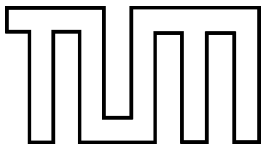
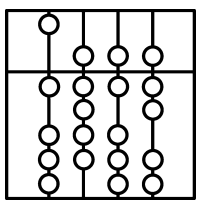


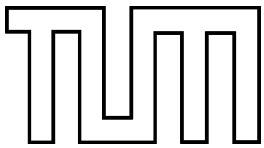
INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Implementierung der MAFFinder-Schnittstelle für die Mobile Agent System Architecture

Josef Georg Gigl





INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Studienarbeit

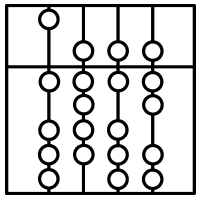
für das

Aufbaustudium Informatik

Implementierung der MAFFinder-Schnittstelle für die Mobile Agent System Architecture

Bearbeiter:	Josef Georg Gigl
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Bernhard Kempter
	Helmut Reiser

Abgabedatum: 3.7.2000



Hiermit versichere ich, dass ich die vorliegende Studienarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 1. Juli 2000

.....

Der Geist ist wie ein Fallschirm.
Er funktioniert nur, wenn er offen ist.
Thomas R. Dewer

An dieser Stelle möchte ich mich besonders bei meinen Betreuern Bernhard Kempter und Helmut Reiser für die Unterstützung und Diskussionsbereitschaft sowie bei Harald Rölle für seine Diskussionsbeiträge bedanken.

Inhaltsverzeichnis

Anhang:

Abbildungen

Tabellen

Abkürzungen

API	Application Programming Interface
BOA	Basic Object Adapter
CORBA	Common Object Request Broker Architecture
CVS	Concurrent Versions System
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
MAF	Mobile Agents Facility
MASA	Mobile Agent System Architecture
MASIF	Mobile Agent System Interoperability Facilities
MNM	Münchner Netz-Management-Team
OMG	Object Management Group
OOC	Object Oriented Concepts (Inc.)
ORB	Object Request Broker
UML	Unified Modelling Language
URI	Uniform Resource Identifiers
URL	Uniform Resource Locators

1 Einführung

Das Münchner Netz-Management-Team (MNM) am Institut für Informatik der Ludwig-Maximilians-Universität München (LMU) hat eine Netz- und System-Management-Plattform basierend auf einer mobilen Agenten-System-Architektur (*Mobile Agent System Architecture*, MASA) entwickelt. MASA ist konform zu den Standards der *Open Management Group* (OMG) plattformunabhängig unter Java realisiert und nutzt die *Common Object Request Broker Architecture* (CORBA) als Kommunikations-Infrastruktur.

Ziel von MASA ist die Verlagerung von Managementfunktionalität von einem zentralen Manager auf dezentrale, intelligente Agenten zur Vorverarbeitung und Entlastung des Managers. Dabei sind zur Erfüllung bestimmter Aufgaben sog. mobile Agenten in der Lage, innerhalb der zu managenden Infrastruktur auf heterogene Plattformen zu migrieren. Eine Übersicht über zugrundliegende Techniken, Aufbau, Funktionalität und Einsatzszenarien von MASA gibt [GHR 99].

Zur Lokalisierung und Registrierung von Agenten setzt MASA derzeit den CORBA-eigenen Namensdienst ein. Der OMG-Standard „*Mobile Agent System Interoperability Facilities*“ (MASIF) sieht dafür allerdings den sog. *MAFFinder* vor, der u.a. auch eine Lokalisierung über entsprechende Filter-Profile ermöglicht. Gegenstand der vorliegenden Studienarbeit ist die Implementierung dieser *MAFFinder*-Schnittstelle in MASA.

Dabei werden nach einer Konkretisierung der Aufgabenstellung zunächst die Anforderungen aus der MASIF-Spezifikation und aus einer Analyse des bestehenden MASA-Systems identifiziert. Darauf aufbauend wird die Architektur zur Realisierung des *MAFFinders* entworfen und anschließend implementiert.

2 Aufgabenstellung

2.1 Komponenten von MASA

Zum Verständnis der nachfolgenden Ausführungen werden grob der Aufbau und die wesentlichen Komponenten von MASA skizziert.

Das MASA-Basissystem, dessen Realsierung in [Kemp 98] ausführlich beschrieben ist, orientiert sich an der MASIF-Spezifikation ([OMG 98-03-09]), deren Konzepte und Terminologie Interoperabilität zwischen den nach ihr realisierten „Mobilen Agenten-Systemen“ sicherstellt.

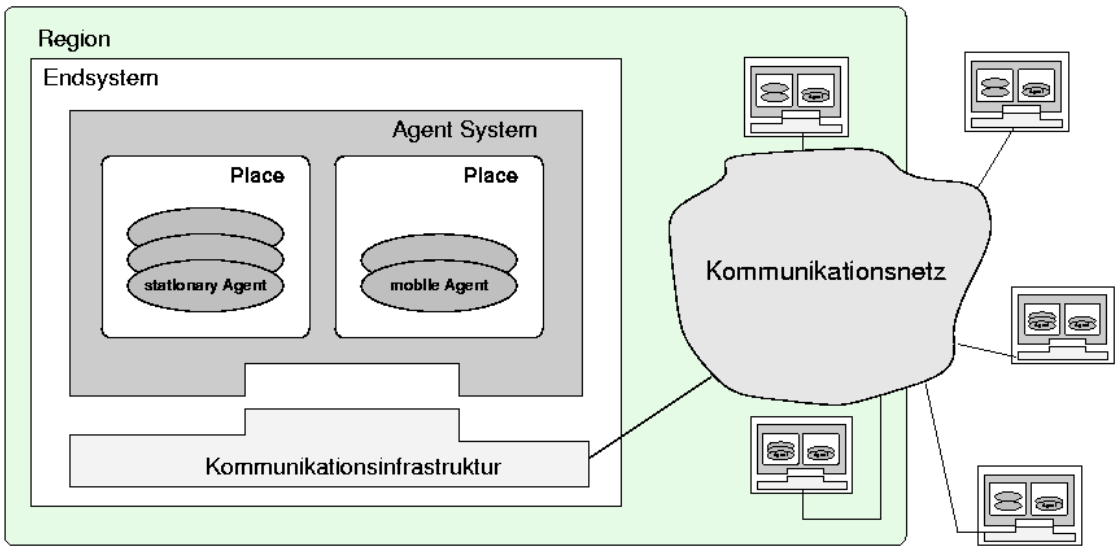


Abbildung 2.1: Agenten-System gemäß MASIF (aus [GHR 99])

Abbildung 2.1 zeigt ein Agenten-System (*Agent System*) gemäß MASIF-Spezifikation. Dabei ist ein *Agent* ein Computer-Programm zur Erfüllung einer bestimmten Aufgabe, das in einer eigenen Ausführungsumgebung, einem sog. *Place*, gestartet wird. Ein *Place* wiederum existiert innerhalb eines *Agent Systems*, das damit selbst die eigentliche Laufzeitumgebung für Agenten bildet und den gesamten Lebenszyklus der Agenten kontrolliert und verwaltet.

Es gibt sog. stationäre Agenten, die nur auf dem Agenten-System, auf dem sie gestartet wurden ablauffähig sind und mobile Agenten, die sich über die Kommunikations-Infrastruktur von Agenten-System zu Agenten-System fortbewegen können.

Agent und Agenten-System werden im Namen einer Person oder Organisation, einer sog. *Authority* ausgeführt, die sich auch unterscheiden können. Agenten-Systeme mit derselben *Authority* werden virtuell zu einer sog. *Region* zusammengefasst.

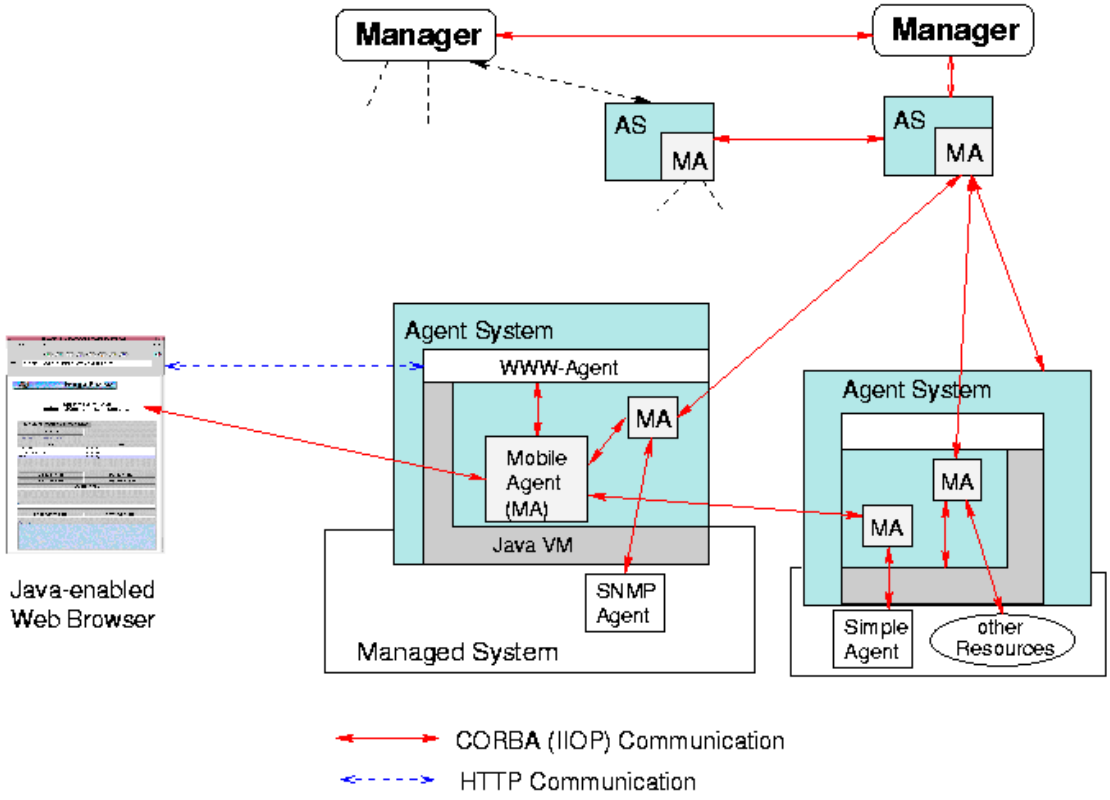


Abbildung 2.2: MASA-Architektur (aus [GHR 99])

Abbildung 2.2 zeigt die tatsächliche MASA-Architektur, die in [GHR 99] detailliert beschrieben wird. MASA ist vollständig in Java implementiert. *Agent* und *Agent System* laufen innerhalb einer *Java Virtual Machine* (JVM) auf einem zu managendem Endsystem. Die an der Kommunikation beteiligten Komponenten sind als CORBA-Objekte realisiert. Zum Einsatz kommt *ORBacus* ™, der *Object Request Broker* (ORB) der Firma OOC (Object Oriented Concepts, Inc.; s. [OOC 99]). Als Benutzeroberfläche für Agenten-System und Agenten dienen HTML-Seiten mit eingebetteten Applets zur Interaktion. Der Zugriff darauf erfolgt über einen Webserver, der als stationärer Agent auf jedem Agenten-System vorhanden ist.

In [Roel 99] wurde darüber hinaus ein Sicherheitsmodell für MASA definiert und umgesetzt, dass die Authentisierung von Personen (Benutzer und Implementierer), Agenten-Systemen und Agenten und die Kontrolle ihrer Zugriffe erlaubt.

2.2 Konkretisierung der Aufgabenstellung

Zu den sog. CORBA-Services, also Dienste, die im Prinzip jede CORBA-Implementierung zur Verfügung stellt, gehört auch ein Verzeichnisdienst, ein sog. *Naming Service*, der die Verbindung eines CORBA-Objektes bzw. seiner global eindeutigen IOR (*Interoperable Object Reference*) mit einem lesbaren Klartext-Namen erlaubt. Mit diesem *Naming Service* können CORBA-Objekte registriert, wieder ausgetragen und damit lokalisiert werden.

Bei der Implementierung des MASA-Basissystems wurde deshalb und aus Gründen der Vereinfachung der bereits existierende *Naming Service* der CORBA-Implementierung verwendet. Die Schnittstellen-Definition des *Naming Service* (s. [OMG 98-10-11]) bietet allerdings nur sehr einfache Methoden, wie `bind()`, `unbind()`, `resolve()` und `list()`, zum „Binden, Entbinden, Auffinden und Auflisten" von Objekten mittels ihrer Namen. Intelligentere Suchen nach z.B. Objekten mit bestimmten Eigenschaften sind damit nicht möglich.

Die MASIF-Spezifikation der OMG definiert für Agenten-Systeme das Interface *MAFFinder* (s. Anhang B), das die Methoden für die Registrierung, Löschung und Lokalisierung für Agenten, Agenten-Systeme und *Places* standardisiert. Die Methoden zur Lokalisierung sind dabei so definiert, dass die Suche nach Objekten mit bestimmten Eigenschaften über ein Suchprofil ermöglicht wird. So kann z.B. ein „migrations-williger" Agent ein für ihn passendes Agenten-System ermitteln, das es ihm erlaubt, den Transfer zu diesem Agenten-System durchzuführen und seine volle Funktionsfähigkeit zu entfalten.

Das wichtigste Ziel, das mit MASIF verfolgt wird, ist die Interoperabilität zwischen Agenten-Systemen. Der Weg dorthin wird vor allem auch durch die Anwendung einer konsistenten Terminologie erreicht. Eines der Basiskonzepte von MASIF ist deshalb u.a. die Standardisierung der Syntax und Semantik der Namen von Agenten und Agenten-Systemen sowie der sog. *Location*, also dem Ort, an dem sich der Agent bzw. das Agenten-System befindet. Da diese Parameter gerade für einen Verzeichnisdienst von zentraler Bedeutung sind und dessen konstruktiven Aufbau beeinflussen, sind sie als Anforderungen im Hinblick auf ihre Zusammenhänge und Konsequenzen eingehend zu betrachten.

Des weiteren ist es in MASA derzeit nicht möglich, Agenten-Instanzen der gleichen Agenten-Gattung auf einem Agenten-System zu starten, da ein Agent mit dem Namen der entsprechenden Agenten-Gattung instanziiert wird.

(*Anmerkung*: Der Begriff der Agenten-Gattung wurde in [Roel 99] eingeführt und bezeichnet alle statischen Eigenschaften, die bei Agenteninstanzen einer Gattung identisch sind. Der Gattungsbegriff kann mit dem Klassenbegriff der objektorientierten Programmierung verglichen werden, der eine Klasse von Objekten (Instanzen) mit gemeinsamen Attributen zusammenfasst.)

Die genannte Beschränkung wurde mit [Kemp 98] vordringlich zur Vereinfachung des Aufrufes bestimmter Agenten eingeführt. Damit können Agenten, wie z.B. der Webserver-Agent, auf jedem Agenten-System unter demselben Namen angesprochen werden. Darüber hinaus sollen so Konfliktfälle für exklusive Agenten, Agenten von denen nur eine Instanz im Netz existieren darf, zu erkennen sein.

Für bestimmte Management-Aufgaben kann es manchmal notwendig und sinnvoll sein, mehrere Instanzen derselben Agenten-Gattung gleichzeitig ihre Aufgaben auf einem Agenten-System erledigen zu lassen. Um die genannten Einschränkungen in MASA beseitigen zu können und damit multiple Agenten-Instanzen einer Gattung auch auf einem Agenten-System zuzulassen, soll deshalb ein neues Konzept für die Benennung von Agenten überlegt werden.

Um für MASA die vollständige Kompatibilität zum MASIF-Standard zu gewährleisten, wird nun als *Naming Service* für MASA die *MAFFinder*-Schnittstelle unter Berücksichtigung der vorstehenden Überlegungen implementiert.

3 Spezifikation der Anforderungen

Die nachfolgenden Anforderungen berücksichtigen die MASIF-Spezifikation ([OMG 98-03-09]), Anforderungen aus der bestehenden MASA-Implementierung sowie grundsätzliche Überlegungen.

Zur besseren Verfolgbarkeit der Anforderungen während Entwurf und Implementierung werden sie mit einer Quellangabe und einer fortlaufenden Nummer gekennzeichnet (<Quelle>-<Nummer>).

3.1 MAFFinder Spezifikation

Wie in [Kemp 98] bereits ausgeführt, gewährleistet die Anwendung des OMG-Standards MASIF Interoperabilität zwischen verschiedenen Agenten-Systemen. Die Anforderungen aus der Spezifikation der *MAFFinder*-Schnittstelle sind deshalb so vollständig wie möglich zu erfüllen. Im Folgenden werden die für die Implementierung der Schnittstelle relevanten Anforderungen aus der MASIF-Spezifikation [OMG 98-03-09] zusammengestellt.

3.1.1 Terminologie und Namens-Konzept

Zum Erreichen von Interoperabilität sind – wie bereits erwähnt - die Einhaltung von Syntax und Semantik der Namen von Agenten und Agenten-Systemen sowie der zugehörigen *Location* als das „Basiskonzept“ von MASIF zwingend vorgeschrieben.

Gemäß MASIF besteht der Name eines Agenten bzw. Agenten-Systems aus den drei Attributen *Authority* (Person oder Organisation, die das Objekt repräsentiert; bedeutend auch im Zusammenhang mit den Sicherheitseigenschaften), *Agent System Type* (Zahl zur Typisierung des zugrundliegenden Agenten-Systems) und *Identity* (eigentlicher Name des Objektes). Die *Identity* muss dabei eine Instanz eines Agenten/Agenten-Systems innerhalb einer spezifischen Authority eindeutig identifizieren. Die Kombination aus *Authority*, *Agent System Type* und *Identity* muss einen global eindeutigen Namen für einen Agenten, ein Agenten-System im Kontext von *Mobile Agent Facilities* (MAF) ergeben.

MASIF-01:
Die Benennung von Agenten und Agenten-Systemen besteht aus den drei Komponenten Authority, Agent System Type und Identity. Die Identity muss Objekte innerhalb einer Authority eindeutig unterscheiden. Der aus den drei Komponenten zusammengesetzte Name muss Objekte global eindeutig unterscheiden.

Teil des Basiskonzeptes von MASIF sind die Einbettungsbeziehungen. Wie oben bereits erläutert ist die Ausführungsumgebung für einen Agenten ein sog. *Place*. Ein *Place* kann mehrere Agenten enthalten und deren Manipulation (z.B. Zugriffskontrolle) ermöglichen. Ein oder mehrere *Places* sind eingebettet in das Agenten-System. Sieht ein Agenten-System keinen *Place* vor, so ist ein sog. *Default-Place* anzunehmen. Das Agenten-System ist wiederum eingebettet in ein Endsystem (s. auch Abb. 2.1).

MASIF-02:
Es ist folgende Einbettungsbeziehung durchzusetzen:
Ein Endsystem enthält ein oder mehrere Agenten-Systeme. Ein Agenten-System enthält einen oder mehrere Places, mindestens den sog. Default-Place. Ein Place ist die Ausführungsumgebung eines oder mehrerer Agenten.

Nach MASIF ist *Place* immer mit einer sog. *Location* zu assoziieren. Eine *Location* bezeichnet einen Ort bzw. eine Adresse. Sie besteht in der Regel immer aus dem Namen eines Places und der Adresse eines Agenten-Systems. Der Name eines Places ist hinsichtlich seiner Struktur nicht näher spezifiziert.

MASIF-03:
Ein Place besitzt einen (hinsichtlich seiner Struktur nicht näher spezifizierten) Namen.

MASIF-04:
Eine Location spezifiziert einen Place und die Adresse des zugehörigen Agenten-Systems.

Die *Location* eines Agenten bezeichnet somit die Adresse eines *Places*. Sie besteht aus dem Namen (oder auch Pfad) eines Agenten-Systems und eines *Places*. Fehlt die Angabe eines Place-Namens, ist ein sog *„Default"-Place* auszuwählen (wird u.U. bei der Migration vom Ziel-Agenten-System vergeben).

MASIF-05:
Die Location eines Agenten bezeichnet die Adresse (Pfad) bzw. den Namen eines Agenten-Systems und eines Places (bzw. Default) auf dem der Agent gestartet wurde.

MASIF baut selbstverständlich auf den *CORBA-Standards* ([OMG 99-10-07]) auf und enthält Hinweise auf die Anwendung der *CORBA-Services,* u.a. den CORBA *Naming Service*.

Der CORBA *Naming Service* verbindet CORBA-Objekte mit ihren Namen. Diese Assoziation wird *Name Binding* genannt, das sich immer auf einen bestimmten Namens-Kontext (*Naming Context*) bezieht. Ein *Naming Context* enthält eine Anzahl von *Name Bindings,* dabei muss jeder Name innerhalb des Kontextes einmalig sein. Die Namens-Kontexte lassen sich zu einem sog. *Naming Graph* oder „Namensbaum" verbinden. Namens-Kontexte bilden die Knoten, CORBA-Objekt-Referenzen die Blätter dieses gerichteten Graphen. Ein bestimmtes Objekt kann also über die Angabe einer Reihenfolge von Namen (Kontexte) innerhalb des Namensdienstes aufgefunden werden. Diese Aneinanderreihung von Namen wird *Compound Name* des Objektes genannt.

(Anmerkung:
Zur Veranschaulichung kann ein *Naming Graph* auch mit einem Verzeichnisbaum für z.B. Dateien verglichen werden. Anstelle der Dateien werden darin Referenzen auf CORBA-Objekte anhand ihres Namens über einen korrespondierenden Namens-Pfad auffindbar gemacht.)

Der *Compound Name* spezifiziert einen bestimmten Pfad im *Naming Graph* und ist sozusagen mit der Adresse des Objektes im Namensverzeichnis vergleichbar. Die Korrespondenz des *Compound Names* zur *Location* (s. oben) ist damit offensichtlich.

