throws AgentNotFound, SuspendFailed, AgentIsSuspended{
// not supported
}

/**
 * Terminates a Voyager agent, i.e. unbins it
 * from Namespace and hopes that
 * Garbage Collection will delete it
 *
 * @param agent_name name of the Voyager agent
 */
public synchronized void terminate_agent(Name agent_name)
throws AgentNotFound, TerminateFailed{
// unbind agent in voyager
try{
try{
Voyager.startup();
}catch(com.objectspace.voyager.StartupException e){
System.out.println("WARNING: " + e);
}
Namespace.unbind((String)_agentsVoyagerName.get(new String (agent_name.identity)));
Voyager.shutdown();
} catch(Exception e){
e.printStackTrace();
Voyager.shutdown();
throw new TerminateFailed();
}
// terminate proxy agent
try{
terminate_voyagerproxyagent(new String (agent_name.identity));
}catch(Exception e){
e.printStackTrace();
throw new TerminateFailed();
}
}

/**
 * @see AgentManager.terminateAgentManager()
 */
protected void terminateAgentManager()
throws TerminateFailed{
// not supported
}

/**
 * @see AgentManager.setAgentSystemService()
 */
protected void setAgentSystemService(AgentSystemService ass){
// not supported
}
```

```
/**
 * @see AgentManager.writeAgentToFile()
 */
public void writeAgentToFile(CfMAF.Name agentName, java.lang.String file)
throws WriteFailed{
// not supported
}

/**
 * @see AgentManager.readAgentFromFile()
 */
public CfMAF.Name readAgentFromFile(java.lang.String file)
throws ReadFailed{
return null; // not supported
}

/**
 * Creates ProxyVoyagerAgents for the VoyagerAgents
 *
 * @param agent_names Name of the created Voyager agent
 * @param voyager_server Voyager server on which Voyager agent has been created
 */
public void create_voyagerproxyagent(String agent_name, String place_name) throws CfMAF.ArgumentInvalid, CfMAF.DeserializationFailed, CfMAF.MAFExtendedException,CfMAF.ClassUnknown
    {
// create the parameters for the create_agent() call
java.io.ByteArrayOutputStream agentBytes = new java.io.ByteArrayOutputStream();

java.io.ObjectOutputStream outputAgent;
java.io.ObjectOutputStream constructorObject;
java.io.ByteArrayOutputStream constructorArgs;
CfMAF.ClassName [] className;
try{
outputAgent = new java.io.ObjectOutputStream(agentBytes);
outputAgent.writeObject("de.unimuenchen.informatik.mnm.masa.agent.voyagerproxyagent");
outputAgent.writeObject("MobileAgent");
outputAgent.writeObject("_tie_");
outputAgent.writeObject("Operations");
outputAgent.writeObject(new Boolean(false));
outputAgent.writeObject("");

constructorArgs = new java.io.ByteArrayOutputStream();
constructorObject = new java.io.ObjectOutputStream(constructorArgs);
// set Classname of agent
className = new CfMAF.ClassName[1];
className[0] = new CfMAF.ClassName("VoyagerProxyAgent", new byte[0]);
constructorObject.writeObject(agent_name); // write the values to constructorObject
}catch(java.io.IOException ioException){
ioException.printStackTrace();
throw new ArgumentInvalid();
}
```

```
// agents names in MASA must NOT contain "."
// change "." to "_"
StringTokenizer token = new StringTokenizer(agent_name,".");
String new_name = new String();
while(token.hasMoreElements())
{
new_name = new_name + token.nextToken() + "_";
if (!token.hasMoreElements())
new_name = new_name.substring(0,new_name.length()-1);
}
agent_name = new_name;
NameWrapper nw = new NameWrapper(agent_name);

//create the profile of the agent
CfMAF.AgentProfile ap = new CfMAF.AgentProfile();
org.omg.CORBA.Any props[] = new org.omg.CORBA.Any[1];
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
org.omg.CORBA.Any any = orb.create_any();
any.insert_wstring("Property");

props[0]=any;
ap.language_id=1;
ap.serialization=1;
ap.properties= props;
ap.agent_system_description= "Voyager Agent System of MNM-Team";