In der Nomenklatur des CORBA *Naming Services* lässt sich der *Compound Name* eines Agenten oder Agenten-Systemes also aus der Kombination der einzelnen Namens-Komponenten *Authority, Agent System Type* und *Identity* bilden. Syntaktisch definiert MASIF eine Namens-Komponente jeweils als ein Objekt vom Typ `CosNaming.Name`. Die darunter liegende Datenstruktur besteht dabei selbst wieder aus den beiden Attributen `Id` und `kind` (s. [OMG 98-10-11]).

MASIF-06:
Der Compound Name eines Agenten/Agenten-Systems ist die Kombination der einzelnen Namens-Komponenten Authority, Agent System Type und Identity. Die einzelne Namens-Komponente ist jeweils ein Objekt vom Typ `CosNaming.Name`.

3.1.2 Anforderungen zur Location

Aufgrund der Bedeutung zum Auffinden von Objekten im *Naming Graph*, muss der Semantik der *Location* besondere Beachtung geschenkt werden.

MASIF definiert *Location* als Datentyp `String`. Dieser String kann zwei Formen von Adressangaben enthalten:

- Eine **URI** (Uniform Resource Identifiers): CosName.Name-Format
- Eine **URL** (Uniform Resource Locators): IIOP-Adresse (IIOP: Internet Inter-ORB Protocol)

In beiden Fällen sieht die Spezifikation vor, dass der Client zuerst das entsprechende Format erkennen muss. Dann kann er daraus den Namen des Agenten-Systems aufbauen (URI) und damit erst die Objekt-Referenz für das Agenten-System beim *Naming Service* erfragen oder direkt die IOR des Agenten-Systems ableiten (URL).

MASIF-07:
Die Location ist vom Datentyp `String` *und enthält eine URI oder URL.*

3.1.3 Anforderungen an die Architektur des MAFFinder's

Der CORBA *Naming Service* wurde laut MASIF mehr für statische Objekte konzipiert und weniger für die Handhabung migrierender Objekte wie z.B. mobile Agenten. Zu diesem Zweck wurde das Interface *MAFFinder* als Schnittstelle einer „dynamischen" Datenbank für Namen und Adressen spezifiziert.

Gemäß Spezifikation kann ein *MAFFinder* mehrere *Regions* bedienen. Es kann aber auch einen oder mehrere *MAFFinder* pro *Region* geben.

MASIF-08:
Unterschiedliche Regions können sich einen MAFFinder teilen. Zur Vereinfachung gibt es aber mindestens einen MAFFinder pro Region.

Konzeptionell kann der *MAFFinder* zusammen mit dem Agenten-System in einem gemeinsamen oder in separaten Software-Modulen implementiert werden.

Die Spezifikation überlässt es der Implementierung, ob ein *MAFFinder*-Objekt seine Verfügbarkeit anderen *MAFFinder*-Instanzen mitteilt.

MASIF definiert die *MAFFinder*-Schnittstelle als *CORBA-Object-Interface*. Je nach Implementierung kann die Referenz auf das *MAFFinder*-Objekt im CORBA *Naming Service* veröffentlicht werden, damit Agenten darauf zugreifen können.

MASIF-09:
Das MAFFinder-Objekt kann – muss aber nicht - im CORBA Naming Service eingetragen werden.

Bevor die *MAFFinder*-Methoden genutzt werden können, muss ein Client eine entsprechende Objekt-Referenz erhalten. Dies geschieht entweder über den CORBA *Naming Service* oder über die Methode `get_MAFFinder()` eines *Agent Systems*.

MASIF-10:
Eine Referenz für ein MAFFinder-Objekt erhält ein Client entweder vom CORBA Naming Service oder über die Methode `get_MAFFinder()` *eines Agent Systems.*

3.1.4 Das Interface MAFFinder

Das *MAF IDL Interface* definiert die Schnittstellen *MAFAgentSystem* und *MAFFinder*. Da vornehmlich das Agenten-System dafür verantwortlich ist Interoperabilität zu gewährleisten, definiert MASIF die Schnittstellen auf Agenten-System-Ebene und nicht auf Agenten-Ebene.

Das *MAFFinder*-Interface stellt Operationen zur Registrierung, Löschung und Lokalisierung von Agenten, *Places* und Agenten-Systemen bereit.

Anmerkung:
Diese sollen eine Anwendung (*Client*) bei der Suche unterstützen, nicht aber deren übergeordnete Such-Methodik einschränken. So kann ein *Client* zur Lokalisierung eines Agenten unterschiedliche Vorgehensweisen implementieren, wie z.B.:

- *Brute Force Search*, d.h. anhand einer Liste der Adressen der existierenden Agenten-Systeme führt ein mobiler Agent die Suche nach einem bestimmten Agenten durch;
- *Logging*, d.h. ein Agent hinterlässt bei der Migration seine Zieladresse beim Agenten-System;
- *Agent Registration*, d.h. der Agent lässt seine aktuelle Position in einer Datenbank registrieren;
- *Agent Advertisement*, d.h. es können nur Agenten geortet werden, die ihre Adresse „veröffentlicht“ haben.

Diese Möglichkeiten nennt MASIF als Beispiele, die auf die *MAFFinder*-Methoden zurückgreifen können.

Das Interface *MAFFinder* ist zusammen mit den entsprechenden Datentypen, Ausnahmefällen (*Exceptions*) und dem Interface *MAFAgentSystem* im Modul *CfMAF* als IDL-Schnittstelle *(Interface Definition Language)* definiert und im Anhang B einsehbar.

MASIF-11:
Implementierung des IDL-Interfaces MAFFinder.

3.1.5 Die Suchstrategien des MAFFinder's

Durch die Realisierung der Methoden `lookup_agent()` und `lookup_agent_system()` ermöglicht der *MAFFinder* gegenüber dem CORBA *Naming Service* wesentlich komfortablere Suchmöglichkeiten.

Diese Methoden enthalten als Suchfilter neben dem Namen (Parameter `agent_name` bzw. `agent_system_name`) als Parameter ein Profil (`agent_profil` bzw. `agent_system_info`), die bestimmte Eigenschaften des zu suchenden Agenten oder des zu suchende Agenten-Systems näher spezifizieren. Syntax und Semantik der Parameter *Agent Profile* und *Agent System Info* sind der IDL-Beschreibung des MAFFinders (s. Anhang B) zu entnehmen.

So enthält *Agent System Info* Name und Typ des Agenten-Systems, eine Liste mit unterstützten Programmiersprachen und Serialisierungs-Methoden (zur Migration und Ausführung von Agenten), eine Text-Beschreibung des Systems, die maximal und minimal unterstützte Version des Agenten-Systems (zur Kompatibilitätsentscheidung) sowie eine Liste nicht näher spezifizierter sog. *Property*-Objekte, die der jeweiligen Implementierung des Agenten-Systems überlassen bleibt.

Korrespondierend dazu enthält *Agent Profile* ebenso den Typ des Agenten-Systems, auf dem der Agent erstmals gestartet wurde, die Implementierungssprache mit zugehöriger Serialisierungsmethode, maximale und minimale Agenten-System-Version, auf der der Agent lauffähig ist, sowie die Liste der anwendungsspezifischen *Properties*.

Eine Suche kann nach dem Namen, einem entsprechenden Eigenschafts-Profil oder einer Kombination aus beiden aufgesetzt werden. Damit kann alternativ nach Agenten/Agenten-Systemen einer bestimmten *Authority*, eines bestimmten Typs, eines bestimmten Namens oder mit bestimmten Eigenschaften und entsprechenden Kombinationen gesucht werden. Die Suche nicht einschränkende Parameter-Bestandteile sind entweder zu „0“ (bei Datentyp `Integer`) oder als leerer `String` anzugeben. Der Einfluss einer intelligenten Namensgebung auf Suchergebnisse und Anwendungsmöglichkeiten wird auch hier wieder offensichtlich.

Als Rückgabewert erhält man eine Liste mit Adressen (der *Location*) von Objekten, auf die das Suchprofil zutrifft. Die Methoden können gemäß Spezifikation allerdings nicht garantieren, dass sich der oder die Agenten bzw. Agenten-Systeme zu einer bestimmten Zeit noch unter der gefundenen Adresse aufhalten.

MASIF-12:
Die `lookup_agent()`*-Methode bzw.* `lookup_agent_system()`*-Methode ist so zu implementieren, dass nach beiden Suchparametern gleichzeitig oder nach einem gesucht werden kann. Nicht einschränkende Parameter oder deren Bestandteile sind mit „0“ oder mit einem leeren String zu belegen.*

3.1.6 Anforderungen an das Agenten-System selbst

Grundsätzlich muss ein Agenten-System nach MASIF einen Namensdienst bereitstellen, um mobile Agenten lokalisieren und auffinden zu können und die Suche nach Kommunikations- und Migrationspartnern unterstützen zu können.

Das Interface *MAFAgentSystem*, das von einem Agenten-System zu implementieren ist, stützt sich bei folgenden Methoden auf das Interface *MAFFinder* ab:

Die Methode `get_MAFFinder()` liefert eine Referenz auf das MAFFinder-Objekt.

MASIF-13:
Implementieren der Methode `get_MAFFinder()` *im Agent System.*

Über die Methode `find_nearby_agent_system_of_profile()` findet ein Agent ein für seine Ausführung geeignetes Agenten-System. Will ein Agent mit einem Objekt kommunizieren, das sich auf einem inkompatiblen Agenten-System oder einem Endsystem ohne Agenten-System befindet, kann er z.B. über diese Methode ein geeignetes Agenten-System mit kompatiblem Typ suchen. Die Methode nutzt die entsprechende `lookup()`-Methode des *MAFFinders*.

MASIF-14:
Implementierung der Methode `find_nearby_agent_system_of_profile()` *unter Berücksichtigung der entsprechenden MAFFinder-Methoden im Agent System.*

Die Methode `list_all_places()` listet alle zur Verfügung stehenden Ausführungsumgebungen für Agenten, die *Places,* eines *Agent Systems* auf und kann dazu den *MAFFinder* nutzen.

MASIF-15:
Implementierung der Methode `list_all_places()` *unter Berücksichtigung der entsprechenden MAFFinder-Methoden im Agent System.*

3.2 Analyse MASA

Die Umsetzung der *MAFFinder*-Schnittstelle bedeutet einen nicht unwesentlichen Eingriff in das bestehende System, da dessen Integrität auf einem funktionierenden Namensdienst basiert. Durch ein *Code Review* ist zu analysieren an welchen Stellen und inwieweit dadurch Änderungen erforderlich werden. Besonderes Augenmerk ist auf die Namensgebung und das *Name Binding* zu richten, da Konzeption und Funktionsfähigkeit der *MAFFinder*-Schnittstelle auf dem Basiskonzept der MASIF-Spezifikation beruhen und indirekt die Interoperabilitätseigenschaften betroffen sind.

3.2.1 Der bestehende Naming Graph von MASA

Abbildung 3.1 zeigt die in der MASA-Implementierung umgesetzte Namens-Hierarchie (*Naming Graph*). Der MASA-*Naming Graph*, der mit [Kemp 98] eingeführt wurde, nutzt die in [OMG 98-10-11] spezifizierten Datentypen und Methoden zur Erstellung von

Namens-Kontexten, zum Binden, Lösen und Auslesen von CORBA-Objektreferenzen und Kontexten anhand ihrer Klartext-Namen im CORBA *Naming Service*.

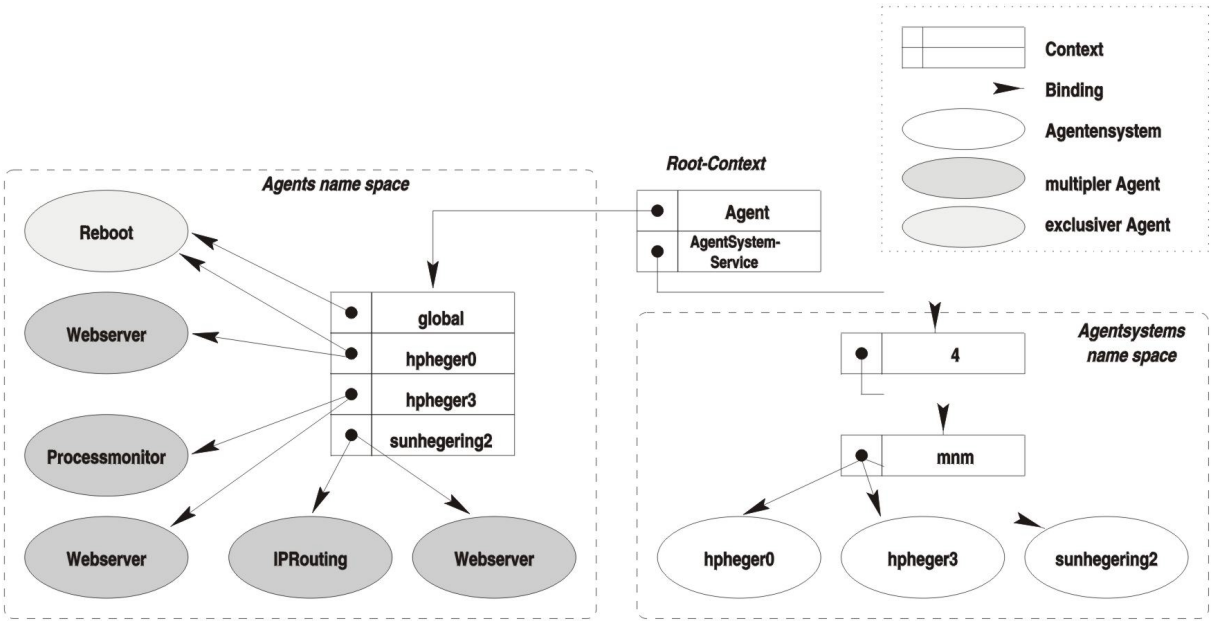


Abbildung 3.1: MASA Name Graph (aus [Kemp 98])

Die Wurzel des MASA *Naming Graph*, der sog. *Root Context*, ist die Referenz auf den CORBA *Naming Service*. Darunter realisiert MASA zwei separate Namensbäume für Agenten (*Agent Context*) und Agenten-Systeme (*AgentSystemService Context*). In *AgentSystemService* werden die Referenzen auf Agenten-Systeme anhand ihres *Compound Names* eingetragen. Der *Agent Context* gliedert sich in den Sub-Kontext *global* und jeweils einen Kontext mit der Bezeichnung des Agenten-Systems, das den einzutragenden Agenten ausführt. Im *global Context* werden die Referenzen von sog. globalen oder exklusiven Agenten, von denen nur eine Instanz im gesamten Netz existieren darf, zusätzlich eingetragen.

Die Agenten im Teilbaum *Agent* können somit nicht anhand ihres *Compound Names* (*Authority, Agent System Type, Identity*) identifiziert werden. Wie in [Kemp 98] beschrieben werden sie lediglich über ihre *Identity* im Kontext, der ihrem Ausführungsort (*Location*) entspricht, eingetragen.

Als Begründung für diese Ausprägung des *Naming Graphen* wird in [Kemp 98] die feste Zuordnung der *Identity* zu einer Klasse bzw. Agentengattung angegeben, um hauptsächlich gleiche Namen für Agenten desselben Typs durchsetzen zu können und evtl. die Konfliktfälle für exklusive Agenten erkennen zu können. Wie bereits erwähnt, dient die erste Anforderung zum gleichlautenden Ansprechen von Agenten, wie z.B. des Webserver-Agenten, auf allen Agenten-Systemen. Die zweite Anforderung wird vordringlich durch die zusätzliche Eintragung im *Context global* erreicht.

Daneben wurde von [Kemp 98] aus Gründen der Vereinfachung auch auf die Einführung von *Locations, Places* und *Regions* verzichtet. Die Semantik der *Location* ist für [Kemp 98] nicht erforderlich, da zur Lokalisierung der CORBA *Naming Service* genutzt wird und ein CORBA-Objekt über seine IOR (*Interoperable Object Reference*) eindeutig festgelegt

ist. *Places* als eigene Ausführungsumgebung von Agenten innerhalb eines Agenten-Systems werden nicht definiert und könnten ohne die Einführung des *Location*-Begriffes auch nicht lokalisiert werden. Die nach MASIF über *Places* durchzusetzenden Zugriffsrechte von Agenten werden anderweitig realisiert. Eine Gruppierung von Agenten-Systemen nach *Regions* wird nicht explizit vorgenommen.

Die flache Kontext-Hierarchie des MASA *Naming Graph* ist für eine einfache Handhabung mit den zur Verfügung stehenden Methoden der CORBA-Implementierung prädestiniert. Die von der MASIF-Spezifikation vorgegebene Namens- und Instanzen-Hierarchie wird aber nicht vollständig unterstützt.

MASA-01:
Intuitiver Aufbau des Naming Graphen, so dass das Konzept der Regions, Places und der Namens-Bestandteile von Agent System und Agent genutzt werden kann.

3.2.2 Name Binding in MASA

Das *Name Binding* bezeichnet die Assoziation eines CORBA-Objektes mit seinem Namen. Um Fragen zur Rückwärtskompatibilität und später zur Namensgebung multipler Agenten-Instanzen klären zu können, wurde das *Name Binding* der CORBA-Objekte in MASA überprüft.

Wie bereits dargestellt, wird ein Objekt (egal ob Kontext oder CORBA-Objekt) mittels seiner *Name Component* in den jeweiligen *Context* eingebunden. Eine *Name Component* ist wie folgt definiert (s. [OMG 98-10-11]):

```
NameComponent ::= {(<Id>, <kind>)}
```

In MASA wird das `kind`-Attribut der *Name Component* bislang nicht belegt, dafür wird ein leerer `String` eingetragen. Als `Id`-Attribut wird nach [Roel 99]in der Regel folgender Name zur Registrierung von Agenten und Agenten-Systemen verwendet:

```
Id ::= [<identity>!<authority>!<type>]
```

Bei Nutzung des vollständigen *Compound Names* nach MASIF, d.h. bei einem wie unter Kapitel 3.1 beschriebenen aufeinander aufbauenden *Name Graph* mit den „Ästen“ *Authority, Agent System Type* und *Identity,* würde für das *Name Binding* auf „Blatt-Ebene“ die *Identity*-Komponente zur Eindeutigkeit genügen.

Ein *Agent System* wird also heute in MASA mit folgender *Name Component* gebunden:
`{([<identity>!<authority>!<type>], "")}`
Beispiel:
`{([pcheger0.nm.informatik.uni-muenchen.de!UID:gigl!4], "")}`

Agenten werden im zugehörigen Agenten-System-Kontext wie folgt eingetragen:
`{([<identity>!<authority>!<type>], "")}`
Beispiel:
`{([Webserver!UID:gigl!4], "")}`

Agenten werden im *global Context* wie folgt eingetragen:
`{(<package>"."<identity><Agententyp>, "")}`

Beispiel:
`{([de.unimuenchen.informatik.mnm.masa.Webserver!UID:gigl!4], "")}`

MASA-02:
Das Name Binding muss sich zur Vereinfachung am Namenskonzept von MASIF orientieren. Das bestehende MASA Name Binding ist bei der Implementierung eines neuen Name Bindings auf Auswirkungen hinsichtlich der Rückwärtskompatibilität zu prüfen und bei einem neuen Konzept für die Benennung von Agenten-Instanzen zu beachten.

Bezugnehmend auf die in Kapitel 3.1.4 vorgestellten Such-Methoden einer Anwendung zum Auffinden von Agenten ist anzumerken, dass MASA die Methode *Agent Registration* unterstützt, da über das Name Binding alle CORBA-Objekte im *Naming Service* registriert werden.

MASA „Secure-Version"

MASA kann ohne die mit [Roel 99] eingeführten Sicherheitseigenschaften (s. Kapitel 3.2.5) oder mit ihnen in der sog. *Secure Version* betrieben werden. In der *Secure Version* wird die *Authority* des Namens durch ein Zertifikat ersetzt. Im Falle des Agenten-Systems ist es das Zertifikat des Anwenders. Im Falle eines Agenten ist es ebenfalls das Zertifikat des Anwenders oder ein vom Agenten-System erstelltes Sitzungszertifikat, das den kompletten Namen sowie Typ- und Versionsangaben des Agenten-Systems enthält, z.B.:
`[pcheger0.nm.informatik.uni-muenchen.de!Test Admin!4](MASA Agent System V0.4)`

Die Belegung der *Authority* des Agenten ist somit nicht einheitlich, sondern abhängig von wem der Agent gestartet wurde. Wurde der Agent automatisch vom Agenten-System gestartet, wie z.B. die Agenten *Webserver* und *ASManagementAgent* beim Start des Agenten-Systems, so gilt als Authority das Agenten-System selbst und trägt das oben vorgestellte Sitzungszertifikat ein. Wurde der Agent vom Anwender gestartet, so wird das Anwender-Zertifikat (z.B. `Test Admin`) eingetragen. In der Version ohne Sicherheitseigenschaften wird die *Authority* beim Agenten mit `NO_CERT_PRESENT`, beim Agenten-System mit `UID:`<Anwender> belegt.

Da die Belegung der *Authority*-Komponente natürlich Auswirkungen auf die Eintragung des Agenten im *Naming Graph* besitzt, ist dieses bei der späteren Realisierung eines neuen Namenskonzeptes zu berücksichtigen und zu überprüfen.

MASA-03:
Die Belegung der Authority-Komponente von Agenten ist auf Durchgängigkeit und Anwendbarkeit bei der Umsetzung eines neuen Name Bindings zu überprüfen.

Kommunikation Applet - Agent

Für die Realisierung multipler Agenten-Instanzen ist neben dem eigentlichen *Name Binding* auch der Aufruf der Benutzoberfläche sowie die Kontaktherstellung zwischen Benutzoberfläche bzw. Applet zum CORBA-Objekt, insbesondere dem Agenten zu betrachten.

Zum Aufruf der Benutzeroberfläche eines Agenten/Agenten-Systems ist die URL des zugehörigen Applets von Interesse. Auch sie enthält zur Identifikation den Namen des jeweiligen Objektes.
Die URL eines Applets setzt sich wie folgt zusammen:

Für ein Agenten-System:
`http://<hostname>:<port>`
Beispiel:
`http://pcheger0.nm.informatik.uni-muenchen.de:4197`

Für einen Agenten:
`http://<hostname>:<port>/[<identity>!<authority>!<type>]/agentapplet`
Beispiel:
`http://pcheger0.nm.informatik.uni-muenchen.de:4197/[Webserver!UID:gigl!4]/agentapplet`

Die Kontaktaufnahme des Applets zu seinem Objekt (hier: die Agenteninstanz) erfolgt in der Methode `init()` der Klasse `AgentApplet` über die Methode `connectToObject()`. Über `getIOR()` wird darin zunächst die IOR des Agenten vom stationären *Webserver*-Agenten erfragt. Die Klasse `RequestHandler` des *Webservers* wendet sich an das lokale *Agent System* und erfragt über `getLocalAgentIOR_byString()` die zugehörige IOR. Das *Agent System* leitet diese Anfrage an seinen *Agent Manager* weiter, der schließlich die CORBA-Objekt-Referenz des Agenten aus seiner *Hash*-Tabelle `AgentTable` zurückgibt. Der Suchparameter ist der Name des Agenten (als *Compound Name* oder als *String*).

Den Namen erhält das Applet in seiner `init()`-Methode über den Methodenaufruf `getParameter(masa.param.myAgent)`. Mit `getParameter()` erfolgt der Zugriff auf die Homepage des Agenten. Dort ist der „stringifizierte“ Name als `VALUE` hinterlegt (`inObjectName`):

```
"<PARAM NAME=\""+AgentAppletConstants.myAgentParam+"\"
VALUE=\""+inObjectName+"\">"
```

Die HTML-Seite wird vom `RequestHandler` des *Webservers* dynamisch auf Anfrage mit der Methode `writeHomePageReply()` und dem Namen als Eingangsparameter erzeugt.

Der Name des betreffenden Agenten wird über die Webseiten des Agenten-Systems an den `RequestHandler` weitergegeben. Die Webseite des Agenten-Systems besitzt u.a. die beiden Register *CreateAgent* und *ManageAgent*. Für das Register *CreateAgent* stellt die Klasse `AgentSystemApplet` über die in ihr definierte Klasse `AgentTypesTableModel` die auf dem Agenten-System verfügbaren Agenten-Gattungen (Namen der Agenten-Klassen) tabellarisch zur Auswahl frei. Die Namen werden vom Agenten-System mit `getAvailableAgentKinds()` erhalten. Angezeigt wird der vollständige *Package* Name, z.B.:
`de.unimuenchen.informatik.mnm.masa.agent.Webserver.WebserverStationaryAgent`

Bei Auswahl eines Agenten wird in der Klasse `AgentSystemApplet` über die Methode `initComponents()` zum Start des Agenten durch `_createAgent_btActionPerformed()` die bekannte `create_agent()`-Methode des Agenten-Systems aufgerufen. Als Aufrufparameter wird als Name für den Agenten der Klassen-Name eingetragen.

Das Register *ManageAgent* zeigt eine Liste der auf dem Agenten-System gestarteten Agenten. Diese Liste wird über die in `AgentSystemApplet` definierte Klasse `ActiveAgentsTableModel` bereitgestellt. Sie wird über die Methode `fetchActiveAgents()` erzeugt, die auf `list_all_agents()` aus `AgentSystem` zurückgreift. Die Namen, wiederum die Klassen-Namen, werden dann aus dem *Hashtable* (`class AgentTable`) von `AgentManager` ausgelesen.

Bei Auswahl eines der angezeigten Agenten kann dann über einen Button z.B. die Hompage des Agenten aufgerufen werden. Dazu wird in der Klasse `AgentSystemApplet` über die Methode `openAgentWebpage()` mit dem ausgewählten Agenten-Namen zunächst die URL des Agenten erfragt und dann über Methoden der Java-API-Klasse Applet die HTML-Seite in einem eigenen Browser-Window angezeigt. Der Kommunikationsaufbau erfolgt wie oben beschrieben.

Entscheidend auch für den Kommunikationsaufbau zwischen Applet und Agenten ist also der Name des Agenten, der bei dessen Erzeugung auf dem Agenten-System festgelegt wird. Ein Agent wird auf einem Agenten-System grundsätzlich über drei Arten erzeugt:

- vom Anwender durch Starten über das Register *CreateAgent* wie oben beschrieben
- vom Agenten-System automatisch nach einer Migration (keine Namensfestlegung, da als Parameter mitgegeben)
- vom *Agent Manager* beim Hochlauf des Agenten-Systems, der die im *Autodescription*-File hinterlegten Agenten automatisch startet; es wird wiederum der Klassen-Name verwendet

Bei der Realisierung multipler Agenten-Instanzen muss also bei der ersten und letzten Erzeugungsart die Trennung zwischen Gattung und Instanz eines Agenten bei der Namensfestlegung berücksichtigt werden.

MASA-04:
Bei einem neuen Konzept für die Benennung von Agenten-Instanzen ist die Adressierung von Applets und die Kommunikation zu seinem Agenten zu beachten. Bei der Erzeugung eines Agenten muss die Trennung zwischen Gattung und Instanz eines Agenten bei der Namensfestlegung beachtet werden.

3.2.3 Places in MASA

Das MASIF-Konzept der *Places*, als eigene Ausführungsumgebung von Agenten innerhalb eines Agenten-Systems mit der Möglichkeit zum Management (Zugriffsrechte, ...) der darin befindlichen Agenten, ist in MASA nicht realisiert. Nach [Roel 99] könnte als *Place* in MASA derzeit die *Thread Group*, unter der spezifische Agenten laufen, oder der Agenten-spezifische Namensraum, der durch den Java *Classloader* für jeden neuen Agenten erzeugt wird, verstanden werden. Eine explizite Ausprägung von *Places* existiert in MASA aber heute nicht.

Da *Places* als Ausführungsumgebung für Agenten ein wesentlicher Bestandteil des Basiskonzeptes von MASIF sind, sollte die *MAFFinder*-Implementierung die spätere Einführung von *Places* in MASA berüchsichtigen.

MASA-05:
Bei der Realisierung des MAFFinders ist das Vorhandensein von Places zu berücksichtigen.

3.2.4 Anpassung von Komponenten des MASA-Basissystems

Die Implementierung der MAFFinder-Schnittstelle hat Auswirkungen auf das Registrieren, Löschen und Suchen von Agenten-Systemen und Agenten anhand ihres Namens. Die bestehenden MASA-Komponenten sind daraufhin zu untersuchen und an die Verwendung der MAFFinder-Schnittstelle anzupassen.

Die wesentlichen Zusammenhänge zeigen die UML-Diagramme (*Unified Modelling Language*, s. [RSC 97]) aus [Kemp 98] und [Roel 99] sowie die Javadoc-Online-Dokumentation von MASA.

Die Klasse `AgentSystem` baut in ihrem Konstruktor und ihrer `main()`-Methode den *Naming Graph* für Agenten und Agenten-Systeme auf. Die Registrierung im *Naming Service* erfolgt in der `main()`-Methode, die Austragung aus dem *Naming Service* in der Methode `terminate_agent_system()`. Gemäß Kapitel 3.1.6 sind die Methoden `get_MAFFinder()`, `find_nearby_agent_system_of_profile()` und `list_all_places()` anzupassen. Darüber hinaus muss die Methode `getAgentSystemIOR()`, die zum Auslesen einer Objektreferenz auf den *Naming Service* zurückgreift, modifiziert werden.

Die Klasse `AgentManager`, die im *Package* `AgentSystem` enthalten ist und die Agenten ihres Agenten-Systemes managed muss lediglich in ihren Methoden `create_agent()` und `deserializeAgent()` im Zusammenhang mit der Prüfung auf Exklusivität eines Agenten umgestellt werden.

Die abstrakte Klasse `AgentEnvironment` liegt ebenfalls im *Package* `AgentSystem`. Sie bietet dem Agenten-System Schnittstellenfunktionen zur Kontrolle ihres repräsentierten Agenten an, der diese Klasse erbt. Alle Zugriffe des Agenten auf den *Naming Service* sind in den Methoden `bindAgentToNamingService()` und `unbindAgentFromNamingService()` gekapselt. In der Regel wird die *bind*-Funktion über die Methode `launchAgent()` und die *unbind*-Funktion über die Methode `terminateAgent_weak()` aufgerufen.

Der `ASManagementAgent` unterstützt die Benutzeroberfläche des Agenten-Systems. Für ihn ist die Methode `identify_agentsystems()` umzubauen, die eine Liste aktiver *Agent Systems* erzeugt.

MASA-06:
Die Komponenten des MASA-Basissystems sind im Hinblick auf die Nutzung der MAFFinder-Schnittstelle zu modifizieren.

3.2.5 Die MASA Sicherheitseigenschaften

Mit [Roel 99] wurde für MASA ein umfassendes Sicherheitsmodell eingeführt und u.a. Mechanismen zur Authentisierung von Entitäten, Autorisierung von Schnittstellen sowie zur Sicherung der Kommunikations-Kanäle bereitgestellt.

Bei der Implementierung des *MAFFinders* darf das MASA-Sicherheitskonzept nicht gefährdet werden. In [Roel 99] werden entsprechende Richtlinien zur Agenten-implementierung angegeben, die auch auf die Implementierung des *MAFFinders* übertragen werden können. Die wichtigsten sind hier kurz zusammengefasst:

Nach [Roel 99] sind deshalb die CORBA-Methoden, die als öffentlich zugängliche Methoden des *MAFFinders* über das *IDL-Interface* definiert und als `public` deklariert sein müssen, zu Beginn der Methode über den sog. *Permission Manager* (s. [Roel 99]) zu autorisieren. Grundsätzlich sind alle Methoden und Attribute die nur innerhalb einer Klasse Verwendung finden als `private` zu deklarieren. Öffentlich benötigte Attribute sind über Zugriffsmethoden (`get()-/set()`-Methoden) bereitzustellen. Nicht zu vererbende Klassen sind als `final` zu deklarieren.

MASA-07:
Einhaltung der MASA Sicherheitseigenschaften gemäß der Richtlinien nach [Roel 99] und insbesondere Einhaltung des Konzeptes zur Anwendung der Java Sichtbarkeits-modifikatoren.

3.2.6 Tools zur Namens-Manipulation in MASA

Mit den Klassen `NameSpace` und `NameWrapper` stehen unter MASA Werkzeuge zur Erzeugung von *Name Components* und zur Manipulation von Namen zur Verfügung.

In der Klasse `NameSpace` können Namens-Komponenten gemäß dem Datentyp `Cosnaming.Name` mit bis zu vier *Name Components* aus entsprechenden `String`-Komponenten erzeugt werden. Es entsteht der sog *Compound Name* für den Zugriff und das *Name Binding* im CORBA *Naming Service*.

Die Klasse `NameWrapper` ermöglicht die Umwandlung von `CfMAF.Name`-Strukturen (`Authority, Identity, AgentSystemType`) in `String`-Darstellung und zurück sowie die Umwandlung in `CosNaming.Name` Strukturen für den Zugriff auf definierte Kontexte im CORBA *Naming Service*.

MASA-08:
Bei der Implementierung des MAFFinders zusätzlich benötigte Funktionen zur Manipulation von Namen sind in den vorhandenen Klassen `NameSpace` *und* `NameWrapper` *zu implementieren.*

3.2.7 Kompatibilität

Bei der Implementierung der *MAFFinder*-Schnittstelle ist auf die Kompatibilität zum bestehenden MASA-System zu achten.

MASA-09:
Die Kompatibiltät zu früheren MASA-Versionen und die Lauffähigkeit bestehender Agenten ist sicherzustellen.

3.3 Weitere Anforderungen

Nachfolgend werden übergeordnete Anforderungen zusammengestellt:

Der Ausfall des MAFFinder's als Einzelkomponente darf nicht zu einem Ausfall des gesamten MASA-Systems führen.

ALLG-01:
Bei der MAFFinder-Implementierung darf kein „Single Point Of Failure" entstehen, der bei Ausfall der Schnittstelle, die Verfügbarkeit des gesamten Agenten-Systems gefährdet.

4 Entwurf

In diesem Abschnitt werden die getroffenen Design-Entscheidungen und Entwurfsvorschläge skizziert. Zur besseren Übersichtlichkeit werden die Design-Entscheidungen, wie die Anforderungen, fortlaufend nummeriert: DE-<Nummer>.

4.1 Design-Entscheidungen

4.1.1 MAFFinder als CORBA-Objekt, als Agent oder Teil des Agenten-Systems

Gemäß MASIF (s. 3.1.3) ist der *MAFFinder* als „dynamische Datenbasis“ für Namen und Adressen vorgesehen. MASA nutzt derzeit den CORBA *Naming Service* als Datenbank für die benannten CORBA-Objekte. Die Spezifikation lässt offen, ob der *MAFFinder* ein eigenes CORBA-Objekt oder ein Teil des Agenten-Systems wird.

Fragestellung:
Es ist zu entscheiden, ob der *MAFFinder* als separates CORBA-Objekt und als Agent (mobil oder stationär) oder als Teil des Agenten-Systems (Java-Klasse innerhalb des *Packages* `AgentSystem`) realisiert werden soll.

DE-01:
Der CORBA-eigene Naming Service bleibt bestehen. Zugriffe darauf erfolgen über die MAFFinder-Schnittstelle. Der MAFFinder selbst wird Teil des Agent Systems. Der MAFFinder wird kein eigenes CORBA-Objekt, sondern stellt seine Methoden über das CORBA-Objekt Agent System zur Verfügung. Jedes Agent System erhält einen MAFFinder.

Begründung:
Der Zugriff auf den *Naming Service* ist – auch aus Gründen der *IT-Security* - von zentraler Bedeutung zum Lokalisieren von Agenten und Agenten-Systemen.

Der *MAFFinder* als einzelne Komponente ohne Abstützung auf den CORBA *Naming Service* würde das Problem der Einzelfehlerfestigkeit (s. ALLG-01), das Problem der konsistenten Datenhaltung und unabhängig davon die schwierige Einbindung in das MASA-Sicherheitsmodell aufwerfen.

Gerade als eigenes CORBA-Objekt außerhalb des Agenten-Systems müsste die Frage der Autorisierung seiner Methoden unbeantwortet bleiben. Eine Kontrolle und Sicherung über das Agenten-System gemäß [Roel 99] wäre nicht möglich.

Als Teil des Agenten-Systems kann der *MAFFinder* den CORBA *Naming Service* kapseln und seine Methoden über das Agenten-System zur Autorisierung anbieten, unabhängig vom Vorhandensein eines ORB’s entsprechender Funktionalität.

Eine Realisierung als Agent – ob mobil oder stationär – wirft folgende Probleme auf:

- Derzeit wird beim Start des Agenten-Systems der gesamte Namensraum für Agenten und Agenten-Systeme im *Naming Service* initialisiert. Wäre der *MAFFinder* ein Agent, könnte das Agenten-System beim Start solange keinen

Naming Graph aufbauen, bis der *MAFFinder*-Agent erzeugt wurde, der dann erst den Aufbau des *Naming Graph* selbst durchführt. Zur selbständigen Erzeugung von Agenten beim Start des Agenten-Systems werden die Daten der betroffenen Agenten in die sog. *Autodescription*-Datei eingetragen. Das Auslesen der Daten aus der *Autodescription*-Datei und das Erzeugen der Agenten daraus wird vom *Agent Manager* erledigt, der selbst Teil des *Agent System* ist und als *Thread* von diesem gestartet wird. Eine Registrierung des *Agent System* innerhalb seiner `main()`-Methode wäre damit unsicher, da die Erzeugung des *MAFFinders* mit der Initialisierung des *Naming Graph* innerhalb des *Agent Manager Thread's* ablaufen würde und seine zeitliche Existenz damit deterministisch nicht vorhergesagt werden kann.

- Nach [Roel 99] kann das *Agent System* fremde Corba-Schnittstellen und damit die Methoden eines als *MAFFinder* erzeugten Agenten nicht autorisieren, da der verwendete ORB diese Funktionalität bislang nicht bereitstellt.

Darüber hinaus kann durch die Nutzung des standardisierten CORBA *Naming Services* als Basis weiterhin von Entwicklungen auf diesem Gebiet ohne großen Anpassungsaufwand profitiert werden.

4.1.2 Semantik der Location

Im MASIF-Basiskonzept und den Methoden `register()` und `lookup()` der *MAFFinder*-Schnittstelle spielt die *Location* eines Agenten eine wichtige Rolle. Die *Location* spezifiziert die Ausführungsumgebung, den *Place*, eines Agenten (s. MASIF-04). Sie enthält die Adresse (Pfad) seiner Ausführungsumgebung bzw. seines Agenten-Systems (s. MASIF-05) und ist in der Form einer URI oder URL gegeben (s. MASIF-07).

Fragestellung:
Es ist zu entscheiden, ob der Parameter `location` eine URI oder URL enthält. Da die `register()`-Methoden der *MAFFinder*-Schnittstelle keinen Paramter für die IOR eines Objektes vorsehen, ist darüber hinaus zu klären, wie die Objektreferenz, die für das *Name Binding* notwendig ist, an den *MAFFinder* und damit an den *Naming Service* übergeben wird.

DE-02:
Der Parameter `location` *enthält eine URI. Die IOR wird mit einem führenden Slash („/<IOR>“) an die URI angehängt.*

Bemerkung:
IOR :: = „`IOR:`“<`hex_Octets`>

DE-03:
Der Rückgabewert `location` *der MAFFinder-Methoden* `lookup()` *ist ebenfalls eine URI. Die mitgelieferte IOR bezeichnet spezifikationsgemäß das Agent System auf dem das Objekt läuft.*

DE-04:
Um die IOR des Agenten selbst zu erhalten wird eine zusätzliche Methode `lookup_agentIOR()` *implementiert.*

Begründung:
Die URI enthält einen String im CORBA `CosNaming.Name`-Format:

mafuri	: =	scheme“:“location
scheme	: =	“`CosNaming`“
location	: =	components | “/“location
components	: =	component | component“/“components
component	: =	Id“!“kind

Die URI ist also aus den `Id`- und `kind`-Bestandteilen einer *Name Component* aufgebaut und stellt insgesamt einen *Compound Name* dar, der wiederum sofort den Pfad auf die Ausführungsumgebung des gewünschten Objektes angibt. (Anmerkung: In Kombination mit dem *Compound Name* eines Agenten entsteht daraus der Pfad zum Agenten selbst.).

Das Anhängen der IOR erspart zusätzliche Abfrage-Methoden, die im *MAFFinder*-Interface auch nicht vorgesehen sind. Ohne IOR wäre eine Bindung des Objektes durch den *MAFFinder* und damit eine Kapselung des *Naming Services* nicht möglich.

Grundsätzlich wäre die Angabe einer IOR als *Location* ausreichend gewesen, denn sie stellt die vollständige Adresse eines Objektes – hier z.B. die Adresse des Agenten-Systems – dar. MASIF kennt aber das Konzept der *Places* als Ausführungsumgebung von Agenten. Wie im Kapitel 3.1.1 erläutert, bezeichnet der *Place* laut Spezifikation eigentlich nur einen „Ort“ innerhalb eines Agenten-Systems und ist ursächlich natürlich von seinem Agenten-System abhängig. Wie derzeit in MASA (s. Kapitel 3.2.3) ist ein *Place* selbst kein CORBA- Objekt, besitzt also keine IOR und ist damit auch nicht direkt über eine IOR ansprechbar (dies ist gemäß MASIF auch nicht vorgesehen!).

Zur Registrierung eines Agenten gemäß seiner Einbettungsbeziehungen (s. MASIF-02) muss also dem MAFFinder auch der Name des zugehörigen *Places* irgendwie mitgeteilt werden. Wird im Parameter `location` kein *Place*-Name angegeben, muss vom Agenten-System spezifikationsgemäß ein *Default-Place* zugewiesen werden. Damit wäre zukünftig eine Registrierung eines Agenten unter einem bestimmten *Place* – ausser des *Default-Places*, der dem *MAFFinder* implizit bekannt ist - über die *MAFFinder*-Schnittstelle nicht möglich gewesen, da die `register_agent()`-Methode zur Angabe eines *Places* keinen separaten Parameter, sondern nur den Parameter `location` nutzt.

Der Name des Agenten-Systems unter dem der Agent registriert werden soll, wäre dem *MAFFinder* bei alleiniger Angabe der IOR implizit bekannt gewesen, da es das Agenten-System ist, auf dem auch der *MAFFinder* ausgeführt wird. Probleme würden dann wieder auftreten, wenn Agenten-Systeme eines anderen Typs auf dem gleichen Endgerät gestartet werden sollen. Eine Namens-konforme Registrierung über den *MAFFinder* wäre auch dafür ohne die URI-Angabe in der *Location* nicht möglich.

4.1.3 Umsetzung eines neuen Naming Graphen

Abbildung 4.1 zeigt den neuen MASA *Naming Tree*, der sich intuitiv aus der Benennung der MASIF-Objekte *Agent System*, *Place* und *Agent* und der Darstellung der Abhängigkeitsstrukturen ergibt. Wurzel-Kontext für den neuen Naming Tree ist der Kontext *MAFRegions*. Als Start-Kontext ist der *Authority*-Anteil des jeweiligen Namens vorgesehen (s. Bild: AgentSystem.authority bzw. Agent.authority), da er bei Agenten-Systemen als erstes die Gruppierung nach *Regions* erlaubt. Auch bei Agenten ist der erste

Kontext die *Authority*-Komponente, da damit auch eine Gruppierung nach *Security*-Belangen unterstützt wird. Das *Name Binding* auf Blatt-Ebene des Namensbaumes, also die Assoziation der *Identity*-Komponente zu ihrer zugehörigen Objekt-Referenz ist in der Abbildung nur aus Darstellungsgründen mit „= =“ kenntlich gemacht. Ein Beispiel mit einer realen Ausprägung des *Naming Tree* gibt Abbildung 5.9 (s. weiter unten).