      // create the agent
CfMAF.Name res = _masaAgentSystemService.create_agent(nw._name,
ap,
agentBytes.toByteArray(),
new String("default"),
constructorArgs.toByteArray(),
className,
"",
(CfMAF.MAFAgentSystem) _masaAgentSystemService);
try{
// get agent corba reference
org.omg.CORBA.Object agentRef = _ownContext.resolve(NameWrapper.getNameComponents(res,false));
// put agent in voyager namingcontext
_voyagerContext.bind(new NameComponent [] {new NameComponent(new String(res.identity),"")}, agentRef);

// save agent and its CfMAF name in table
_voyagerProxy.put(agent_name,res);

// save full voyager Name
// put agent and its voyager name in agent-hashtable
_agentsVoyagerName.put(agent_name,place_name + "/" + agent_name);
}catch(Exception corbaException){
corbaException.printStackTrace();
// try to terminate proxy again
try{
```

```
terminate_voyagerproxyagent(agent_name);
}catch(Exception terminateProxyException){
terminateProxyException.printStackTrace();
// can't do any thing more, user should restart VoyagerMasaGateway!
}
throw new CfMAF.MAFExtendedException();
}
    }

/**
 * Terminates a Voyager Agent's proxyavoyageragent in Masa
 *
 * @param agent_name Name of the Voyager Agent
 */
public void terminate_voyagerproxyagent(String agent_name) throws TerminateFailed{
Debug.debug("VoyagerAgentManager.terminate_voyagerproxyagent()");
try{
CfMAF.Name name = (CfMAF.Name) _voyagerProxy.remove(agent_name);
// set voyager proxy ready for termination
org.omg.CORBA.Object obj = _ownContext.resolve(NameWrapper.getNameComponents(name,false));
VoyagerProxyAgent proxyAgent = VoyagerProxyAgentHelper.narrow(obj);
proxyAgent.setReadyForTermination();

// terminate voyager proxy
_masaAgentSystemService.terminate_agent(name);
// unbind it from voyager directory in MASA
_voyagerContext.unbind(new NameComponent [] {new NameComponent(new String(name.identity),"")});

// remove voyager agent name from table
_agentsVoyagerName.remove(agent_name);
}catch(Exception generalTerminateException){
generalTerminateException.printStackTrace();
throw new TerminateFailed();
}
}

/**
 * Terminates all voyager proxy agents
 * in MASA at termination of voyager masa gateway
 */
public void cleanUp() throws CfMAF.TerminateFailed{
// terminate all proxy agents
Enumeration proxyAgents = _voyagerProxy.keys();
while(proxyAgents.hasMoreElements()){
terminate_voyagerproxyagent((String) proxyAgents.nextElement());
}
}

/**
 * migrates a voyager agent
 *
```

```
 * @param agent_name Name pf the Voyager agent in MASA
 * @param target     Voyager server to which the agent should be migrated
 * @param execute    Name of the mathod which will be executed after migration
 * @param args       Arguments for the method
 */
public synchronized void migrate_agent(String agent_name, String target, String execute, Object [] arguments) throws MAFExtendedException{
// build agentname for voyager
String voyager_agent_name = (String) _agentsVoyagerName.get(agent_name);
try{
try{
Voyager.startup();
}catch(com.objectspace.voyager.StartupException e){
System.out.println("WARNING: " + e);
}
// get proxy
Object remObj = Namespace.lookup(voyager_agent_name);
// make agent of remote object
IAgent agent = Agent.of(remObj);
// migrate agent
if(execute.length() > 0 && arguments.length > 0){
agent.moveTo(target,execute,arguments);
}
else if (execute.length() > 0)
agent.moveTo(target,execute);
else
agent.moveTo(new String(target));
Namespace.unbind(voyager_agent_name);
try{
Namespace.bind(target + "/" + agent_name,remObj);
// update entry in agent table with new location of voyager agent
_agentsVoyagerName.put(agent_name,target + "/" + agent_name);
}catch(NamespaceException bindException){
//try to rebind to old location
Namespace.bind(voyager_agent_name,remObj);
throw new MAFExtendedException();
}
// shutdown voyager client
Voyager.shutdown();
} catch (Exception generalVoyagerException){
generalVoyagerException.printStackTrace();
Voyager.shutdown();
throw new MAFExtendedException();
}
}

/**
 * Executes a method of a Voyager Agent
 * @param agent_name Name of the Voyager agent
 * @param method Methodname
 * @param args List of Arguments for method
 */
```