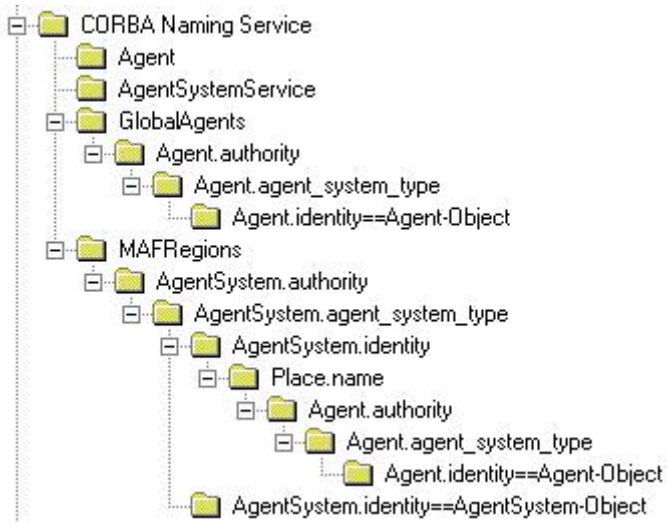


Abbildung 4.1: Der neue MASA Naming Tree

Aus Gründen der Rückwärtskompatibilität bleiben die Kontexte *AgentSystemService* und *Agent* des alten *Naming Graph* zunächst noch erhalten. Sie können später ganz entfallen.

Fragestellung:
Anhand der in [Kemp 98] angegebenen Begründung (s. Kapitel 3.2.1) ist zu entscheiden, ob der neue *Naming Graph* umgesetzt werden kann.

DE-05:
Der neue Naming Graph kann realisiert werden.

Begründung:
Der neue *Naming Graph* durchbricht das in [Kemp 98] angegebene Konzept zunächst nicht. Die feste Zuordnung zwischen *Agent-Identity* und *Agent Class Name* bleibt ohnehin erhalten, bis die Umsetzung eines neuen *Agent-Identity*-Konzeptes das Erzeugen mehrerer Instanzen desselben Agententyps auf einem Agenten-System erlaubt. Auswirkungen auf den *Naming Graph* bzw. Rückwirkungen vom *Naming Graph* sind dabei nicht zu erwarten (s. auch Kap. 4.2: Multiple Agenten-Instanzen). Theoretisch bedeutet der neue *Naming Graph* eigentlich nur das Zusammenhängen der beiden Teilbäume aus [Kemp 98] und das Einfügen der zusätzlichen Kontexte *Place* und der restlichen Namensteile eines Agenten (*Authority*, *AgentSystemType*), so dass ein zusammenhängender Namensbaum gemäß den Einbettungsbeziehungen entsteht (s. MASIF-02).

Die Grundlagen für den neuen *Naming Graph* wurden mit der konsequenten Umstellung auf die *Compound Names* in [Roel 99] geschaffen. Entsprechende Umwandlungsmöglichkeiten der Namen aus der *Compound Name*- in die `String`-Darstellung sind möglich und werden bereits heute genutzt (s. Kap. 3.2.6).

Die Methoden des *MAFFinders* sind auf diesen intuitiven Aufbau des Namensbaumes als gerichteten Graphen abgestimmt. Die damit möglichen und unter Kapitel 3.1.5 aufgezeigten Suchmöglichkeiten erfordern diese Struktur und sind damit einfach zu implementieren.

4.1.4 Behandlung globaler Agenten

Wie im Kapitel 3.2.1 erwähnt, kennt MASA sog. globale oder exklusive Agenten, die nur einmal im gesamten Netzwerk gestartet werden dürfen. Zum schnellen Auffinden globaler Agenten steht in MASA ein eigener Kontext (*global Context*) zur Verfügung. Dieser Kontext soll aus dem gleichen Grund erhalten bleiben. Es ist jedoch sinnvoll, den Kontext als kleinen Namensbaum nach dem *Compound Name* eines Agenten MASIF-konform aufzubauen (s. Abb. 4.1, Kontext „*GlobalAgents*“).

DE-06:
Es wird ein Naming Graph für globale Agenten realisiert. Die Struktur lehnt sich an den Compound Name des Agenten an. Der globale Agent wird darin zusätzlich zum eigentlichen Naming Graph registriert.

Die Methoden der *MAFFinder*-Schnittstelle bieten jedoch keinen Parameter, um zu unterscheiden, ob ein Agent exklusiv ist und damit auch im *Name Graph* für globale Agenten registriert werden muss. (Anm.: Das Attribut `_isGlobal` (s. Klasse `AgentEnvironment`), als Indikator ob der Agent global ist, wird bei der Erzeugung des Agenten festgelegt (s. auch Klasse `AgentManager`).) Um diese Entscheidung durchführen zu können, muss die Methode `register_agent()` einen zusätzlich Parameter zur Verfügung stellen.

Ein ähnliches Problem besteht bei der Überprüfung eines Agenten, ob dieser bereits als globaler Agent existiert. Diese Überprüfung muss im Rahmen der Erzeugung eines Agenten (in den Methoden `create_agent()` und `deserializeAgent()` der Klasse `AgentManager`) durchgeführt werden. Damit entschieden werden kann, dass der Namensbaum für globale Agenten von der *MAFFinder*-Implementierung zu durchsuchen ist, muss auch die Methode `lookup_agent()` einen zusätzlichen Parameter dafür anbieten.

Durch die Erweiterung dieser Methoden um einen zusätzlichen Parameter würde allerdings eine Inkompatibilität zur standardisierten *MAFFinder*-Schnittstelle entstehen. Um die Komformität zum Standard beizubehalten, werden deshalb die beiden Methoden `masa_register_agent()` und `masa_lookup_agent()` bereit gestellt, die sich in ihrer Signatur von den standardisierten Methoden `register_agent()` und `lookup_agent()` nur um den zusätzlichen Parameter `is_global` vom Typ `boolean` unterscheiden.

Wie die Methoden des Interfaces *MAFFinder* werden diese beiden Methoden als CORBA-Methoden realisiert und gemäß DE-01 in `AgentSystem` implementiert. MASA-intern werden für die Regsitrierung und Suche eines Agenten die Methoden `masa_register_agent()` und `masa_lookup_agent()` über `AgentSystem` aufgerufen und an die *MAFFinder*-Implementierung weitergeleitet, die dann über den zusätzlichen Parameter `is_global` die entsprechende Auswahl treffen kann. Externe Systeme und Agenten, die auf MASA zugreifen, verwenden die standardisierten Methoden `register_agent()` und

`lookup_agent()`. `AgentSystem` gibt diesen Aufruf - nach aussen hin transparent - über die MASA-eigenen, korrespondieren Methoden mit dem Parameter `is_global = false` an die *MAFFinder*-Implementierung weiter.

DE-07:
Zur Unterscheidung bei der Registrierung und Suche exklusiver Agenten werden die beiden Methoden `masa_register_agent()` *und* `masa_lookup_agent()` *eingeführt. Sie unterscheiden sich von den korrespondierenden Methoden der MAFFinder-Spezifikation nur um den zusätzlichen Parameter* `is_global`.

4.1.5 Name Binding im neuen Naming Tree

Im bestehenden *Naming Graph* werden die Objekte in `String`-Darstellung ihres *Compound Name* gebunden (s. Kapitel 3.2.2).

Fragestellung:
Es ist zu entscheiden, die CORBA-Objekte lediglich mit dem *Identity*-Anteil ihres *Compound Name* im *Naming Service* zu binden.

DE-08:
Im letzten durch den jeweiligen Compound Name des Objektes festgelegten Kontext werden diese nur mit ihrem Identity-Attribut gebunden.

Begründung:
Im Gegensatz zum alten *Naming Graph* ist das Objekt eindeutig über seinen gesamten *Compound Name* identifizierbar.

Das *Identity*-Attribut des Namens eines Agenten-Systems muss dabei als Name für das CORBA-Objekt selbst und als Kontext-Name zur weiteren Bindung von *Places* und Agenten genutzt werden können. Da innerhalb eines Kontextes die Namen eindeutig sein müssen und nicht gleichzeitig ein Objekt und einen Kontext bezeichnen können, ist eine Unterscheidung notwendig. Diese kann über das `kind`-Attribut der *Identity*-Komponente des Agenten-System-Namens herbeigeführt werden. Diese Unterscheidung kann innerhalb der *MAFFinder*-Implementierung transparent für darauf aufbauende Anwendungen erfolgen

DE-09:
Der Kontext zu einer Agent System Identity erhält als `kind`*-Attribut "*`ctx`*".*

4.1.6 MAFFinder als Thread

Um eine quasi-Nebenläufigkeit bei *MAFFinder*-Anfragen zu erreichen, stellt sich die Frage ob der *MAFFinder* als *Thread* ausgeführt werden soll.

DE-10:
Die MAFFinder-Methoden werden nicht als Thread's realisiert. Der MAFFinder selbst sieht aber eine run-Methode für spätere Ergänzungen vor.

Begründung:
Der *MAFFinder* ist selbst kein CORBA-Objekt, sondern erhält seine Anfragen über das *Agent System*. Damit die Methoden in einem eigenen *Thread* ausgeführt werden, müssten sie über eine gemeinsame `run()`-Methode des *MAFFinders* angesprochen werden oder die Aufrufmethoden des *Agent System's* wären bereits als *Thread* zu realisieren.

Insgesamt ist allerdings kaum ein Performance-Gewinn zu erwarten. Aufrufe von außerhalb des *Agent System* werden ohnehin über den ORB/BOA (*Basic Object Adapter*) als *Thread* ausgeführt. Bei Aufrufen des *Agent System's* selbst muss dieses sowieso auf die Rückkehr des Aufrufes und das Ergebnis warten.

4.1.7 Die Methode unregister_agent()

Die Methode `unregister_agent()` enthält als Parameter nur den Namen des Agenten, der aus dem *Naming Graph* gelöscht werden soll. Da keine *Location* angegeben werden kann, bei der sich der Agent befindet und der *Identity*-Anteil des Agenten-Namens derzeit nicht eindeutig ist, tritt ein ähnliches Problem mit der Lokalisierung des richtigen Agenten auf, wie bereits im Kapitel 4.1.2 geschildert.

Solange kein neues Konzept für die Benennung der *Identity* des Agenten existiert, kann keine zuverlässige Zuordnung zum zu löschenden Objekt erfolgen.

Da die Umsetzung eines neuen *Identity*-Konzeptes den Umfang der Arbeit sprengen würde, wird zur Realisierung der `unregister_agent()`-Methode folgender Kompromiss geschlossen:

Der *MAFFinder* geht beim Aufruf `der unregister_agent()`-Methode davon aus, dass der Agent, dessen *Name Binding* im *Naming Graph* aufzulösen ist, sich auf dem Agenten-System des *MAFFinders* befindet. Den zugehörigen *Place*-Namen kann er transparent für die Anwendung über einen Aufruf der `lookup_agent()`-Methode mit den entsprechenden Suchparametern ermitteln.

Problematisch wird wiederum das Auflösen von *Name Bindings* von Agenten auf Agenten-Systemen fremden Typs, da dort die Annahmen nicht greifen. Jedoch könnte bei diesen Agenten bereits die Möglichkeit einer Suche mit der `lookup_agent()`-Methode greifen, wenn eine entsprechende Namensgebung des Agenten Eindeutigkeit garantiert.

DE-11:
Die Implementierung der `unregister_agent()`*-Methode beruht auf der Annahme, dass sich der Agent, dessen Name Binding zu lösen ist, auf dem Agenten-System des MAFFinders befindet.*

4.1.8 Behandlung der Eigenschaftsprofile AgentSystemInfo und AgentProfile

Die Strukturen `AgentSystemInfo` und `AgentProfile` sind im CfMAF-Modul (s. Anhang B) als Datentypen definiert. Sie enthalten Informationen über das zugehörige Agenten-System bzw. den zugehörigen Agenten. Mit den Methoden `register_agent_system()` bzw.

`register_agent()` werden sie an das Interface *MAFFinder* übergeben und müssten in Zusammenhang mit der Objektreferenz zur Nutzung für spätere Suchläufe gespeichert werden.

DE-12:
Die Strukturen `AgentSystemInfo` *und* `AgentProfile` *werden nicht gespeichert, sondern bei Bedarf über die jeweilige Objektreferenz angefordert.*

Begründung:
Das Speichern dieser Eigenschaftsprofile macht nur in Verbindung mit der Objektreferenz in einer zentralen Datenbasis Sinn, um zusätzliche Abfragen zu sparen. Es müsste deshalb eine Speicherung im CORBA *Naming Service* erfolgen. Dieser sieht die Speicherung solcher Datentypen grundsätzlich nicht vor. Es müsste deshalb eine nicht spezifikationsgerechte Implementierung erfolgen.

Der *MAFFinder* selbst kann die geforderte Funktionalität ebenfalls nicht erbringen. Gemäß DE-01 wird er dezentral in jedem Agenten-System aufgebaut. Eine Speicherung der Eigenschaftsprofile für das eigene Agenten-System und die darauf aktiven Agenten bringt keine Vorteile. Die Speicherung aller Eigenschaftsprofile auf allen *MAFFinder*-Objekten verbietet sich aus Lastgründen ohnehin.

4.2 Multiple Agenten-Instanzen

In [Kemp 98] wurde aus Gründen des einfachen Ansprechens von Agenten und Zuordnung zu *Agent Classes* und *Agent Applets* die Erzeugung nur einer Instanz einer Agenten-Gattung auf einem Agenten-System erlaubt. Dies wurde durch die Beschränkung der *Identity*-Komponente auf den Namen der Agenten-Gattung erreicht.

Für bestimmte Management-Aufgaben wäre es allerdings wünschenswert, mehrere Instanzen einer Agenten-Gattung erzeugen zu können. Auch hat die Implementierung der *MAFFinder*-Schnittstelle und des neuen *Naming Graphen* einige Konfliktsituationen (`unregister_agent()`, eindeutiges Suchen, Erkennung globaler Agenten) offenbart, die ein neues Konzept für ein eindeutiges *Identity*-Attribut von Agenten erfordern.

Folgende Anforderungen an ein solches Konzept lassen sich zusammenfassen:

- Gemäß MASIF-Spezifikation ist die *Identity* so zu wählen, dass sich in Verbindung mit den Komponenten *Authority* und *AgentSystemType* ein global eindeutiger Name für Agenten-Instanzen ergibt.
- Durch diese eindeutige Namensgebung müssen Suchen nur nach dem Agenten-Namen eindeutige Ergebnisse liefern, unabhängig vom Ausführungsort des Agenten (`unregister()`-Problem!).
- Zugehörige Applets müssen sich leicht identifizieren lassen, der Kommunikationsaufbau zur spezifischen Agenten-Instanz muss eindeutig sein.
- Der Aufruf bestimmter Agenten muss leicht möglich sein (z.B. Webserver-Agent auf jedem Agenten-System)
- Es müssen mehrere Agenten-Instanzen desselben Typs auf dem gleichen Agenten-System möglich sein.
- Die Agenten-Gattung muss ableitbar sein.

Denkbar wären dabei folgende Lösungen:

(1) Ergänzung um Ursprungsort der Erzeugung (zur Unterscheidung mobiler Agenten!) und um Datum und Uhrzeit der Erzeugung, z.B.:

Identity :: = <Agenten-Gattung>"_"<hostname>"_"<date>"_"<time>

Hierbei ergibt sich allerdings das Problem nicht synchronisierter Uhren und der notwendigen zeitlichen Auflösung.

(2) Ergänzung um eine laufende Nummer (NameCount), die an die *Identity*-Komponente angefügt wird oder das `kind`-Attribut des `CosNaming.Name`-Formats füllt, z.B.:

`Id` :: = <identity>"_"<NameCount>	`kind` :: = ""
`Id` :: = <identity>	`kind` :: = <NameCount>

Bei dieser Variante müsste die laufende Nummer von einer zentralen Instanz, z.B. dem *Naming Service* vergeben und verwaltet werden. Das Problem des synchronisierten Zugriffes ist zu lösen. Das `kind`-Attribut ist weniger geeignet, da es zur Unterscheidung nur im Naming Service existiert und kein Bestandteil des Namens nach `CfMAF.Name`-Struktur ist.

(3) Ein vollständiger Ansatz könnte so aussehen:

Identity :: =
<Agenten-Gattung>"["<hostname>"_"<date>"_"<time>"_"<sessionID>"]"

Die Agenten-Gattung kann sich an der heutigen Form der *Identity* orientieren. Sie könnte darüber hinaus konstante Hinweise auf den Agenten-Typ wie „*mobile*" oder „*stationary*" (evtl. „*exclusive*", falls konstant bleibend) enthalten. *Hostname*, *Date* und *Time* würden wie bei (1) um eine *SessionID* ergänzt, die bereits heute als Attribut vom Agenten-System zur Verfügung gestellt wird. Der Vorschlag kombiniert die Vorteile aus (1) und (2).

Auch der Zugriff auf HTML-Seiten und zugehörige Applets sowie der Aufbau der Kommunikation zur richtige Agenten-Instanz dürften gemäß Analyse aus Kapitel 3.2.2 bei einer Lösung nach (3) keine Probleme bereiten. So kann sich die Benennung für „statischen" Code wie *Applet Class Name*, *Applet URL* sowie *Code Base* und *Class Name* für Agenten nach dem statischen Namensanteil <Agenten-Gattung> richten, ggf. ist der Quellcode auf entsprechende Ableitungen aus dem Agenten-Namen nochmals detailliert zu überprüfen (s. auch entsprechende *Classloader*-Klassen). Für die Kommunikation selbst wird der vollständige Name herangezogen, der dann auch die „dynamischen" Anteile für die jeweilige Instanz enthalten wird. Entscheidend ist die Einführung eines Teil-Namens für die Instanz bei der Festlegung des Namens im Rahmen der Erzeugung des Agenten auf dem Agenten-System wie unter 3.2.2 beschrieben.

Die Verwirklichung eines entsprechenden Konzeptes würde den Rahmen dieser Arbeit sprengen, der *MAFFinder* soll aber eine einfache Methode zur Abfrage z.B. einer laufenden Nummer gemäß Vorschlag (2) bereitstellen.

DE-13:
Bereitstellen einer Methode `get_NameCount()`*, um einem neu zu instanziierendem Agenten einer Gattung ein Unterscheidungsmerkmal für die Identity geben zu können.*

Anmerkung:
[Scho 99] realisiert ebenfalls eine Plattform für mobile Agenten auf Basis der MASIF-Spezifikation. Für die Benennung von Agenten wird dort eine ähnliche Bezeichnung wie (3) vorgeschlagen, die unabhängig davon entwickelt wurde:

<Agentengattung>"@"<Host>":"<Port>"-"<Datum/Zeit der Erzeugung>"-"<Instanzzähler>

Ein Agenten-System besitzt in [Scho 99] ebenfalls eine ähnliche Bezeichnung wie in MASA:
<Host>":"<Port der Registry>

[Scho 99] wurde auch bzgl. der Umsetzung des *MAFFinders* und des *Place*-Konzeptes geprüft.

Es wird lediglich ein sehr rudimentärer Namensdienst (*Location Server*) realisiert. Er besteht aus einer Liste mit Identifikatoren laufender Agenten-Systeme und ist Teil eines Agenten-Systems. Die Funktionalität kann an andere Agenten-Systeme delegiert werden. Andere Agenten-Systeme besitzen eine Referenz auf das Agenten-System mit *Location Server*. Der *MAFFinder* selbst ist nicht realisiert. Auch ein *Naming Graph* mit entsprechenden Namenskomponenten der Objekte wurde nicht umgesetzt.

Das Konzept der *Places* ist nicht explizit erwähnt. Als Ausführungsumgebung von Agenten werden wie in MASA *Thread's* verwendet, die geeignet über *ThreadGroups* gruppiert werden können.

Resumee:
Die Realisierung eines neuen Konzeptes für global eindeutige Namen von Agenten ist nicht Gegenstand dieser Arbeit. Es wurden lediglich Anforderungen formuliert, die an diese Namensgebung zu stellen sind und Vorschläge zur Lösung unterbreitet. Um die bestehenden Einschränkungen mit der vorhandenen Benennung der Agenten-*Identity* beseitigen zu können und einen global eindeutigen Agenten-Namen nach MASIF zu verwirklichen, sollte die Umsetzung eines neuen Konzeptes vorrangig erfolgen.

5 Implementierung

Zur späteren Auswertung werden wichtige Entscheidungen, die während der Implementierung getroffen wurden mit IE-<Nummer> gekennzeichnet.

Gemäß der Design-Entscheidung DE-01 wird die *MAFFinder*-Schnittstelle als Teil des Agenten-Systems realisiert und damit in der neuen Klasse `MASAFinder` implementiert. `MASAFinder` speichert selbst keine Daten. Als Datenbasis wird weiterhin der CORBA *Naming Service* genutzt. Die Klasse `MASAFinder` kapselt die Zugriffe darauf, ermöglicht erweiterte Suchfunktionen und wird in jedem *Agent System* instanziiert. Zusammen mit dem CORBA *Naming Service* bildet `MASAFinder` den „virtuellen“ *MAFFinder* (s. Abbildung 5.1).