```
public String execute_agent(String agent_name, String method, Object[] args)throws CfMAF.MAFExtendedException{
try{
Voyager.startup();
}catch(StartupException e){
System.out.println("WARNING: " + e);
}
try{
String voyagerName = (String) _agentsVoyagerName.get(agent_name);
if(voyagerName == null)
throw new CfMAF.MAFExtendedException();
Object ref = Namespace.lookup(voyagerName);
Debug.debug("changed");
Result result = result = Sync.invoke(ref,method,args);
Voyager.shutdown();
try{ return result.readObject().toString();}
catch(NullPointerException voidReturnException){return "void";}
}catch(Exception e){
e.printStackTrace();
throw new CfMAF.MAFExtendedException();
}
}

    /**
     * Returns the location of a Voyager server
 *
 * @param agent_name Name of the Voyager agent
     */
    public String get_server(String agent_name){
String fullName = (String) _agentsVoyagerName.get(agent_name);
fullName = fullName == null ? "" : fullName.substring(0,fullName.indexOf(agent_name));
return fullName;
    }
}
```

B.5 VoyagerProxyAgentMobileAgent

```
package de.unimuenchen.informatik.mnm.masa.agent.voyagerproxyagent;


import de.unimuenchen.informatik.mnm.masa.agentSystem.voyagerintegration.*;
import de.unimuenchen.informatik.mnm.masa.agentSystem.*;
import de.unimuenchen.informatik.mnm.masa.agent.*;
import de.unimuenchen.informatik.mnm.masa.agent.voyagermasagateway.*;
import CfMAF.*;

import de.unimuenchen.informatik.mnm.masa.tools.Debug;

import java.util.StringTokenizer;
```

```
/**
 * Class to represent a Voyager agent in MASA
 */

public class VoyagerProxyAgentMobileAgent extends MobileAgent
implements VoyagerProxyAgentOperations
{

/**
 * Name of the Voyager agent
 * (not supported)
 */
private String myname;

/**
 * Mark if termination is executed by VoyagerAgentManager
 * or by AgentSystemApplet
 */
private boolean readyForTermination = false;

public VoyagerProxyAgentMobileAgent(String myname){
Debug.debug("VoyagerProxyAgent.<init>");
this.myname = myname;
}

/**
 * To call for a clean temrination, i.e. if this method is not called before
 * VoyagerProxyMobileAgent is terminated then VoyagerProxyMobileAgent supposes that the Voyager Agent is not terminated!!
 */
public void setReadyForTermination(){
Debug.debug("VoyagerProxyAgent ready for termination");
readyForTermination = true;
}

/**
 * Is called during termination
 * @see Agent.cleanUp()
 */
public void cleanUp() throws Throwable
    {
Debug.debug("VoyagerProxyAgentMobileAgent.cleanUp()");
// terminate Voyager Agent
if(!readyForTermination)
throw new Exception("VoyagerProxyAgent must not be terminated if voyager agent is still running");
    }

/**
 * @see Agent.checkSerialization()
 */
public void checkSerialization() throws CouldNotMigrate
```

```
    {
Debug.debug("VoyagerVoyagerProxyAgentMobileAgent.checkSerialization()");
    }

/**
 * Modifies the _url: new _url shows Applet of the VoyagerMasaGatewayStationaryAgent
 *
 * @see Agent.run()
 */
public void run()
    {
Debug.debug("VoyagerVoyagerProxyAgentMobileAgent.run()");
// Modify url
Debug.debug("VoyagerVoyagerProxyAgentMobileAgent.run() - url:" + _url);
StringTokenizer token = new StringTokenizer(_url,"/");
String result = "/";
String nextToken;
while(token.hasMoreElements()){
nextToken = token.nextToken();
if(!token.hasMoreElements())
result = result + "VoyagerMasaGateway.html";
else
result = result + nextToken + "/";
}
_url = result;
Debug.debug("VoyagerVoyagerProxyAgentMobileAgent.run() - url:" + _url);
    }
}
```

B.6 VoyagerMasaGatewayApplet

```
package de.unimuenchen.informatik.mnm.masa.agent.voyagermasagateway;

import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;

import de.unimuenchen.informatik.mnm.masa.agentSystem.*;
import de.unimuenchen.informatik.mnm.masa.agent.*;
import de.unimuenchen.informatik.mnm.masa.tools.*;


public class VoyagerMasaGatewayApplet extends AgentApplet
{


    /**
     * AgentSystemService
     */
```

```
    AgentSystemService _ass;