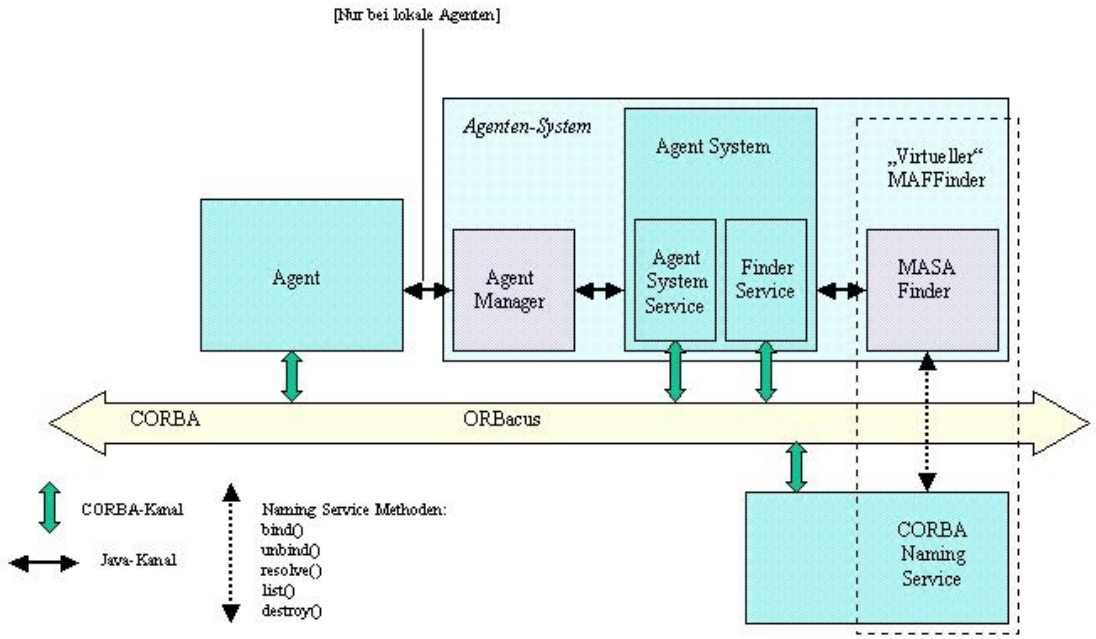


Abbildung 5.1: Die „virtuelle“ MAFFinder-Architektur

5.1 Die IDL-Schnittstelle und ihre Implementierung in der Klasse AgentSystem

Die Definition des Interfaces *MAFFinder* als IDL-Schnittstelle ist in MASA im Modul `CfMAF` bereits enthalten. Gemäß den Design-Entscheidungen DE-04, DE-07 und DE-13 muss das Interface um die beiden Methoden `masa_register_agent()` und `masa_lookup_agent()` sowie die Methoden `lookup_agentIOR()` und `get_nameCount()`

erweitert werden. Ähnlich wie im Interface ***AgentSystemService*** werden diese Methoden im IDL-Interface ***FinderService*** definiert, das die *MAFFinder*-Methoden vom *MAFFinder*-Interface erbt (s. Abbildung 5.2)

```
module agentSystem
{
   interface FinderService : CfMAF::MAFFinder
   {
   /************************************************************************
   * Finder related functions.
   */

   string getNameCount( in CfMAF::Name name)
          raises(CfMAF::EntryNotFound);

   CfMAF::Locations lookup_agentIOR( in CfMAF::Name agent_name,
                                     in CfMAF::AgentProfile agent_profile)
          raises (CfMAF::EntryNotFound);

   void masa_register_agent(  in CfMAF::Name agent_name,
                              in CfMAF::Location agent_location,
                              in CfMAF::AgentProfile agent_profile,
                              in boolean is_global)
          raises (CfMAF::NameInvalid);


   CfMAF::Locations masa_lookup_agent(  in CfMAF::Name agent_name,
                                        in CfMAF::AgentProfile agent_profile,
                                        in boolean is_global)
          raises (CfMAF::EntryNotFound);

   };
};
```

Abbildung 5.2: `FinderService.idl`

Da gemäß DE-01 **`MASAFinder`** nicht als CORBA-Objekt realisiert wird, muss das Interface ***FinderService*** zunächst von der Klasse **`AgentSystem`** implementiert werden (s. Abbildung 5.3: **`FinderServiceOperations`**).

```
final class AgentSystem implements AgentSystemServiceOperations,
                                   org.omg.CosEventComm.PushSupplierOperations,
                                   org.omg.CosEventComm.PushConsumerOperations,
                                   FinderServiceOperations
{}
```

Abbildung 5.3: Klassendefinition `AgentSystem`

Ähnlich wie beim **`AgentManager`** nimmt das **`AgentSystem`** die Anfragen an den **`MASAFinder`** über den CORBA-Kanal entgegen und schleift diese über den Java-Kanal zum **`MASAFinder`**

durch (s. Abbildung 5.4). Erst die Klasse **MASAFinder** implementiert die Methoden tatsächlich. (Anmerkung: Der Begriff „Kanal“ wird hier abstrakt wie in [Roel 99] für Methodenaufrufe als Kommunikationsbeziehungen verwendet).

```
public void register_agent_system( Name agent_system_name,
                                   String agent_system_location,
                                   AgentSystemInfo agent_system_info)
            throws NameInvalid
{
Principal_MASA_Internal callingPrincipal =
            _callTrustDecider.DecideTrust_AS_Multi( _agentSystemService);
Debug.logMessage( Debug.FACILITY_AGENTSYSTEM, Debug.LEVEL_DEBUG1, "Was
called");

getMASAFinder().register_agent_system( callingPrincipal,agent_system_name,
                                                        agent_system_location,
                                                        agent_system_info;
}
```

Abbildung 5.4: Beispiel: Methode `register_agent_system()` in `AgentSystem`

Abbildung 5.5 illustriert die im Kapitel 4.1.4 erläuterte und mit DE-07 eingeführte Vorgehensweise zur Unterscheidung globaler Agenten bei den Methoden **`register_agent()`** und **`lookup_agent()`**.

```
public void register_agent( Name agent_name,
                            String agent_location,
                            AgentProfile agent_profile)
            throws NameInvalid
{
Principal_MASA_Internal callingPrincipal =
            _callTrustDecider.DecideTrust_AS_Multi( _agentSystemService);
Debug.logMessage( Debug.FACILITY_AGENTSYSTEM, Debug.LEVEL_DEBUG1,
                "Was called");

// for clients which use the interface MAFFinder, is_global is always false
boolean is_global = false;

getMASAFinder().masa_register_agent( callingPrincipal, agent_name,
                                     agent_location, agent_profile,
is_global);
}
```

Abbildung 5.5: Methode `register_agent()` in `AgentSystem` mit Aufruf von `masa_register_agent()`

Die Klasse **`MASAFinder`** realisiert anstelle der Methoden **`register_agent()`** und **`lookup_agent()`** nur die Methoden **`masa_register_agent()`** und **`masa_lookup_agent()`** mit dem zusätzlichen Parameter **`is_global`**. Für MASA-interne Aufrufe werden sie als

CORBA-Methoden von **AgentSystem** angeboten und wie in Abbildung 5.4 gezeigt an **MASAFinder** durchgereicht. Externe (nicht in MASA implementierte) Clients können die standardisierten CORBA-Methoden nutzen. Diese Aufrufe gibt **AgentSystem** mit der Parameter-Zuweisung **is_global = false** an die entsprechende **MASAFinder**-Methode weiter (s. Abbildung 5.5).

Gemäß DE-10 wird der **MASAFinder** nicht als *Thread* realisiert. Für spätere Erweiterungen implementiert die Klasse aber trotzdem das *Runnable*-Interface mit einer entsprechenden **run()**-Methode ohne „Body", d.h. ohne weitere Anweisungen.

Im Konstruktor von **AgentSystem** wird neben der Instanziierung eines **MASAFinder**-Objektes ein eigener *Thread* mit *ThreadGroup* angelegt (s. Abbildung 5.6). Der *Thread* wird in der **main()**-Methode von **AgentSystem** gestartet (Methode **start_masafinderThread()**) und sofort nach Abarbeitung der leeren **run()**-Methode von **MASAFinder** wieder gestoppt.

```
// create MASAFinder
MASAFinder masaFinder = new MASAFinder( _certManager, _callTrustDecider,
                                        orb, boa, initContext,
                                        agentSystemInfo.agent_system_name);
_masaFinder = masaFinder;

// create ThreadGroup and Thread for MASAFinder
_masaFinderThreadGroup = new ThreadGroup( _rootThreadGroup,
"MASAFinderGroup");
_masaFinderThread = new Thread( _masaFinderThreadGroup, _masaFinder,
                                "MASAFinderThread");
_masaFinderThread.setPriority( (Thread.MAX_PRIORITY*2)/3);
```

Abbildung 5.6: Auszug aus dem Konstruktor von AgentSystem

5.2 Die Klasse MASAFinder

Abbildung 5.7 zeigt das Klassenmodell der **MASAFinder**-Implementierung. Die Klasse **MASAFinder** ist im *Package* **agentSystem** realsiert und besitzt außer dem Interface **Runnable** keine weiteren Schnittstellen. Der gestrichelte Pfeil zwischen **AgentSystem** und **MASAFinder** kennzeichnet die CORBA-Methodenaufrufe, die von **AgentSystem** an **MASAFinder** weitergeleitet werden, d.h. zur Abarbeitung der Methoden ist **AgentSystem** von **MASAFinder** abhängig.

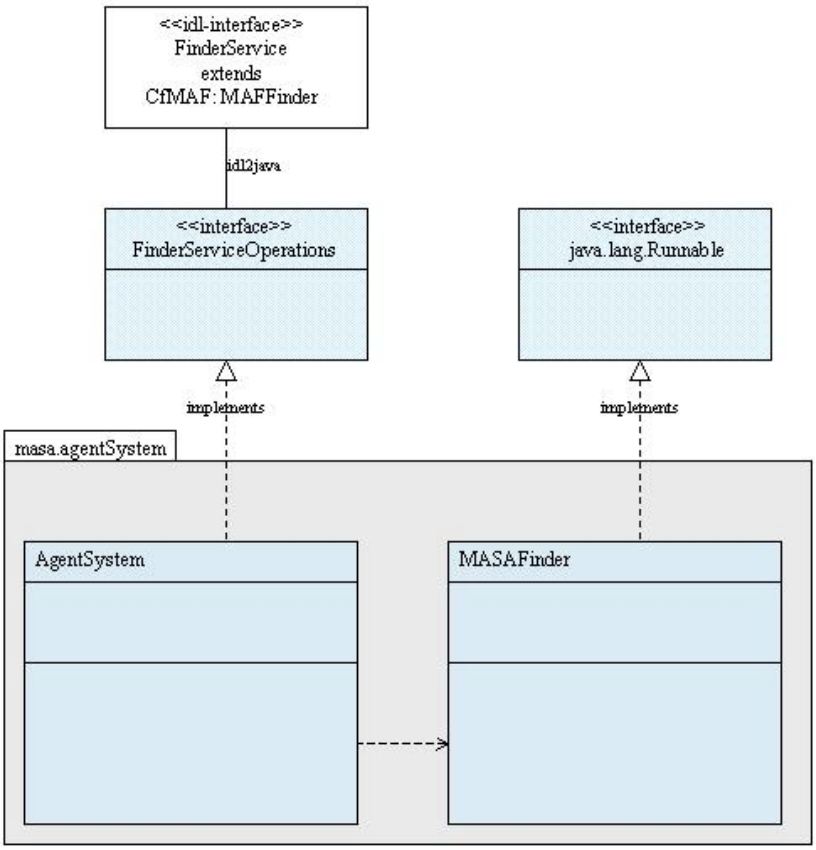


Abbildung 5.7: Klassenmodell `MASAFinder` in UML-Notation

5.2.1 Die Klassendefinition

Der Java-Quellcode der Klasse kann im Anhang C eingesehen werden.

Zur Umsetzung der Sicherheitseigenschaften gemäß [Roel 99] (s. MASA-07) wird die Klasse selbst `final` deklariert. Eine unerwünschte Vererbung an Unterklassen wird damit ausgeschlossen. Attribute und Methode, die nur Klassen-intern Verwendung finden, werden als `private` definiert, ein Zugriff von außerhalb der Klasse wird damit sicher verhindert. Die Implementierung der Methoden aus den Interfaces ***`MAFFinder`*** und ***`FinderService`*** geschieht im Modus `protected`, da sie nur von der Klasse `AgentSystem` aus, also innerhalb des *Packages* aufgerufen werden.

IE-01:
Umsetzung der Sicherheitseigenschaften gemäß MASA-07.

Das `MASAFinder`-Objekt wird durch das `AgentSystem` instanziiert. Dabei werden beim Konstruktor-Aufruf die Parameter `inSystemCertManager`, `inCallTrustDecider`, `orb`, `boa`, `initContext` und `agent_system_name` übergeben.

Die Parameter `inSystemCertManager`, `inCallTrustDecider` dienen dem Aufruf des sog. `PermissionManager` (s. [Roel 99]), der eine Autorisierung von Schnittstellen-Methoden

vornehmen kann. Da alle Schnittstellen-Methoden bereits über **AgentSystem** autorisiert werden, wurden diese Parameter nur für spätere Erweiterungen bereit gestellt.

Um selber über den CORBA-Kanal kommunizieren zu können werden mit den Parametern **orb** und **boa** entsprechende Referenzen übergeben, die bereits für das **AgentSystem** initialisiert wurden.

Der Parameter **initContext** liefert eine Referenz auf den CORBA *Naming Service*. Der Name des Agenten-Systems, das den **MASAFinder** instanziiert wird zum Aufbau des *Naming Tree* benötigt.

Bereits bei der Instanziierung wird der für das *Agent System* gültige Namensbaum erzeugt. Durch die Methode **initNamingTree()** werden die Wurzel-Kontexte *MAFRegions* und *GlobalAgents* angelegt. Innerhalb dieser Methode werden über **initAgentSystemTree()** die für das eigene Agenten-System gültigen Kontexte seiner *Authority* und seines Typs aus seinem Namen erzeugt. Im *Naming Service* registriert wird das Agenten-System allerdings erst durch den Aufruf der Methode **register_agent_system()** innerhalb der **main()**-Methode in der Klasse **AgentSystem** selbst.

Existieren die entsprechenden Kontexte bereits, wird deren Referenz vom CORBA *Naming Service* erfragt. Abbildung 5.8 zeigt den Konstruktor der Klasse **MASAFinder**.

```
protected MASAFinder(  AgentSystemCertManager inSystemCertManager,
                       AgentSystemPermissionManager inCallTrustDecider,
                       org.omg.CORBA.ORB orb, org.omg.CORBA.BOA boa,
                       org.omg.CosNaming.NamingContext initContext,
                       org.omg.CfMAF.Name agent_system_name)
{
super();

_systemCertManager = inSystemCertManager;
_callTrustDecider = inCallTrustDecider;
_orb = orb;
_boa = boa;
_initContext = initContext;
_agent_system_name = agent_system_name;

_masaFinder = this;

// create context "MAFRegions", context for the Agent System on which
// MASAFinder is running and context for global Agents.
this.initNamingTree( agent_system_name);
}
```

Abbildung 5.8: Konstruktor MASAFinder

5.2.2 Der neue Naming Tree

Der MASA *Naming Tree* wie er mit DE-05 beschlossen wurde ist in Abbildung 5.9 mit eingetragenen Beispielen zu sehen. Die Wurzel des Baumes wird vom Kontext *MAFRegions* gebildet. Darunter werden *AgentSystem*, *Place* und *Agent* jeweils mit ihrem

Compound Name in der Reihenfolge ihrer Namenskomponenten *Authority*, *Type* und *Identity* eingetragen.

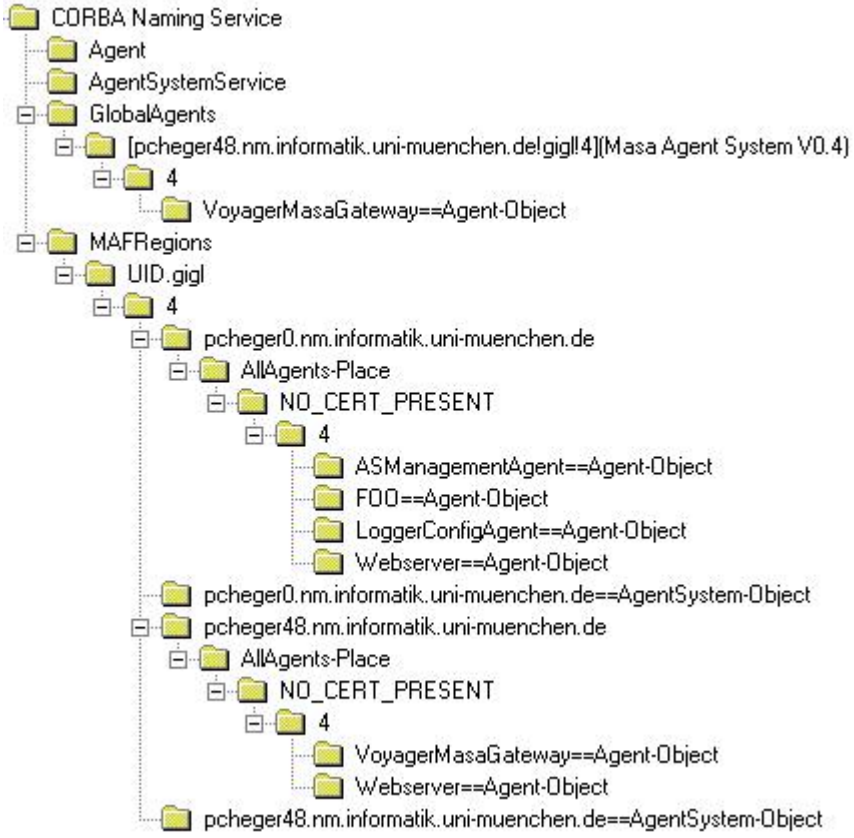


Abbildung 5.9: MASA Naming Tree mit Beispielen

Im Beispiel sind die beiden Agenten-Systeme `pcheger0.nm.informatik.uni-muenchen.de` und `pcheger48.nm.informatik.uni-muenchen.de` (jeweils die *Identity*) unter ihrer gemeinsamen *Authority* `UID:gigl` und dem *AgentSystemType* `4` registriert. Daran schließt sich jeweils der zur Zeit einzige in MASA realisierte *Place*, der *Default-Place* `AllAgents-Place` an.

Die Belegung der *Authority*-Komponente entspricht der MASA-Version ohne Sicherheitseigenschaften. Sie enthält für Agenten derzeit `NO_CERT_PRESENT` (s. Kapitel 3.2.2). Hinter dem *AgentSystemType* `4` der Agenten sind schließlich die Agenten-Referenzen unter ihrer *Identity* registriert, z.B. `Webserver`.

Die Idee des Kontextes *GlobalAgents* aus [Kemp 98] wurde beibehalten, um die Überprüfung auf das Vorhandensein eines exklusiven Agenten zu beschleunigen. Grundsätzlich wäre eine Identifizierung durch die `kind`-Komponente der *Identity* des Agenten-Namens möglich, würde aber ein Durchsuchen des gesamten Baumes erforderlich machen (Performance-Verlust!).

Im Beispiel oben ist der globale Agent `VoyagerMasaGateway` z.B. herkömmlich unter seinem Agenten-System `pcheger48` und zusätzlich im Kontext *GlobalAgents* registriert. Beispielhaft für die sog. *Secure Version* von MASA wurde im Kontext *GlobalAgents* als

Authority die *Authority* eines vom Agenten-System erzeugten Agenten, also das Zertifikat des Agenten-Systems selbst eingetragen (s. Kapitel 3.2.2), hier:
`[pcheger48.nm.informatik.uni-muenchen.de!gigl!4](Masa Agent System V0.4)`

Der Baum wird durch die entsprechenden `register()`-Methoden erweitert, durch `unregister()`-Methoden evtl. reduziert. Diese Methoden greifen über die entsprechenden Funktionen wie `bind()`, `unbind()`, `resolve()` und `destroy()`, die in [OMG 98-10-11] spezifiziert sind, auf den CORBA *Naming Service* zu.

Mit `register_agent_system()` trägt sich das Agenten-System im *Naming Service* ein. Das *Name Binding* wird mit der *Identity*-Komponente seines Namens im Kontext *AgentSystemType* versucht. Die für das *Name Binding* notwendige CORBA-Objekt-Referenz wird als IOR mit dem Parameter `location` übergeben. Existieren die bei der Initialisierung des `MASAFinders` angelegten Kontexte für *Authority* und *AgentSystemType* nicht, werden sie jetzt erzeugt. Damit können z.B. auch fremde Agenten-Systeme über diese Methode registriert werden.

Gleichzeitig mit dem *Name Binding* für das Objekt wird ein Kontext mit der Bezeichnung der *Identity* des Agenten-System-Namens angelegt, um später *Places* und darauf die Agenten registrieren zu können. Wie unter 4.1.4 erläutert und mit DE-09 entschieden, wird der *Identity*-Kontext mit dem Zusatz „`ctx`“ für Context in seiner `kind`-Komponente im *Naming Service* unter dem *Authority*-Kontext angelegt, da die *Identity* als Bezeichnung bereits für das Binding des Objektes verwendet wurde und die Bezeichnungen innerhalb eines Kontextes nach [OMG 98-10-11] unterscheidbar sein müssen. Der Zusatz „`ctx`“ wird jeweils vom `MASAFinder` transparent für darauf zugreifende Anwendungen erzeugt, d.h. die Entscheidung ob das Objekt oder der korrespondierende Kontext gemeint sind, trifft der `MASAFinder` intern.

Die Methode `register_place()` erzeugt einen Kontext mit dem Namen eines *Places*, auf dem dann die Agenten zu registrieren sind. Der Parameter `location` dieser Methode enthält als `mafuri` den Namen des Agenten-Systems für den der *Place* registriert werden soll. Der *Place*-Kontext wird unter den *Identity*-Kontext des jeweiligen Agenten-Systems eingehängt. Existiert dieser schon, wird die Referenz vom *Naming Service* erfragt.

Über die Methode `masa_register_agent()` können schließlich die Agenten registriert werden. Da damit globale Agenten auch im Kontext *GlobalAgents* eingetragen werden müssen, ist zu Beginn der Methode zu entscheiden, ob es sich um einen globalen Agenten handelt oder nicht. Diese Entscheidung kann über den gegenüber der Methode `register_agent()` zusätzlich eingeführten Parameter `is_global` getroffen werden.

Wie das Agenten-System wird auch der Agent mit seinem *Compound Name* in den *Naming Tree* eingetragen. Existieren die Kontexte entsprechend seiner *Authority* und seines *AgentSystemTypes* unterhalb des *Place*-Kontextes und im *GlobalAgents*-Kontext (falls globaler Agent) nicht, müssen diese erst über die Methode `initAgentTree()` angelegt werden. Das *Name Binding* erfolgt wie beim Agenten-System mit der *Identity*-Komponente seines Namens im *AgentSystemType*-Kontext. Die für das *Name Binding* notwendige CORBA-Objekt-Referenz wird als IOR im Parameter `location` übergeben, der beim Agenten darüber hinaus die `mafuri` seines Pfades (Name von Agenten-System und *Place*) enthält.

Bei der `unregister_agent()`-Methode wird wie unter Kapitel 4.1.6 erläutert, davon ausgegangen, dass sich der Agent auf dem Agenten-System des `MASAFinders` befindet. Der entsprechende *Place* auf dem der Agent läuft wird dann über die interne Methode `list_and_compare()` ermittelt (s. später). Daraufhin kann das *Name Binding* gelöscht werden. Existieren keine weiteren *Name Bindings*, werden auch die Kontexte für *AgentSystemType* und *Authority* des Agenten entfernt. Sofern der Agent als globaler Agent auch im Kontext *GlobalAgents* eingetragen ist, wird das *Name Binding* auch dort aufgelöst und die entsprechenden Sub-Kontexte soweit möglich gelöscht.