    /**
     * VoyagerMasaGateway
     */
    VoyagerMasaGateway _voyagerMasaGateway;

    /**
     * GUI Elements
     */
    private GridBagLayout _gridLayout = new GridBagLayout();

    private JTabbedPane _jtp_panels = new JTabbedPane();
    private JTextArea _jta_output = new JTextArea();
    private JScrollPane _jscr_output;

    private JButton _jb_create = new JButton("Create");
    private JButton _jb_create_clear = new JButton("Clear");
    private JTextField _jt_create_classname = new JTextField("FOOVoyagerAgent",20);
    private JTextField _jt_create_package = new JTextField("voyageragent",20);
    private JTextField _jt_create_agentname = new JTextField(20);
    private JComboBox _jc_create_server = new JComboBox();

    private JButton _jb_execute = new JButton("Execute");
    private JComboBox _jc_execute_agents= new JComboBox();
    private JTextField _jt_execute_method = new JTextField(40);
    private JTextField _jt_execute_arguments = new JTextField(40);


    private JButton _jb_migrate = new JButton("Migrate");
    private JComboBox _jc_migrate_agents = new JComboBox();
    private JComboBox _jc_migrate_server = new JComboBox();
    private JTextField _jt_migrate_method = new JTextField(40);
    private JTextField _jt_migrate_arguments = new JTextField(40);

    private JComboBox _jc_info_agents = new JComboBox();
    private JLabel _jl_info_voyagerserver;


    private JButton _jb_terminate = new JButton("Terminate");
    private JComboBox _jc_terminate_agents= new JComboBox();


/**
 * Initialize the GUI, init and add elements tu applet
 */
    private void init_gui(){
// create constraint for gridbaglayout, not yet used
GridBagConstraints c = new GridBagConstraints();

//////////////////////
```

```
// create create-panel
//////////////////////
JPanel jpanel_create = new JPanel(false);
jpanel_create.setLayout(_gridLayout);

// set action listener
_jb_create.addActionListener(new AppletActionListener());
_jt_create_agentname.addCaretListener(new AppletCaretListener());

// create labels
JLabel jl_create_titel = new JLabel("Create Voyager Agent");
JLabel jl_create_agentname = new JLabel("Agentname:");
JLabel jl_create_classname = new JLabel("Classname:");
JLabel jl_create_package = new JLabel("Package:");
JLabel jl_create_server = new JLabel("Server:");

// set constraints
// labels
c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor = GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
c.insets = new Insets(10,10,1,10);
_gridLayout.setConstraints(jl_create_agentname,c);
_gridLayout.setConstraints(jl_create_classname,c);
_gridLayout.setConstraints(jl_create_package,c);
_gridLayout.setConstraints(jl_create_server,c);
c.insets = new Insets(10,10,10,10);
c.anchor = GridBagConstraints.CENTER;
_gridLayout.setConstraints(jl_create_titel,c);
// elements for user input
c.insets = new Insets(1,10,10,10);
c.anchor = GridBagConstraints.WEST;
_gridLayout.setConstraints(_jc_create_server,c);
c.fill = GridBagConstraints.HORIZONTAL;
_gridLayout.setConstraints(_jt_create_agentname,c);
_gridLayout.setConstraints(_jt_create_classname,c);
_gridLayout.setConstraints(_jt_create_package,c);

c.insets = new Insets(10,10,10,10);
c.anchor = GridBagConstraints.CENTER;
c.fill = GridBagConstraints.NONE;
_gridLayout.setConstraints(_jb_create,c);

// set size
_jc_create_server.setMaximumSize(new Dimension(getContentPane().getSize().width,_jc_create_server.getSize().height));

// add elements to panel
jpanel_create.add(jl_create_titel);

jpanel_create.add(jl_create_agentname);
jpanel_create.add(_jt_create_agentname);
```

```
jpanel_create.add(jl_create_classname);
jpanel_create.add(_jt_create_classname);

jpanel_create.add(jl_create_package);
jpanel_create.add(_jt_create_package);

jpanel_create.add(jl_create_server);
jpanel_create.add(_jc_create_server);

jpanel_create.add(_jb_create);

// create and add panel to contentpane
_jtp_panels.addTab("create",jpanel_create);


////////////////////////
// execute panel
///////////////////////
JPanel jpanel_execute = new JPanel(false);
jpanel_execute.setLayout(_gridLayout);