Auch bei der Methode `unregister_place()` wird implizit davon ausgegangen, dass es der *Place* auf dem Agenten-System des `MAFFinders` ist. Dies ist auch zulässig, da jedes Agenten-System einen eigenen `MASAFinder` besitzt und nur seine *Places* löschen wird.

Anmerkung:
Spezifikationsgemäß ist der Name eines *Places* vom Datentyp `String`, hinsichtlich seiner Semantik aber nicht näher bestimmt. Ein *Place* ist unmittelbar seinem Agenten-System zugehörig. Eine *Authority* könnte derzeit nur über das Agenten-System abgeleitet werden. Die Suche nach *Places* einer bestimmten *Authority* lässt die Signatur der Methode `lookup_place()` auch nicht zu. Bei der Erarbeitung eines Konzeptes für *Places* sollte darüber hinaus eine sinnfällige Namensgebung überlegt werden.

Bei `unregister_agent_system()` wird neben der Auflösung des *Name Bindings* für das CORBA-Objekt selbst auch der entsprechende *Identity*-Kontext gelöscht. Sind keine weiteren Agenten-Systeme mehr registriert, werden auch die darüber befindlichen Kontexte für *AgentSystemType* und *Authority* gelöscht.

5.2.3 Die Attribute

Die Klasse `MASAFinder` definiert die folgenden Attribute, die alle gemäß IE-01 `private` deklariert sind:

Nachfolgende Attribute werden wie oben beschrieben bei der Instanziierung durch `AgentSystem` als Parameter übergeben und von `MASAFinder` zusätzlich als `final` deklariert, da sie sich nicht mehr ändern:

- `_systemCertManager`
- `_callTrustDecider`
- `_orb`
- `_boa`
- `_initContext`
- `_agent_system_name`

Variable für die Referenz auf ein Objekt der Klasse selbst:

- `_masaFinder`

Variablen für Referenzen auf die Kontext-Objekte des *Naming Graphen*; der zugehörige Kontext geht aus der Namensgebung hervor:

- `_regionsCtx`
- `_systemAuthCtx`
- `_systemTypeCtx`

- _systemIdCtx
- _placesCtx
- _agentAuthCtx
- _agentTypeCtx
- _agentIdCtx
- _agentGlobalCtx

Die folgenden Konstanten definieren eine bestimmte (aus ihrem Namen hervorgehende) Ebene des Namensbaumes ausgehend vom Kontext *MAFRegions*:

- LEVEL_START = 0;
- LEVEL_AS_AUTH = 1;
- LEVEL_AS_TYPE = 2;
- LEVEL_AS_ID = 3;
- LEVEL_PLACE = 4;
- LEVEL_A_AUTH = 5;
- LEVEL_A_TYPE = 6;
- LEVEL_A_ID = 7;

Für den Kontext *GlobalAgents* gelten die Konstanten:

- LEVEL_GA_AUTH = 1;
- LEVEL_GA_TYPE = 2;
- LEVEL_GA_ID = 3;

Diese Konstanten werden in der Methode **list_and_compare()** (s. unten) zur Belegung der Parameter verwendet, die vorwiegend die relevanten Ebenen für die durchzuführenden Vergleiche festlegen. (Anm.: AS: *Agent System*, A: *Agent*, GA: *Global Agent*)

Die Konstanten im untenstehenden Block dienen in den Methoden **list_and_compare()** bzw. **composeMAFURI()** (s. unten) der Steuerung der Komposition der relevanten MAFURI inkl. der interessierenden IOR:

- AGENTSYSTEM = 1;
  Such nach *Agent System*, Rückgabe MAFURI und IOR des *Agent Systems*
- PLACE = 2;
  Suche nach *Place*, Rückgabe MAFURI und IOR des zugehörigen *Agent Systems*
- AGENT = 3;
  Suche nach *Agent*, Rückgabe MAFURI und IOR des *Agents*
- AGENT_IORAS = 4;
  Suche nach *Agent*, Rückgabe MAFURI (Place) und IOR des zugehörigen *Agent Systems*
- AGENTGLOBAL = 5;
  Suche nach *Agent* in Kontext *GlobalAgents*, Rückgabe MAFURI und IOR des *Agents*
- DEFAULT = -1;
  Rückgabe lediglich der ersten drei Namenskomponenten eines Pfades als MAFURI, keine IOR

5.2.4 Die Methoden

Die Klasse **MASAFinder** implementiert die im Interface ***MAFFinder*** definierten CORBA-Methoden:

- register_agent_system()

- register_place()
- lookup_agent_system()
- lookup_place()
- unregister_agent()
- unregister_agent_system()
- unregister_place()

Anstelle der in ***MAFFinder*** definierten Methoden

- register_agent()
- lookup_agent()

werden wie oben beschrieben die Methoden

- masa_register_agent()
- masa_lookup_agent()

implementiert, die im IDL-Interface ***FinderService*** definiert sind.

Sie implementiert die ebenfalls im Interface ***FinderService*** zusätzlich definierten CORBA-Methoden:

- lookup_agentIOR()
- getNameCount()

Die **private** deklarierten Methoden werden nur innerhalb der Klasse verwendet:

- initNamingTree()
- initAgentSystemTree()
- initAgentTree()
- list_and_compare()
- compareElement()
- compareFilter()
- composeMAFURI()
- listNameContext()

Zur Implementierung des Interfaces ***Runnable*** wird die Methode

- run()

implementiert, die derzeit keine Anweisungen enthält und für spätere Erweiterungen vorgehalten wird.

Die Funktionsweise der Methoden **register()** und **unregister()** wurde bereits im vorhergehenden Kapitel vorgestellt und eingehend erläutert. Von Interesse sind vor allem die **lookup()**-Methoden, die die Komfortabilität des **MASAFinders** zur Suche von CORBA-Objekten gegenüber der vorhergehenden Nutzung des *Naming Services* in MASA etablieren.

Die **lookup()**-Methoden wurden spezifikationsgemäß (s. MASIF-12) implementiert, so dass jetzt die im Kapitel 3.1.5 ausführlich erklärten Suchmöglichkeiten angewandt werden können. **Lookup()** stützt sich auf die interne Methode **list_and_compare()** ab, die sehr allgemein gehalten wurde und den Baum systematisch durchsuchen kann. Sie soll im folgenden detaillierter betrachtet werden.

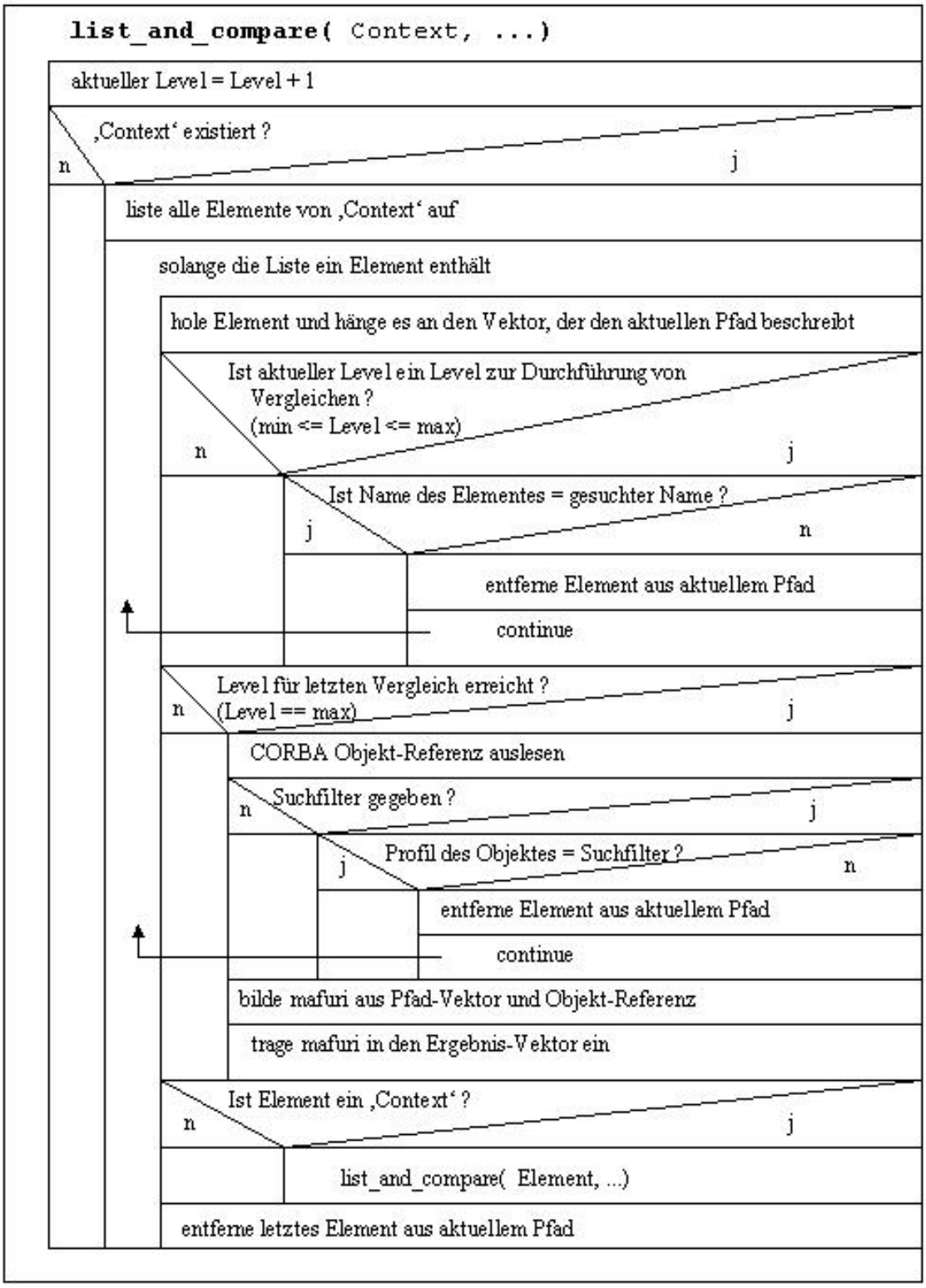


Abbildung 5.10: Nassi-Shneiderman-Diagramm von list_and_compare()

Das Nassi-Shneiderman-Diagramm in Abbildung 5.10 gibt den prinzipiellen Ablauf der Methode wieder. Die Signatur der Methode ist mit Abbildung 5.11 gegeben.

```
private void list_and_compare(NamingContext ctx,
                              int level, int minLevel, int maxLevel,
                              int iorLevel,
                              NameComponent[] name,
                              java.lang.Object filter, boolean is_system,
                              java.util.Vector result,
                              java.util.Vector path)
{ }
```

Abbildung 5.11: Signatur der Methode list_and_compare()

Der Parameter **ctx** enthält den Kontext in dem mit der Suche begonnen wird, **level** die zughörige Ebene des Baumes (als Startparameter immer: **level = 0** bzw. **level = LEVEL_START**). Die Attribute **minLevel** und **maxLevel** schränken die Ebenen ein auf denen die Namen der Kontexte mit den *Compound Names* der gesuchten Objekte verglichen werden müssen (Belegung mit obigen Konstanten). Da z.B. bei der Methode **masa_lookup_agent()** zwar nach einem Agenten gesucht wird, aber die IOR des zughörigen Agenten-Systems zurückgegeben werden muss, wird mit **iorLevel** die Rückgabe der entsprechenden IOR gesteuert (Belegung mit obigen Konstanten; im Beispiel **iorLevel = AGENT_IORAS**). Der Parameter **name** enthält die Namenskomponenten des zu suchenden Objektes, in der Regel die Komponenten *Authority*, *AgentSystemType* und *Identity* (jeweils **Id**- und **kind**-Anteil). Wird nur nach einer bestimmten Komponente gesucht, muss für die unbestimmten Komponenten **null** oder ein leerer **String** angegeben werden. Der Parameter **filter** enthält entweder das Eigenschaftsprofil **AgentSystemInfo** für die Suche nach einem Agenten-System oder **AgentProfile** für die Suche nach einem Agenten. Das Attribut **is_system** ermöglicht dabei die Typ-Bestimmung für **filter** und muss **true** gesetzt werden, wenn **filter** das Profil **AgentSystemInfo** enthält. Zur Ergebnisbestimmung dienen die beiden Vektoren **result** und **path**. **Path** enthält den Pfad von der Wurzel zum gefundenen Objekt, aus ihm wird die zurückzugebende *MAFURI* gebildet, die zusammen mit der IOR in **result** zwischengespeichert wird. **Result** wird in den **lookup()**-Methoden auf deren Rückgabewert **locations** abgebildet.

Zum Traversieren des Baumes wird in **list_and_compare()** das Prinzip der Tiefensuche (*Depth-First-Search*-Algorithmus, s. z.B. [Heun 97]) angewandt. Ausgehend von der Wurzel des Baumes bzw. eines Knotens auf einer bestimmten Baumebene (Parameter **ctx** und **level**) werden zunächst die darunter liegenden Knoten (hier: Kontexte und Objekte) aufgelistet. Das jeweils nächste Element der Liste wird daraufhin überprüft, ob es sich um ein Objekt oder einen Kontext handelt. Ist das Element ein Kontext, wird dasselbe Verfahren auf dieses Element angewandt. Mit den dabei entstehenden Rekursionen wird der Baum bis zur entsprechenden Suchtiefe durchschritten.

Mit den Parametern **minLevel** und **maxLevel** wird der Suchfunktion mitgeteilt, auf welchen Ebenen Vergleiche durchzuführen sind. Der Namens-Vergleich zwischen aktuellem Element und der Namenskomponente des gesuchten Objektes wird an die Methode **compareElement()** delegiert. Namens-Komponenten, die nicht zum Vergleich herangezogen werden sollen, sind bereits bei Aufruf der entsprechenden **lookup()**-Methode vom Anwender mit einem leeren **String** zu versehen. Der implementierte Vergleich liefert dann das Ergebnis **true**. So kann z.B. nach Objekten einer bestimmten *Authority*, eines bestimmten Typs oder einer bestimmten Gattung (*Identity*-Komponente)

gesucht werden. Beispielsweise könnten damit die Agenten-Systeme einer bestimmten *Region* erfragt werden.

Bei Erreichen des **maxLevels**, in dem sich in der Regel die Referenzen auf die CORBA-Objekte befinden, kann ein Vergleich der Objekt-Eigenschaften **AgentSystemInfo** bzw. **AgentProfile** des gefundenen Objektes vorgenommen werden. Der entsprechende Suchfilter dazu verbirgt sich im Parameter **filter**. Er enthält **null** wenn keine Vergleiche erforderlich sind. Der Vergleich selbst wird wieder an die Methode **compareFilter()** delegiert. Vor einem Vergleich der Parameter **AgentSystemInfo** und **AgentProfile** mit dem Suchfilter müssen diese erst über die Objekt-Referenz und die entsprechende **get()**-Methode vom in Frage kommenden Agenten/Agentensystem abgeholt werden, da diese nicht bei der Objekt-Registrierung im **MASAFinder** bzw. im CORBA *Naming Service* gemäß DE-12 gespeichert werden. Auch hier müssen für einen Vergleich irrelevante Komponenten entweder mit **0** oder leerem **String** vorbelegt sein.

Die Parameter der jeweiligen Eigenschaftsprofile werden der Reihenfolge nach von **compareFilter()** mit den Parametern des Suchprofils verglichen. Der Abbruch erfolgt, sobald ein Vergleich das erste negative Ergebnis liefert.

Findet **list_and_compare()** Objekte, auf die Such-Namen oder Such-Filter zutreffen, wird über die Methode **composeMAFURI()** die jeweilige **mafuri** des gefundenen Objektes, also der Pfad (zu erhalten aus dem Vektor **path**) und die IOR (über den CORBA *Naming Service*), zusammengesetzt und dem Ergebnisvektor **result** übergeben. Die Suche ist beendet, wenn der gesamte Baum durchschritten ist.

```
AgentSystemInfo()
{
Name                agent_system_name; // Beispiele s. oben
short               agent_system_type; // MASA
LanguageMap[]       language_maps;
                // language_maps[0].language_id: Java
                // language_maps[0].serializations[0]: Java object
serialization
String              agent_system_description;
                // MobileAgentSystemArchitecture (MASA); (c) MNM-Team, Munich
short               major_version; // 0
short               minor_version; // 4
org.omg.CORBA.Any[] properties; // z. Zt. keine
}

AgentProfile()
{
short               language_id; // Java
short               agent_system_type; // MASA
String              agent_system_description;
                // MobileAgentSystemArchitecture (MASA); (c) MNM-Team, Munich
short               major_version; // 0
short               minor_version; // 4
short               serialization; // Java object serialization
org.omg.CORBA.Any[] properties; // z. Zt. keine
}
```

Abbildung 5.12:: AgentSystemInfo und AgentProfile

Die Suchmöglichkeiten hängen natürlich von einer entsprechenden Belegung der Eigenschaftsprofile für Agenten-Systeme und Agenten ab, die bislang in MASA nicht vorrangig angelegt wurden. Bild 5.12 zeigt die beiden Strukturen `AgentSystemInfo` und `AgentProfile` mit ihren Attributen und Beispiele ihrer derzeitigen Belegung (jeweils dargestellt als Kommentarzeile). Die Attribute selbst wurden bereits im Kapitel 3.1.5 näher beschrieben. Vor allem das *Property*-Objekt besitzt zur Zeit keine Ausprägung in MASA. MASIF enthält einen Hinweis, dass entsprechende *Properties* Gegenstand zukünftiger Standardisierung sein können. Eine sinnvolle Belegung, um die Suchmöglichkeiten weiter zu differenzieren ist nicht Gegenstand der Aufgabenstellung und kann künftigen Arbeiten im Rahmen der Weiterentwicklung von MASA vorbehalten werden.

Der Vollständigkeit halber sei noch die Funktionalität der Methoden `listNameContext()`, `lookup_agentIOR()` und `getNameCount()` erläutert.

Die Methode `listNameContext()` wurde als Hilfsfunktion zum *Debugging* angelegt. Sie nimmt als Parameter eine Referenz auf ein Kontext-Objekt, listet die Elemente des Kontextes einzeln auf und gibt sie über den MASA-*Logging*-Mechanismus aus.

`GetNameCount()` wurde gemäß DE-13 implementiert, um evtl. später eine Identifizierungsnummer für die global eindeutige Benennung einer Agenteninstanz abfragen zu können. Die Methode wurde als sog. *Skeleton* bereitgestellt und enthält derzeit keine Funktionalität.

Da (`masa_`)`lookup_agent()` gemäß MASIF die IOR des zugrundeliegenden Agenten-Systems liefert, wird mit DE-04 die Methode `lookup_agentIOR()` implementiert. Sie funktioniert genau so wie `lookup_agent()`, gibt aber bei Bedarf die IOR des Agenten selbst zurück. Damit kann eine weitere Abfrage der IOR des Agenten erspart werden.

5.3 Änderungen in weiteren MASA-Klassen

5.3.1 De-/Registrieren beim Naming Service

Aus Kompatibilitätsgründen wird der neue *Name Graph* mit dem `MASAFinder` im ersten Schritt so implementiert, dass auch der bisherige *Naming Service* und *Name Graph* noch funktionsfähig bleibt. Ein Nebeneinander beider Namensbäume im CORBA *Naming Service* ist unproblematisch. Eine Trennung ist über die entsprechende Benennung der Wurzelkontexte sichergestellt.

Die Methoden zum De-/Registrieren beim *Naming Service* müssen in `AgentSystem` für das Agenten-System selbst und die darauf existierenden *Places* sowie in `AgentEnvironment` für die Agenten angepasst bzw. bereitgestellt werden.

5.3.2 Places in MASA

Gemäß MASA-04 sollen der *Place* im neuen *Naming Tree* berücksichtigt werden. Da das Konzept der *Places* (*Classloader* oder *ThreadGroup*) in MASA noch nicht gefestigt ist, wird zum Aufbau des Namensbaumes die *ThreadGroup „AllAgents-Place*“ als d*efault-Place* angenommen.

IE-02:
Als Place fungiert die ThreadGroup "All-Agents-Place".

5.3.3 Änderungen in der Klasse AgentSystem

Wie bereits im Kapitel 5.1 beschrieben implementiert die Klasse alle Methoden der Interfaces ***MAFFinder*** und ***FinderService*** und instanziiert im eigenen Konstruktor ein Objekt der Klasse **MASAFinder** nebst zugehöriger **ThreadGroup**.

Zum Start des **MASAFinder** *Threads* wird eine eigene Methode **startMASAFinder()** implementiert. Der Start wird in der **main()**-Methode angestossen.

Die Methode **getMASAFinder()** liefert die Java-Referenz auf ein **MASAFinder**-Objekt.

Die Klasse **AgentSystem** erzeugt, wie Abbildung 5.13 zeigt, in ihrer **main()**-Methode einen eigenen BOA-Anschluss (Basic Object Adapter) für die Methoden des Interfaces ***FinderService***, über den auch die Methoden des Interfaces ***MAFFinder*** erreicht werden können. Damit werden diese Methoden vom BOA-Anschluss für das Interface ***AgentSystemService*** separiert und zusätzliche Übersichtlichkeit und auch Sicherheit gewonnen.

```
_AgentSystemServiceImplBase_tie agentSystemService = null;
_FinderServiceImplBase_tie finderService           = null;

agentSystemService= new _AgentSystemServiceImplBase_tie(agentSystem);
boa.obj_is_ready(agentSystemService, null);

finderService= new _FinderServiceImplBase_tie(agentSystem);
boa.obj_is_ready(finderService, null);
```

Abbildung 5.13: BOA-Anschluss für das Interface FinderService

Über die Aufrufe **register_agent_system()** und **register_place()** in der **main()**-Methode registriert sich das Agenten-System und seinen zur Zeit einzigen *Place „All-Agents-Place"* über den **MASAFinder** beim *Naming Service.*

Bei Beendigung des Agenten-Systems werden *Place* und Agenten-System in **terminate_agent_system()** über die **unregister()**-Methoden des **MASAFinders** wieder aus dem *Naming Service* ausgetragen.