// set actnio listener
_jb_execute.addActionListener(new AppletActionListener());
_jt_execute_method.addCaretListener(new AppletCaretListener());

// create labels
JLabel jl_execute_titel = new JLabel("Execute Method of Voyager Agent");
JLabel jl_execute_agentname = new JLabel("Agentname:");
JLabel jl_execute_method = new JLabel("Method:");
JLabel jl_execute_arguments = new JLabel("Arguments:");
JLabel jl_execute_descript = new JLabel("(seperated by comma, e.g. \"java.lang.Short 4,java.lang.String Message\")");

// set constraints
// labels
c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor = GridBagConstraints.WEST;
c.insets = new Insets(10,10,1,10);
_gridLayout.setConstraints(jl_execute_agentname,c);
_gridLayout.setConstraints(jl_execute_method,c);
_gridLayout.setConstraints(jl_execute_arguments,c);
c.insets = new Insets(1,10,1,10);
_gridLayout.setConstraints(jl_execute_descript,c);
c.insets = new Insets(10,10,10,10);
c.anchor = GridBagConstraints.CENTER;
_gridLayout.setConstraints(jl_execute_titel,c);
// elements for user input
c.insets = new Insets(1,10,10,10);
c.anchor = GridBagConstraints.WEST;
_gridLayout.setConstraints(_jc_execute_agents,c);
c.fill = GridBagConstraints.HORIZONTAL;
```

```
_gridLayout.setConstraints(_jt_execute_method,c);
_gridLayout.setConstraints(_jt_execute_arguments,c);
c.anchor = GridBagConstraints.CENTER;
c.insets = new Insets(10,10,10,10);
c.fill = GridBagConstraints.NONE;
_gridLayout.setConstraints(_jb_execute,c);

// set size
_jc_execute_agents.setMaximumSize(new Dimension(jpanel_execute.getSize().width,_jc_execute_agents.getSize().height));

// add elements to panel
jpanel_execute.add(jl_execute_titel);

jpanel_execute.add(jl_execute_agentname);
jpanel_execute.add(_jc_execute_agents);

jpanel_execute.add(jl_execute_method);
jpanel_execute.add(_jt_execute_method);

jpanel_execute.add(jl_execute_arguments);
jpanel_execute.add(jl_execute_descript);
jpanel_execute.add(_jt_execute_arguments);

jpanel_execute.add(_jb_execute);
_jtp_panels.addTab("execute",jpanel_execute);


////////////////////////
// migrate panel
///////////////////////
JPanel jpanel_migrate = new JPanel(false);
jpanel_migrate.setLayout(_gridLayout);

// init button
_jb_migrate.addActionListener(new AppletActionListener());

// create labels
JLabel jl_migrate_titel = new JLabel("Migrate Voyager Agent to Server and Execute Method");
JLabel jl_migrate_agentname = new JLabel("Agentname:");
JLabel jl_migrate_server = new JLabel("Server:");
JLabel jl_migrate_method = new JLabel("Method:");
JLabel jl_migrate_arguments = new JLabel("Arguments:");
JLabel jl_migrate_descript = new JLabel("(seperated by comma, e.g. \"java.lang.Short 4,java.lang.String Message\")");

// set constraints
// labels
c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor = GridBagConstraints.WEST;
c.insets = new Insets(10,10,1,10);
_gridLayout.setConstraints(jl_migrate_agentname,c);
_gridLayout.setConstraints(jl_migrate_server,c);
```

```
_gridLayout.setConstraints(jl_migrate_method,c);
_gridLayout.setConstraints(jl_migrate_arguments,c);
c.insets = new Insets(1,10,1,10);
_gridLayout.setConstraints(jl_migrate_descript,c);
c.insets = new Insets(10,10,10,10);
c.anchor = GridBagConstraints.CENTER;
_gridLayout.setConstraints(jl_migrate_titel,c);
// elements for user input
c.insets = new Insets(1,10,10,10);
c.anchor = GridBagConstraints.WEST;
_gridLayout.setConstraints(_jc_migrate_agents,c);
_gridLayout.setConstraints(_jc_migrate_server,c);
c.fill = GridBagConstraints.HORIZONTAL;
_gridLayout.setConstraints(_jt_migrate_method,c);
_gridLayout.setConstraints(_jt_migrate_arguments,c);
c.anchor = GridBagConstraints.CENTER;
c.insets = new Insets(10,10,10,10);
c.fill = GridBagConstraints.NONE;
_gridLayout.setConstraints(_jb_migrate,c);

// set size
_jc_migrate_agents.setMaximumSize(new Dimension(jpanel_migrate.getSize().width,_jc_migrate_agents.getSize().height));
_jc_migrate_server.setMaximumSize(new Dimension(jpanel_migrate.getSize().width,_jc_migrate_server.getSize().height));

// add elements to panel
jpanel_migrate.add(jl_migrate_titel);

jpanel_migrate.add(jl_migrate_agentname);
jpanel_migrate.add(_jc_migrate_agents);

jpanel_migrate.add(jl_migrate_server);
jpanel_migrate.add(_jc_migrate_server);

jpanel_migrate.add(jl_migrate_method);
jpanel_migrate.add(_jt_migrate_method);

jpanel_migrate.add(jl_migrate_arguments);
jpanel_migrate.add(jl_migrate_descript);
jpanel_migrate.add(_jt_migrate_arguments);

jpanel_migrate.add(_jb_migrate);
_jtp_panels.addTab("migrate",jpanel_migrate);

///////////////////////
// terminate panel
//////////////////////
JPanel jpanel_terminate = new JPanel(false);
jpanel_terminate.setLayout(_gridLayout);

// set action listener
_jb_terminate.addActionListener(new AppletActionListener());
```

```
// create label
JLabel jl_terminate_titel = new JLabel("Terminate Voyager Agent");
JLabel jl_terminate_agentname = new JLabel("Agentname:");

// set constraints
c.anchor = GridBagConstraints.CENTER;
_gridLayout.setConstraints(jl_terminate_titel,c);
c.anchor = GridBagConstraints.WEST;
c.insets = new Insets(10,10,1,10);
_gridLayout.setConstraints(jl_terminate_agentname,c);
c.gridwidth = GridBagConstraints.REMAINDER;
c.insets = new Insets(1,10,10,10);
_gridLayout.setConstraints(_jc_terminate_agents,c);
c.anchor = GridBagConstraints.CENTER;
c.insets = new Insets(10,10,10,10);
c.fill = GridBagConstraints.NONE;
_gridLayout.setConstraints(_jb_terminate,c);

// set size
_jc_terminate_agents.setMaximumSize(new Dimension(jpanel_terminate.getSize().width,_jc_execute_agents.getSize().height));


// add elements to panel
jpanel_terminate.add(jl_terminate_titel);
jpanel_terminate.add(jl_terminate_agentname);
jpanel_terminate.add(_jc_terminate_agents);
jpanel_terminate.add(_jb_terminate);

// add panel to tabbed panel
_jtp_panels.addTab("terminate",jpanel_terminate);

/////////////////////////////
// info panel
////////////////////////////
JPanel jpanel_info = new JPanel(false);
jpanel_info.setLayout(_gridLayout);

// set action listener
_jc_info_agents.addItemListener(new ItemListener(){
public  void itemStateChanged(ItemEvent e){
// update info
if(_jc_info_agents.getSelectedItem() != null)
_jl_info_voyagerserver.setText(_voyagerMasaGateway.get_server((String) _jc_info_agents.getSelectedItem()));
}
});

// create label
_jl_info_voyagerserver = new JLabel();
JLabel jl_info_titel = new JLabel("Location of Voyager Agent");
JLabel jl_info_isrun = new JLabel("is running on");
```

```
// set constraints
// labels
c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor = GridBagConstraints.CENTER;
c.insets = new Insets(10,10,10,10);
_gridLayout.setConstraints(jl_info_titel,c);
c.insets = new Insets(10,10,1,10);
_gridLayout.setConstraints(_jc_info_agents,c);
c.insets = new Insets(1,10,1,10);
_gridLayout.setConstraints(jl_info_isrun,c);
c.insets = new Insets(1,10,10,10);
_gridLayout.setConstraints(_jl_info_voyagerserver,c);

jpanel_info.add(jl_info_titel);
jpanel_info.add(_jc_info_agents);
jpanel_info.add(jl_info_isrun);
jpanel_info.add(_jl_info_voyagerserver);


// add panel to tabbed panel
_jtp_panels.addTab("info",jpanel_info);

///////////////////////////
// init content pane
///////////////////////////
getContentPane().setLayout(_gridLayout);