Gemäß Kapitel 3.2.4 wurde die Methode **get_MAFFinder()** angepasst. Sie gibt jetzt eine Referenz auf den Teil des Agenten-Systems als CORBA-Objekt zurück, das die Methoden des Interfaces **FinderService** implementiert.

In der Methode **find_nearby_agent_system_of_profile()** wurde der gesuchte Datentyp **AgentProfile** auf den für die Suche notwendigen Datentyp **AgentsystemInfo** abgebildet.

Mit diesem Datentyp kann über `lookup_agent_system()` das entsprechende Agenten-System gesucht werden.
Die Funktion `list_all_places()` liefert derzeit nur den einzigen implementierten *Place*, den *Default-Place „AllAgents-Place“*.

In `getAgentSystemIOR()` wird die IOR eines Agenten-Systems über die Suche im alten *Name Graph* erhalten. Parallel dazu wurde die Suche über `lookup_agent_system()` im neuen *Name Graph* in der Methode realisiert und über die Variable `newNameGraph = false` deaktiviert, solange der alte *Name Graph* funktionsfähig bleibt. Bei Ausbau des alten *Name Graphen* muss die Variable `true` gesetzt werden bzw. der zugehörige Code entfernt werden.

5.3.4 Änderungen in der Klasse AgentManager

In der Klasse `AgentManager` musste lediglich die Abfrage, ob der Agent bereits als globaler Agent existiert in den Methoden `create_agent()` und `deserializeAgent()` so ergänzt werden, dass auch die Suche im neuen *Name Graph* erfolgt.

5.3.5 Änderungen in der Klasse AgentEnvironment

Auch in der Klasse `AgentEnvironment` begrenzen sich die Eingriffe auf wenige Funktionen. Die Methode `bindAgentToNamingService()` musste mit einem Aufruf von `masa_register_agent()` zur Registrierung des Agenten und die Methode `unbindAgentFromNamingService()` mit einem Aufruf von `unregister_agent()` zur Löschung des Agenten ergänzt werden.

Darüber hinaus wurde die Methode `getAgentProfile()` zur Abfrage des Eigenschafts-profils `AgentProfile` beim Agenten selbst als CORBA-Methode umfunktioniert (s. DE-12). Sie wurde in das IDL-Interface *`AgentService`* aufgenommen.

5.3.6 Änderungen in der Klasse ASManagementAgent

In der Klasse `ASManagementAgent` wurde die Methode `identify_agentsystems()` so umgebaut, dass die Liste aktiver *Agent Systems* über `lookup_agent_system()` nun auch aus dem neuen *Naming Tree* erhältlich ist. Auch hier ist das entsprechende Code-Segment noch über die Variable `newNameGraph = false` deaktiviert, um Konflikte zu vermeiden, solange der alte *Name Graph* noch besteht.

5.3.7 Tool-Klassen

Weitergehende Ergänzungen waren in der Klasse `NameWrapper` notwendig, um den `MASAFinder` hinsichtlich der erforderlichen Manipulation von Namen und der Umwandlung in und aus dem URI-Format zu unterstützen. Im einzelnen wurden folgende Methoden zusätzlich erstellt:

- `NameComponent[] getNameComponents( Name name)`
  Erzeugt aus dem übergebenen Namen einen entsprechenden *Compound Name* in der richtigen Reihenfolge des Namensbaumes.

- `String name_to_mafuri( NameComponent[] nc)`
  Erzeugt aus den Namens-Komponenten ein entsprechendes URI-Format.

- `String name_to_mafuri( NameComponent[] nc, String mafuri)`
  Verbindet eine bestehende `mafuri` mit einer neu erzeugten `mafuri`.

- `NameComponent[] mafuri_to_components( String mafuri)`
  Erzeugt aus einer `mafuri` die entsprechenden Namens-Komponenten.

- `String mafuri_to_IOR( String mafuri)`
  Isoliert die IOR aus einer `mafuri`.

- `boolean compare_names( Name name1, Name name2)`
  Prüfung zweier Namen auf Gleichheit.

Darüber hinaus wurde der MASA-*Logging*-Mechanismus, die Klasse `Debug`, um die klassenbezogene Ausgabe von Meldungen für die Klassen `MASAFinder` und `NameWrapper` (`Facility_MASAFINDER`, `Facility_NAMEWRAPPER`) ergänzt.

Anmerkung:
Das revidierte Dokument [OMG 98-10-11] zum CORBA *Naming Service*, das den *Interoperable Naming Service* beschreibt, standardisiert u.a. die `String`-Darstellung des `CosNaming::Name`-Formats und definiert URL-Formate. Im zugehörigen Interface `NamingContextExt` werden zusätzlich entsprechende Umwandlungs-Methoden spezifiziert. Zukünftige CORBA Implementierungen bauen u.U. auf diesen Verbesserungen bereits auf.

5.4 Tipps zur Implementierung

Anbei einige Tipps, die sich aus dem Umgang mit MASA ergeben haben und so nicht aus den bestehenden Informationsquellen abgeleitet werden konnten.

Beim Auschecken der MASA-Quellen aus CVS, beim Übersetzen und Starten von MASA ist auf die Angaben in der MASA „**FAQ-o-matic**“ Verlass.

Zum Starten der *Secure Version* ist zunächst im Verzeichnis **Masa/system/bin** im **genProperties**-File der Parameter `runsecure = true` zu setzen. Das File muss dann im Verzeichnis **Masa/system** mit **make install_props** neu übersetzt werden.

Neue IDL-Files müssen im **make**-File im Verzeichnis
Masa/system/PRODUCTION.default eingetragen werden.

5.5 Zusammenfassung

Die nachfolgende Matrix (s. Tabelle 5.1) lässt eine Überprüfung des Designs auf den Abdeckungsgrad der Anforderungen zu. Es sind die Anforderungen MASIF-xx, MASA-xx und ALLG-xx, den Design-Entscheidungen DE-xx und Implementierungs-Entscheidungen IE-xx gegenübergestellt.

Ein „√" im Feld bedeutet, dass die Anforderung über die Realisierungs-Entscheidung in der zugehörigen Spalte erfüllt wird. Kann eine Anforderung nicht erfüllt werden, enthält die Spalte „Bem." für Bemerkungen die entsprechende Aktivität (TODO-xx) aus Kapitel 6.3. Dort sind die offenen Punkte zusammengefasst, die von zukünftigen Entwicklungsarbeiten zu erledigen sind. Bereits durch die bestehende MASA-Implementierung erfüllte Anforderungen werden in der Spalte Bemerkungen durch „Masa" gekennzeichnet. Werden die Anforderungen implizit durch die Implementierungs- und Analyse-Arbeiten abgedeckt, steht unter Bemerkungen „Finder".

	DE -01	DE -02	DE -03	DE -04	DE -05	DE -06	DE -07	DE -08	DE -09	DE -10	DE -11	DE -12	DE -13	IE-01	IE-02	Bem.
MASIF-01																Masa TODO-01
MASIF-02															√	TODO-02
MASIF-03															√	
MASIF-04		√														Finder
MASIF-05		√	√	√												Finder
MASIF-06																Masa
MASIF-07		√														Finder
MASIF-08	√															
MASIF-09	√															
MASIF-10	√															
MASIF-11	√															Finder
MASIF-12																Finder
MASIF-13																Finder
MASIF-14																Finder
MASIF-15																Finder
MASA-01					√											
MASA-02								√								TODO-01
MASA-03																TODO-01
MASA-04																TODO-01
MASA-05					√											
MASA-06																Finder
MASA-07														√		
MASA-08																Finder
MASA-09																Finder
ALLG-01	√															

Tabelle 5.1: Matrix der Anforderungen und Realisierungs-Entscheidungen

Zusammenfassend ist festzustellen, dass die Matrix eine relativ dünne Besetzung aufweist. Dies ist allerdings nicht ungewöhnlich, da die Design- und Implementierungs-Entscheidungen in der Regel nicht unbedingt auf die jeweiligen Anforderungen passen müssen. Wichtig ist vor allem die vollständige Abdeckung aller Anforderungen, sei es durch eben die Entwurfs-Entscheidungen oder implizit durch die Realisierung selbst. Die Form der Matrix bietet hier lediglich eine Hilfestellung zur übersichtlichen Prüfung.

Zum raschen Überblick enthält Anhang A nochmals alle Anforderungen und Entscheidungen im Kurztext.

6 Zusammenfassung und Ausblick

6.1 Ergebnis

Mit dieser Arbeit wurde für die *Mobile Agent System Architecture* das Interface *MAFFinder* implementiert. MASA besitzt dadurch einen MASIF-konformen Zugriff auf den CORBA *Naming Service*. Darüber hinaus wurde der *Name Graph* von MASA so angepasst, dass Agenten-Systeme, *Places* und Agenten entsprechend ihrer Einbettungsbeziehung mit ihrem vollständigen, von MASIF definierten Namen eingetragen werden können. Damit ist auch eine Gruppierung von Agenten-Systemen nach *Regions* möglich. Die Funktionalität des *MAFFinders* gestattet es nunmehr, durch eine Traversierung des *Naming Tree* Agenten-Systeme und Agenten nach bestimmten Eigenschaften, nach der Zugehörigkeit zu einer bestimmten *Authority* oder eines Agenten-System-Typs oder aber nach dem Gattungsnamen aufzufinden. Daneben wurde mit dieser Arbeit die Problematik im Hinblick auf das Konzept der *Places* und der eindeutigen Benennung von Agenten erörtert. Für die Benennung von Agenten wurden Lösungsvorschläge gegeben. Insgesamt wurde mit der vorliegenden Arbeit MASA um ein Stück Konformität zur MASIF-Spezifikation und damit zur weitergehenden Interoperabilität bereichert.

6.2 Ausbaumöglichkeiten

Mit **ORBacus Version 4** und dem darin enthaltenem Feature **ORBacus Names** wird eine sog. *Names Console* zur Verfügung gestellt, die über ein *Windows-Explorer* ähnliches *User-Interface* die Administration des *Naming Service* angelehnt an der Administration von Datei-Verzeichnissen erlaubt. Neben *cut&paste*-Funktionen können *Name Bindings* angelegt und gelöscht werden, Kontexte verknüpft werden, Namens-Komponenten geändert werden und vieles mehr. Details können auf der OOC-Homepage (http://www.ooc.com/) nachgelesen werden.

Mit der Einführung von **ORBacus 4** könnte der `MASAFinder` mit einer komfortablen Benutzeroberfläche ausgestattet werden.

Unter den CORBA *Services* ist auch der sog. *Property Service* spezifiziert, der eigentlich dazu gedacht ist, Objekten zusätzliche Attribute (*Properties*) zuweisen zu können, die nicht in ihrem IDL-Interface vorgesehen sind. Der *Property Service*, dessen Funktionsweise mit dem *Naming Service* vergleichbar ist, erlaubt vereinfacht die Definition einer sog. *Property* vom Datentyp `Any` und die Zuweisung eines Namens zum Auffinden der *Property*.

Obwohl die Eigenschaftsprofile `AgentSystemInfo` und `AgentProfile` über die IDL-Schnittstelle ihrer Objekte zugreifbar sind, wäre es vorstellbar, den *Property Service* als zusätzliche Datenbank für den `MASAFinder` zur zentralen Speicherung dieser *Properties* zu benutzen. Die Eigenschaftsprofile werden dem `MASAFinder` über die `register()`-Methoden übergeben und könnten als *Property* z.B. mit dem eindeutigen Namen ihres zugehörigen Objektes im *Property Service* gespeichert werden. Die Möglichkeit der Nutzung des *Property Service* ist anhand seiner Spezifikation [OMG 97-12-20] noch genauer zu untersuchen.

Darüber hinaus könnte der *Property Service* evtl. als Speichermedium für den im Kapitel 4.2 angedachten Instanzen-Zähler für multiple Agenten-Instanzen dienen.

6.3 Offene Punkte

Während der Implementierung des *MAFFinders* wurden eine Reihe offener Punkte identifiziert, die den Rahmen dieser Arbeit gesprengt hätten. Sie wurden nachfolgend mit einer laufenden Nummer versehen und können Gegenstand zukünftiger Verbesserungs- und Weiterentwicklungs-Maßnahmen sein:

TODO-01:
Global eindeutige Benennung der Identity von Agenten.

TODO-02:
Ausarbeitung und Umsetzung eines Konzeptes für die Einführung von Places in MASA, inkl. ihrer Benennung.

TODO-03:
Überprüfung aller MASA-Agenten und Anpassung an die MAFFinder-Schnittstelle.

TODO-04:
Ausbau des alten Name Graphen und der darauf bezogenen Funktionen nach einer entsprechenden Testphase des MASAFinders.

TODO-05:
Feingranulare Anpassung der MASAFinder-Exceptions. (In der Implementierung werden zwar alle Exceptions geeignet abgefangen, z.T. wäre aber eine wesentlich feinere Staffelung der Ausnahmebehandlung möglich. Auch die an darauf zugreifende Anwendungen weiterzugebenden Exceptions, die über das MAF-Interface gegeben sind, könnten über benutzerdefinierte Exceptions noch an Aussagegehalt bereichert werden, z.B. Ausbau der Exception „NameInvalid" der register()-Methoden.)

TODO-06:
Umstellung des ORB auf ORBacus-Version 4, um von dem dort enthaltenen Naming Service Browser als GUI für den MASAFinder zu profitieren.

TODO-07:
Einführung geeigneter Property-Objekte für Agenten-Systeme und Agenten, um die Suchmöglichkeiten zu verbessern.

TODO-08:
Einführung des CORBA Property Services, um auf Properties von CORBA-Objekten, speziell auf die Eigenschaftsprofile Agent System Info und Agent Profile besser zugreifen zu können.

A) Anforderungen, Realisierungs-Entscheidungen und offene Punkte

Anforderungen aus MASIF

MASIF-01:
Die Benennung von Agenten und Agenten-Systemen besteht aus den drei Komponenten Authority, Agent System Type und Identity. Die Identity muss Objekte innerhalb einer Authority eindeutig unterscheiden. Der aus den drei Komponenten zusammengesetzte Name muss Objekte global eindeutig unterscheiden.

MASIF-02:
Es ist folgende Einbettungsbeziehung durchzusetzen:
Ein Endsystem enthält ein oder mehrere Agenten-Systeme. Ein Agenten-System enthält einen oder mehrere Places, mindestens den sog. Default-Place. Ein Place ist die Ausführungsumgebung eines oder mehrerer Agenten.

MASIF-03:
Ein Place besitzt einen (hinsichtlich seiner Struktur nicht näher spezifizierten) Namen.

MASIF-04:
Eine Location spezifiziert einen Place und die Adresse des zugehörigen Agenten-Systems.

MASIF-05:
Die Location eines Agenten bezeichnet die Adresse (Pfad) bzw. den Namen eines Agenten-Systems und eines Places (bzw. Default) auf dem der Agent gestartet wurde.

MASIF-06:
Der Compound Name eines Agenten/Agenten-Systemes ist die Kombination der einzelnen Namens-Komponenten Authority, Agent System Type und Identity. Die einzelne Namens-Komponente ist jeweils ein Objekt vom Typ `CosNaming.Name`

MASIF-07:
Die Location ist vom Datentyp `String` *und enthält eine URI oder URL.*

MASIF-08:
Unterschiedliche Regions können sich einen MAFFinder teilen. Zur Vereinfachung gibt es aber mindestens einen MAFFinder pro Region.

MASIF-09:
Das MAFFinder-Objekt kann – muss aber nicht - im CORBA Naming Service eingetragen werden.

MASIF-10:
Eine Referenz für ein MAFFinder-Objekt erhält ein Client entweder vom CORBA Naming Service oder über die Methode `get_MAFFinder()` *eines Agent Systems.*

MASIF-11:
Implementierung des IDL-Interfaces MAFFinder.

MASIF-12:
Die `lookup_agent()`*-Methode bzw.* `lookup_agent_system()`*-Methode ist so zu implementieren, dass nach beiden Suchparametern gleichzeitig oder nach einem gesucht werden kann. Nicht einschränkende Parameter oder deren Bestandteile sind mit „0“ oder mit einem leeren String zu belegen.*

MASIF-13:
Implementieren der Methode `get_MAFFinder()` *im Agent System.*

MASIF-14:
Implementierung der Methode `find_nearby_agent_system_of_profile()` *unter Berücksichtigung der entsprechenden MAFFinder-Methoden im Agent System .*

MASIF-15:
Implementierung der Methode `list_all_places()` *unter Berücksichtigung der entsprechenden MAFFinder-Methoden im Agent System.*

Anforderungen aus MASA

MASA-01:
Intuitiver Aufbau des Naming Graphen, so dass das Konzept der Regions, Places und der Namens-Bestandteile von Agent System und Agent genutzt werden kann.

MASA-02:
Das Name Binding muss sich zur Vereinfachung am Namenskonzept von MASIF orientieren. Das bestehende MASA Name Binding ist bei der Implementierung eines neuen Name Bindings auf Auswirkungen hinsichtlich der Rückwärtskompatibilität zu prüfen und bei einem neuen Konzept für die Benennung von Agenten-Instanzen zu beachten.

MASA-03:
Die Belegung der Authority-Komponente von Agenten ist auf Durchgängigkeit und Anwendbarkeit bei der Umsetzung eines neuen Name Bindings zu überprüfen.

MASA-04:
Bei einem neuen Konzept für die Benennung von Agenten-Instanzen ist die Adressierung von Applets und die Kommunikation zu seinem Agenten zu beachten. Bei der Erzeugung eines Agenten muss die Trennung zwischen Gattung und Instanz eines Agenten bei der Namensfestlegung beachtet werden.

MASA-05:
Bei der Realisierung des MAFFinders ist das Vorhandensein von Places zu berücksichtigen.

MASA-06:
Die Komponenten des MASA-Basissystemes sind im Hinblick auf die Nutzung der MAFFinder-Schnittstelle zu modifizieren.

MASA-07:
Einhaltung der MASA Sicherheitseigenschaften gemäß der Richtlinien nach [Roel 99] und insbesondere Einhaltung des Konzeptes zur Anwendung der Java Sichtbarkeitsmodifikatoren.
MASA-08:
Bei der Implementierung des MAFFinders's zusätzlich benötigte Funktionen zur Manipulation von Namen sind in den vorhandenen Klassen `NameSpace` *und* `NameWrapper` *zu implementieren.*

MASA-09:
Die Kompatibiltät zu früheren MASA-Versionen und die Lauffähigkeit bestehender Agenten ist sicherzustellen.

Allgemeine Anforderungen

ALLG-01:
Bei der MAFFinder-Implementierung darf kein „Single Point Of Failure" entstehen, der bei Ausfall der Schnittstelle, die Verfügbarkeit des gesamten Agenten-Systems gefährdet.

Design-Entscheidungen

DE-01:
Der CORBA-eigene Naming Service bleibt bestehen. Zugriffe darauf erfolgen über die MAFFinder-Schnittstelle. Der MAFFinder selbst wird Teil des Agent System. Der MAFFinder wird kein eigenes CORBA-Objekt, sondern stellt seine Methoden über das CORBA-Objekt Agent System zur Verfügung. Jedes Agent System erhält einen MAFFinder.

DE-02:
Der Parameter `location` *enthält eine URI. Die IOR wird mit einem führenden Slash („/<IOR>") an die URI angehängt.*

DE-03:
Der Rückgabewert `location` *der MAFFinder-Methoden* `lookup()` *ist ebenfalls eine URI. Die mitgelieferte IOR bezeichnet spezifikationsgemäß das Agent System auf dem das Objekt läuft.*

DE-04:
Um die IOR des Agenten selbst zu erhalten wird eine zusätzliche Methode `lookup_agentIOR()` *implementiert.*

DE-05:
Der neue Naming Graph kann realisiert werden.

DE-06:
Es wird ein Naming Graph für globale Agenten realisiert. Die Struktur lehnt sich an den Compound Name des Agenten an. Der globale Agent wird darin zusätzlich zum eigentlichen Naming Graph registriert.

DE-07:
Zur Unterscheidung bei der Registrierung und Suche exklusiver Agenten werden die beiden Methoden `masa_register_agent()` *und* `masa_lookup_agent()` *eingeführt. Sie unterscheiden sich von den korrespondierenden Methoden der MAFFinder-Spezifikation nur um den zusätzlichen Parameter* `is_global`.

DE-08:
Im letzten durch den jeweiligen Compound Name des Objektes festgelegten Kontext werden diese nur mit ihrem Identity-Attribut gebunden.

DE-09:
Der Kontext zu einer Agent System Identity erhält als `kind`*-Attribut "*`ctx`*".*

DE-10:
Die MAFFinder-Methoden werden nicht als Thread's realisiert. Der MAFFinder selbst sieht aber eine run-Methode für spätere Ergänzungen vor.

DE-11:
Die Implementierung der `unregister_agent()`*-Methode beruht auf der Annahme, dass sich der Agent, dessen Name Binding zu lösen ist, auf dem Agenten-System des MAFFinder's befindet.*

DE-12:
Die Strukturen AgentSystemInfo und AgentProfile werden nicht gespeichert, sondern bei Bedarf über die jeweilige Objektreferenz angefordert.

DE-13:
Bereitstellen einer Methode `get_NameCount()`*, um einem neu zu instanziierendem Agenten einer Gattung ein Unterscheidungsmerkmal für die Identity geben zu können.*

Implementierungs-Entscheidungen

IE-01:
Umsetzung der Sicherheitseigenschaften gemäß MASA-07.

IE-02:
Als Place fungiert die ThreadGroup All-Agents-Place.

Offene Punkte

TODO-01:
Global eindeutige Benennung der Identity von Agenten.

TODO-02:
Ausarbeitung und Umsetzung eines Konzeptes für die Einführung von Places in MASA, inkl. ihrer Benennung.

TODO-03:
Überprüfung aller MASA-Agenten und Anpassung an die MAFFinder-Schnittstelle.

TODO-04:
Ausbau des alten Name Graphen und der darauf bezogenen Funktionen nach einer entsprechenden Testphase des MASAFinders.