// init text area for output
_jta_output.setMargin(new Insets(5,5,5,5));
_jta_output.setEditable(false);
Color color = new Color(188,210,238);
_jta_output.setBackground(color);
_jscr_output = new JScrollPane(_jta_output,
 JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
 JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
_jscr_output.setBorder(BorderFactory.createTitledBorder("Output"));
_jscr_output.setMinimumSize(new Dimension(getContentPane().getSize().width,100));

// init control panel
//    _jtp_panels.setBorder(BorderFactory.createTitledBorder("Control Agents"));

// set constraints
c.gridwidth = GridBagConstraints.REMAINDER;
c.anchor = GridBagConstraints.CENTER;
_gridLayout.setConstraints(_jtp_panels,c);
c.fill = GridBagConstraints.BOTH;
c.weighty = 2;
_gridLayout.setConstraints(_jscr_output,c);

// add elements to content pane
```

```
getContentPane().add(_jtp_panels);
getContentPane().add(_jscr_output);

}

/**
 * Update the gui after values, like list of agents, have chenged
 */
private void update_gui(){
String [] agents = _voyagerMasaGateway.list_all_agents();
String [] server = _voyagerMasaGateway.list_all_voyagerserver();

// update comboboxes
if (_jc_create_server.getItemCount()!=0) _jc_create_server.removeAllItems();
if (_jc_migrate_agents.getItemCount()!=0) _jc_migrate_agents.removeAllItems();
if (_jc_migrate_server.getItemCount()!=0) _jc_migrate_server.removeAllItems();
if (_jc_terminate_agents.getItemCount()!=0) _jc_terminate_agents.removeAllItems();
if (_jc_execute_agents.getItemCount()!=0) _jc_execute_agents.removeAllItems();
if (_jc_info_agents.getItemCount()!=0) _jc_info_agents.removeAllItems();

// list of agents
for(int i=0;i<agents.length;i++){
_jc_migrate_agents.addItem(agents[i]);
_jc_terminate_agents.addItem(agents[i]);
_jc_execute_agents.addItem(agents[i]);
_jc_info_agents.addItem(agents[i]);
}

//list of voyager server
for(int i=0;i<server.length;i++){
_jc_create_server.addItem(server[i]);
_jc_migrate_server.addItem(server[i]);
}
set_buttons();
}

/**
 * Enables or disables buttons depending on the user input
 */
public void set_buttons(){
// enable or disable buttons
// enables create button if user has specified agent name and there is a Voyager server for the agent
_jb_create.setEnabled((_jt_create_agentname.getText().length()!=0)&&(_jc_create_server.getItemCount()!= 0));
// enables execute button only if ther is a agent and a method specified
_jb_execute.setEnabled((_jc_execute_agents.getItemCount()!= 0)&&(_jt_execute_method.getText().length()!=0));
// enbled if there is an agent and a server to migrate to
_jb_migrate.setEnabled((_jc_migrate_agents.getItemCount()!= 0)&&(_jc_migrate_server.getItemCount()!= 0));
// only enabled if there is an agent
_jb_terminate.setEnabled(_jc_terminate_agents.getItemCount()!= 0);
}
```

```
/**
 * Initializes applet: gets AgenSystemService and adds GUI elements to applet
 */
public void init(){
// get voyager masa gaetway
try {
org.omg.CORBA.Object obj = initCORBA("VoyagerMasaGateway");
_voyagerMasaGateway = VoyagerMasaGatewayHelper.narrow(obj);
}catch(Exception e){
e.printStackTrace();
}
init_gui();
update_gui();
}



/**
 * Handles GUI events
 */
private class AppletActionListener implements ActionListener{

public void actionPerformed(ActionEvent e){
// create agent event
if(e.getActionCommand().equals("Create")){
try{
String name = _voyagerMasaGateway.create_agent(_jt_create_agentname.getText(),
 _jt_create_classname.getText(),
 _jt_create_package.getText(),
 (String) _jc_create_server.getSelectedItem());
_jta_output.append(_jt_create_agentname.getText() + " as " + name + " created.\n");
}catch(Exception ex){
// user output
_jta_output.append("Creation failed: " + ex + "!\n");
}
}
// migrate agent event
else if(e.getActionCommand().equals("Migrate")){
try{
_voyagerMasaGateway.migrate_agent((String) _jc_migrate_agents.getSelectedItem(),
(String) _jc_migrate_server.getSelectedItem(),
_jt_migrate_method.getText(),
_jt_migrate_arguments.getText());
_jta_output.append((String) _jc_migrate_agents.getSelectedItem() + " migrated to " + (String) _jc_migrate_server.getSelectedItem() + ".\n");
}catch(Exception ex){
// user output
_jta_output.append("Migration failed: " + ex + "!\n");
}
}
// terminate agent event
```

```
else if((e.getActionCommand().equals("Terminate"))){
try{
_voyagerMasaGateway.terminate_agent((String) _jc_terminate_agents.getSelectedItem());
_jta_output.append((String) _jc_terminate_agents.getSelectedItem() + " terminated.\n");
}catch(Exception ex){
// user output
_jta_output.append("Termination failed: " + ex + "!\n");
}
}
// execute method
else if((e.getActionCommand().equals("Execute"))){
try{
String result = _voyagerMasaGateway.execute_agent((String) _jc_execute_agents.getSelectedItem(),_jt_execute_method.getText(),_jt_execute_arguments.getText());
_jta_output.append((String) _jc_execute_agents.getSelectedItem() + "."
 +_jt_execute_method.getText() + "("
 + _jt_execute_arguments.getText() + ") executed, result: "
 + result + "\n");
}catch(Exception ex){
// user output
_jta_output.append("Execution failed: " + ex + "!\n");
}
}

// Update list of agents and voyager server in comboboxes
update_gui();
}
}

/**
 * ActionListener for the text areas to check whether there was an input
 * and to update buttons
 */
private class AppletCaretListener implements CaretListener{
public void caretUpdate(CaretEvent e) {
set_buttons();
}
}
}
```

B.7 FOO

```
package voyageragent;

/**
 * Interface for the FOOVoyagerAgent as an example for Voyager agents
 */

public interface FOO{
    void message();
    void print_message( java.lang.String message );
```

```
    void print_number_message(java.lang.Short number, java.lang.String message );
    String get_time();
}
```

B.8 FOOVoyagerAgent

```
package voyageragent;

import com.objectspace.voyager.*;
import com.objectspace.voyager.agent.*;
import java.io.*;
import java.net.*;
import java.util.*;
import com.objectspace.lib.util.*;


/**
 * Example for Voyager agents
 */

public class FOOVoyagerAgent implements voyageragent.FOO, Serializable
{

public FOOVoyagerAgent() {
System.out.println("FOOVoyagerAgent.<init>");
}

public void message(){
System.out.println("This is a message!");
}
public void print_message(String message){
System.out.println("Message:" +  message);
}
public void print_number_message(Short number, String message){
System.out.println("Number: " + number + " Message: " +  message);
}

public String get_time(){
String host = "unknown host";
try{
host = InetAddress.getLocalHost().getHostName();
}catch(UnknownHostException e){
e.printStackTrace();
}
return "Time on " +  host + ": "
+ Calendar.getInstance().getTime();
}
}
```

Literaturverzeichnis

[Bros] BROSE, G.: *Programing Guide JacORB 0.9f.*

[CORBA 2.0] *The Common Object Request Broker: Architecture and Specification.* OMG Specification Revision 2.0, Object Management Group, Juli 1995.

[CORBA 3.0] *The Common Object Request Broker: Architecture and Specification.* OMG Specification Revision 3.0, Object Management Group, December 1995.

[Kemp 98] KEMPTER, B.: *Entwurf eines Java/CORBA-basierten Mobilen Agenten.* Diploma Thesis, Technische Universität München, August 1998.

[Obje a] OBJECT-ORIENTED CONCEPTS, INC.: *ORBacus for C++ and Java, Version 3.1.2.*

[Obje b] OBJECTSPACE: *Voyager, Core Technology 2.0, User Guide.*

[OMG a] OMG: *Interoperable Naming Service.* OMG TC Document orbos/98-10-11.

[OMG b] OMG: *ORB Portability Java Language Mapping.* Initial Submission, OMG TC Document orbos/97-11-12.

[OrHa 98] ORFALI, R. und D. HARKEY: *Client/Server Programming with Java and Corba.* John Wiley and Sons, 2nd Auflage, 1998.

[Röl 99] RÖLLE, H.: *Authentisierung und Autorisierung für das Java/CORBA-Agentensystem MASA.* Dipl, TUM, August 1999.

[TE] TINA EERLIGH, OBJECTSPACE: *Voyager - Namespace.* E-Mail, s. Erster Anhang.