TODO-05:
Feingranulare Anpassung der MASAFinder-Exceptions. (In der Implementierung werden zwar alle Exceptions geeignet abgefangen, z.T. wäre aber eine wesentlich feinere Staffelung der Ausnahmebehandlung möglich. Auch die an darauf zugreifende Anwendungen weiterzugebenden Exceptions, die über das MAF-Interface gegeben sind, könnten über benutzerdefinierte Exceptions noch an Aussagegehalt bereichert werden, z.B. Ausbau der Exception „NameInvalid" der register()-Methoden.)

TODO-06:
Umstellung des ORB auf ORBacus-Version 4, um von dem dort enthaltenen Naming Service Browser als GUI für den MASAFinder zu profitieren.

TODO-07:
Einführung geeigneter Property-Objekte für Agenten-Systeme und Agenten, um die Suchmöglichkeiten zu verbessern.

TODO-08:
Einführung des CORBA Property Services, um auf Properties von CORBA-Objekten, speziell auf die Eigenschaftsprofile Agent System Info und Agent Profile besser zugreifen zu können.

B) MAF IDL Interfaces

```
#ifndef _MAFAgentSystem_idl_
#define _MAFAgentSystem_idl_

module CfMAF {
        typedef sequence<octet>           OctetString;
        typedef sequence<OctetString>     OctetStrings;
        typedef OctetString               Authority;
        typedef OctetString               Identity;
        typedef short                     LanguageID;
        typedef short                     AgentSystemType;
        typedef short                     Authenticator;
        typedef short                     SerializationID;
        typedef sequence<SerializationID> SerializationIDList;
        typedef any                       Property;
        typedef sequence<Property>        PropertyList;

        struct Name {
                Authority       authority;
                Identity        identity;
                AgentSystemType agent_system_type;
        };
        typedef sequence<Name> NameList;

        struct AuthInfo {
                boolean       is_authenticated;
                Authenticator authenticator;
        };

        struct LanguageMap {
                LanguageID          language_id;
                SerializationIDList serializations;
        };
        typedef sequence<LanguageMap> LanguageMapList;

        struct AgentSystemInfo {
                Name            agent_system_name;
                AgentSystemType agent_system_type;
                LanguageMapList language_maps;
                string          agent_system_description;
                short           major_version;
                short           minor_version;
                PropertyList    properties;
        };

        struct AgentProfile{
                LanguageID      language_id;
                AgentSystemType agent_system_type;
                string          agent_system_description;
                short           major_version;
                short           minor_version;
                SerializationID serialization;
                PropertyList    properties;
        };

        struct ClassName{
                string      name;
                OctetString discriminator;
        };
        typedef sequence<ClassName> ClassNameList;

        typedef sequence<octet> Arguments;

        typedef string Location;
        typedef sequence<Location> Locations;

        enum AgentStatus {
                CfMAFRunning,
                CfMAFSuspended,
                CfMAFTerminated
        };

        exception AgentNotFound {};
        exception AgentIsRunning {};
```

```
        exception AgentIsSuspended {};
        exception ArgumentInvalid {};
        exception ClassUnknown {};
        exception DeserializationFailed {};
        exception EntryNotFound {};
        exception FinderNotFound {};
        exception MAFExtendedException {};
        exception NameInvalid {};
        exception ResumeFailed {};
        exception SuspendFailed {};
        exception TerminateFailed {};


        interface MAFFinder {

                void register_agent (
                        in Name agent_name,
                        in Location agent_location,
                        in AgentProfile agent_profile)
                        raises (NameInvalid);

                void register_agent_system (
                        in Name agent_system_name,
                        in Location agent_system_location,
                        in AgentSystemInfo agent_system_info)
                        raises (NameInvalid);

                void register_place (
                        in string place_name,
                        in Location place_location) raises (NameInvalid);

                Locations lookup_agent (
                        in Name agent_name,
                        in AgentProfile agent_profile)
                        raises (EntryNotFound);

                Locations lookup_agent_system (
                        in Name agent_system_name,
                        in AgentSystemInfo agent_system_info)
                        raises (EntryNotFound);


                Locations lookup_place (in string place_name)
                        raises (EntryNotFound);

                void unregister_agent (in Name agent_name)
                        raises (EntryNotFound);

                void unregister_agent_system (in Name agent_system_name)
                        raises (EntryNotFound);

                void unregister_place (in string place_name)
                        raises (EntryNotFound);
        };



        interface MAFAgentSystem {
        };
};
#endif
```

C) Die Klasse MASAFinder

```
package __MASA_PACKAGE_agentSystem;

import __MASA_PACKAGE_agent__(*);
import __MASA_PACKAGE_agentSystem__(nameservice.NamingWebServer);
import __MASA_PACKAGE__(tools.*);

import __MASA_PACKAGE_CfMAF__(*);
import org.omg.CosNaming.*;

import java.util.*;
import java.lang.reflect.*;
import java.net.*;
import java.io.*;


/**
 *  Class MASAFinder serves as an interface to the CORBA Naming Service.
 *  In conjunction with the CORBA Naming Service it implements the <CODE>MAFFinder</CODE>.
 *  <P>
 *  It finally implements the methods of the interfaces
 *  <CODE>MAFFinder</CODE> and <CODE>FinderService</CODE> which are
 *  implemented as CORBA methods by <CODE>AgentSystem</CODE>.
 *  <P>
 *  MASAFinder also implements an interface <CODE>java.lang.Runnable</CODE>,
 *  because <CODE>AgentSystem</CODE> will launch a thread for it.
 *  At the moment the thread has no functionality, but it is needed for later use.
 *  <P>
 *  MASAFinder is providing a new naming tree!
 *
 *  @author Josef Gigl
 *
 *  @see AgentSystem
 *  @see MAFAgentSystem
 *  @see FinderService
 *  @see NameWrapper
 *  @see NameSpace
 */
final class MASAFinder implements java.lang.Runnable
{
   /**
        *  Reference to the ORB.
        */
        private final org.omg.CORBA.ORB _orb;

   /**
        *  Reference to the BOA.
        */
        private final org.omg.CORBA.BOA _boa;

   /**
        *  The manager of security certificates.
        */
        private final AgentSystemCertManager _systemCertManager;

   /**
        * Call trust decider.
        */
        private final AgentSystemPermissionManager _callTrustDecider;

   /**
        * Root context of the naming service.
        */
        private final org.omg.CosNaming.NamingContext _initContext;

   /**
        *  MAF name of agent system.
        */
        private Name _agent_system_name;

   /**
        * Context of MAFRegions.
        */
        private org.omg.CosNaming.NamingContext _regionsCtx;
```

```
/**
     * Context of Agent Systems Authority.
     */
     private org.omg.CosNaming.NamingContext _systemAuthCtx;

/**
     * Context of Agent Systems Type.
     */
     private org.omg.CosNaming.NamingContext _systemTypeCtx;

/**
     * Context of Agent Systems Identity.
     */
     private org.omg.CosNaming.NamingContext _systemIdCtx;

/**
     * Context of Agent Systems Places.
     */
     private org.omg.CosNaming.NamingContext _placesCtx;

/**
     * Context of Agent's Authority.
     */
     private org.omg.CosNaming.NamingContext _agentAuthCtx;

/**
     * Context of Agent's System-Type.
     */
     private org.omg.CosNaming.NamingContext _agentTypeCtx;

/**
     * Context of Agent's Identity.
     */
     private org.omg.CosNaming.NamingContext _agentIdCtx;

/**
     * Context of global Agent's.
     */
     private org.omg.CosNaming.NamingContext _agentGlobalCtx;


/**
     *  Reference to us.
     */
     private MASAFinder _masaFinder;


/**
     *  Constants which indicate a certain level of the naming tree
     */
     // Startlevel for list_and_compare() always '0'
     private final static int LEVEL_START   =  0;
     // Levels for contexts of agent system
     private final static int LEVEL_AS_AUTH =  1;
     private final static int LEVEL_AS_TYPE =  2;
     private final static int LEVEL_AS_ID   =  3;
     // Level for context of place
     private final static int LEVEL_PLACE   =  4;
     // Levels for contexts of agent
     private final static int LEVEL_A_AUTH  =  5;
     private final static int LEVEL_A_TYPE  =  6;
     private final static int LEVEL_A_ID    =  7;
     // Levels for contexts of global agent in context 'GlobalAgents'
     private final static int LEVEL_GA_AUTH  =  1;
     private final static int LEVEL_GA_TYPE  =  2;
     private final static int LEVEL_GA_ID    =  3;


/**
     *  Constants which indicate the parameter 'iorLevel' of method list_and_compare()
     *  to decide the right composition of mafuri in method composeMAFURI()
     */
     // Search for agent system, mafuri and ior of agent system
     private final static int AGENTSYSTEM = 1;
     // Search for a place, mafuri and ior of agent system
     private final static int PLACE       = 2;
     // Search for an agent, mafuri of place, ior of agent
```

```
        private final static int AGENT       = 3;
        // Search for an agent, mafuri of place, ior of agent system
        private final static int AGENT_IORAS = 4;
        // Search for agent in 'GlobalAgents' context
        private final static int AGENTGLOBAL = 5;
        // Delivers mafuri of first three contexts without ior
        private final static int DEFAULT     = -1;


//*****************************************************************************************
        //
        // Constructors

   /**
        *  Constructor.
        *  <P>
        *  This one is <CODE>private</CODE>, because it's illegal to call it!
        */
        private MASAFinder()
        {
                _orb                          = null;
                _boa                          = null;
                _systemCertManager            = null;
                _callTrustDecider             = null;
                _masaFinder                   = null;

                _initContext                  = null;
                _regionsCtx                   = null;
                _systemAuthCtx                = null;
                _systemTypeCtx                = null;
                _systemIdCtx                  = null;
                _placesCtx                    = null;
                _agentAuthCtx                 = null;
                _agentTypeCtx                 = null;
                _agentIdCtx                   = null;
                _agentGlobalCtx               = null;

                _agent_system_name            = null;
        }

   /**
        *  Constructor.
        *
        *  @param inSystemCertManager The certificate manager of this agent system
        *  @param inCallTrustDecider  The class which decides wether a call is allowed or
        *                              not
        *  @param orb                 Reference to the ORB shared with
        *                             <CODE>AgentSystem</CODE>
        *  @param boa                 Reference to the BOA shared with
        *                             <CODE>AgentSystem</CODE>
        *  @param initContext         Root context of the naming service
        *  @param agen_system_name    Name of <CODE>AgentSystem</CODE> on which
        *                             <CODE>MASAFinder</CODE> is running
        *
        */
        protected MASAFinder( AgentSystemCertManager          inSystemCertManager,
                              AgentSystemPermissionManager    inCallTrustDecider,
                              org.omg.CORBA.ORB               orb,
                              org.omg.CORBA.BOA               boa,
                              org.omg.CosNaming.NamingContext initContext,
                              org.omg.CfMAF.Name              agent_system_name)
        {
                super();

                _systemCertManager            = inSystemCertManager;
                _callTrustDecider             = inCallTrustDecider;
                _orb                          = orb;
                _boa                          = boa;
                _initContext                  = initContext;
                _agent_system_name            = agent_system_name;

                _masaFinder                   = this;

                // create context "MAFRegions", context for the Agent System on which
                // MASAFinder is running and context for global Agents.
                this.initNamingTree( agent_system_name);
```

```
        }


//*****************************************************************************************
        //
        // Begin private methods


   /**
        *  Create the initial naming tree.<BR>
        *  Create Context "MAFRegions" and context for the Agent System on which MASAFinder
        *  is running.
        *  Create Context for global Agents.
        */
        private void initNamingTree( org.omg.CfMAF.Name agent_system_name)
        {
          try {

              try {
                  // create context of MAFRegions (example 'MAFRegions')
                  org.omg.CosNaming.NameComponent arr1[] =
                                       NameSpace.createNameComponentArray("MAFRegions","");
                  _regionsCtx = _initContext.bind_new_context(arr1);
              }
              catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                  // 'MAFRegions' is already  bound, get the reference of it
                  org.omg.CosNaming.NameComponent arr1[] =
                                       NameSpace.createNameComponentArray("MAFRegions","");
                  org.omg.CORBA.Object obj = _initContext.resolve(arr1);
                  _regionsCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
              }

              initAgentSystemTree( agent_system_name);

              try {
                  // create context of global agents (example 'GlobalAgents')
                  org.omg.CosNaming.NameComponent arr0[] =
                                       NameSpace.createNameComponentArray("GlobalAgents","");
                  _agentGlobalCtx = _initContext.bind_new_context(arr0);
                  listNameContext( _agentGlobalCtx);
              }
              catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                  // GlobalAgents Context is already  bound, get the reference of it
                  org.omg.CosNaming.NameComponent arr0[] =
                                       NameSpace.createNameComponentArray("GlobalAgents","");
                  org.omg.CORBA.Object obj = _initContext.resolve(arr0);
                  _agentGlobalCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
              }

          }
          catch(Exception e){//error
              Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                                "Fatal error creating namespace", e);
              System.exit(1);
          }
        }


   /**
        *  Create the AgentSystemTree.<BR>
        *  Part of the naming tree for AgentSystems of different authority or type.
        */
        private void initAgentSystemTree( org.omg.CfMAF.Name agent_system_name)
        {
          try {

              try {
                  // create context of agent system's authority (example 'UID:gigl')
                  org.omg.CosNaming.NameComponent arr2[] =
                                       NameSpace.createNameComponentArray(
                                                new String(agent_system_name.authority),"");
                  _systemAuthCtx = _regionsCtx.bind_new_context(arr2);
              }
              catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                  // Agent System's Authority is already  bound, get the reference of it
                  org.omg.CosNaming.NameComponent arr2[] =
```

```
                                NameSpace.createNameComponentArray(
                                        new String(agent_system_name.authority),"");
            org.omg.CORBA.Object obj = _regionsCtx.resolve(arr2);
            _systemAuthCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        }

        try {
            // create context of agent system's type (example for MASA: '4')
            org.omg.CosNaming.NameComponent arr3[] =
                NameSpace.createNameComponentArray(
                      String.valueOf((int)agent_system_name.agent_system_type),"");
            _systemTypeCtx = _systemAuthCtx.bind_new_context(arr3);
        }
        catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
            // Agent System's Type is already  bound, get the reference of it
            org.omg.CosNaming.NameComponent arr3[] =
                NameSpace.createNameComponentArray(
                      String.valueOf((int)agent_system_name.agent_system_type),"");
            org.omg.CORBA.Object obj = _systemAuthCtx.resolve(arr3);
            _systemTypeCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        }

      }
      catch(Exception e){//error
          Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                           "Fatal error creating namespace", e);
          System.exit(1);
      }
    }


/**
     *  Create the AgentTree.<BR>
     *  Part of the naming tree for Agents of different authority or type.
     */
    private void initAgentTree( org.omg.CfMAF.Name agent_name,
                                org.omg.CosNaming.NamingContext _placesCtx)
    {
      try {

        try {
            // create context of agent's authority
            // (example for unsecure version 'NO_CERT_PRESENT')
            // (example for secure version: session certificate build from agent
            //  system)
            org.omg.CosNaming.NameComponent arr2[] =
             NameSpace.createNameComponentArray(new String(agent_name.authority),"");
            _agentAuthCtx = _placesCtx.bind_new_context(arr2);
        }
        catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
            // Agent System's Authority is already  bound, get the reference of it
            org.omg.CosNaming.NameComponent arr2[] =
             NameSpace.createNameComponentArray(new String(agent_name.authority),"");
            org.omg.CORBA.Object obj = _regionsCtx.resolve(arr2);
            _agentAuthCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        }

        try {
            // create context of agent's type (example for MASA: '4')
            org.omg.CosNaming.NameComponent arr3[] =
                NameSpace.createNameComponentArray(
                           String.valueOf((int)agent_name.agent_system_type),"");
            _agentTypeCtx = _agentAuthCtx.bind_new_context(arr3);
        }
        catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
            // Agent System's Type is already  bound, get the reference of it
            org.omg.CosNaming.NameComponent arr3[] =
                NameSpace.createNameComponentArray(
                           String.valueOf((int)agent_name.agent_system_type),"");
            org.omg.CORBA.Object obj = _agentAuthCtx.resolve(arr3);
            _agentTypeCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
        }

      }
      catch(Exception e){//error
          Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                           "Fatal error creating namespace", e);
          System.exit(1);
```

```
        }
      }


/**
      *  Search for an object in the CORBA Naming Service.
      *  This method uses the "Depth First Search" algorithm to traverse the naming tree.
      *
      *  @param ctx       context at which the search starts
      *  @param level  the current search level in the tree (for initial call always '0')
      *  @param minLevel  the Level at which the comparison has to start
      *  @param maxLevel  the Level at which the comparison ends and the result has to be
      *                    build
      *  @param iorLevel  level of an item which corresponds to the object found and
      *                    whose ior we have to deliver
      *  @param name      name of the searched object
      *  @param filter    filter to search for, e.g. agent profile or agent system info
      *  @param is_system to decide what type of filter is
      *  @param result    vector with mafuris and iors as a result of the search
      *  @param path      vector with the path to a object found
      *
      *
      *  @return void
      *
      *  @exception
      *
      *  @see
      */
      private void list_and_compare(NamingContext ctx, int level, int minLevel,
                                    int maxLevel, int iorLevel, NameComponent[] name,
                                    java.lang.Object filter, boolean is_system,
                                    java.util.Vector result, java.util.Vector path)
      {
        level++;

        try {
            BindingListHolder bl = new BindingListHolder();
            BindingIteratorHolder blIt= new BindingIteratorHolder();
            BindingHolder bh = new BindingHolder();

            if ( ctx == null) return;

            // list all elements of the context
            ctx.list(0, bl, blIt);
            if ( blIt.value == null){
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                                  "Context has no elements.");
                return;
            }

            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                              "listNameContext" );
            listNameContext( ctx);

            // if there is another element in the context, take it
            while (blIt.value.next_one( bh) == true) {
              Binding element = bh.value;

              // add the element to the path which indicates the object we search
              path.addElement( element);

              // now we have to compare the element name with the name we search
              if (level >= minLevel && level <= maxLevel) {
                if ( compareElement( element, name, level, minLevel) == false) {
                    // if the name doesn't match remove it from path, go back and take
                    // next element
                    path.removeElementAt( path.lastIndexOf( element));
                    continue;
                }
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                                  ((Binding) path.lastElement()).binding_name[0].id);
              }

              // the level where our comparison ends, the object reference can be
              // obtained
              if (level == maxLevel) {
                org.omg.CORBA.Object obj = ctx.resolve(element.binding_name);
                String objStr = _orb.object_to_string(obj);
```

```
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                              "IOR: " + objStr);

            // if there is a search filter we have to compare it
            if ( filter != null) {
              if ( compareFilter( filter, obj, is_system) == false) {
                  // if it doesn't match, remove element from path, go back and take
                  // next one
                  path.removeElementAt( path.lastIndexOf( element));
                  continue;
              }
            }

            // we found a matching element, add path and ior of it to the result
            // vector
            String mafuri = composeMAFURI( iorLevel, path, objStr);
            result.addElement( mafuri);
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                              "MAFURI: " + mafuri);
          }

          // if the element is a context we go on with the search
          if (element.binding_type == BindingType.ncontext) {
            org.omg.CORBA.Object obj = ctx.resolve(element.binding_name);
            NamingContext context =
                  org.omg.CosNaming.NamingContextHelper.narrow(obj);

            list_and_compare( context, level, minLevel, maxLevel, iorLevel, name,
                              filter, is_system, result, path);
          }

          // ok, remove last element from path, go back and take next one
          path.removeElementAt( path.lastIndexOf( element));
        }

        blIt.value.destroy();
      }
      catch(Exception e) {// Error
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                              "Error: Can't proceed search.", e);
      }

      return;
    }


/**
    *  Compares to name components: the name of the current search level
    *  with the name of the component to serach.
    *
    *  @param element    name of the current level
    *  @param name       name to be compared
    *  @param level      the current level
    *  @param minLevel   the level we started to compare
    *
    *  @return boolean   true if element and name component of this level are equal
    *
    *  @exception
    *
    *  @see
    */
    private boolean compareElement( Binding element, NameComponent[] name, int level,
                                    int minLevel)
    {
        int index = level - minLevel;
        int comp1, comp2;

        Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1, "compareE: " +
                          (String) name[index].id);

        if ( name[index] != null && name[index].id.length() > 0) {
            comp1 = ((String) element.binding_name[0].id).compareTo(
                                                        (String) name[index].id);
            if ( comp1 != 0) return false;

            comp2 = ((String) element.binding_name[0].kind).compareTo(
                                                          (String) name[index].kind);
            if ( comp2 != 0) return false;
```

```
        }
        return true;
    }


/**
    *  Compares the properties of an object to search for with the properties
    *  of the object under test.
    *
    *  @param filter      the properties of the searched object, e.g. agent profile or
    *                      agent system info
    *  @param obj         the object we have founded and we have to test for compliance
    *                      with the filter properties
    *  @param is_system   to differnce between agent system and agent properties
    *
    *  @return boolean    if the properties match
    *
    *  @exception
    *
    *  @see
    */
    private boolean compareFilter( Object filter, org.omg.CORBA.Object obj,
                                   boolean is_system)
    {
        if ( is_system) {//filter must be a AgentSystemInfo
            AgentSystemInfo filter_info = null;
            filter_info = (AgentSystemInfo) filter;

            // fetch AgentSystemInfo form agent system itself
            AgentSystemService system_service = AgentSystemServiceHelper.narrow(obj);
            AgentSystemInfo agent_system_info = system_service.get_agent_system_info();

            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1, "IOR: " +
                              _orb.object_to_string(obj));

            // if specific property doesn't match return false

            //if there is a agent system name and
            //the id-component of the identity is not an empty string
            //each part of the name to compare must match
            if ( filter_info.agent_system_name != null &&
                 (new String (filter_info.agent_system_name.identity)).length() >= 0) {
                boolean nametrue = NameWrapper.compare_names(
                                   (Name) filter_info.agent_system_name,
                                   (Name) agent_system_info.agent_system_name);
                if ( nametrue == false) return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "ASName ok!");

            if ( filter_info.agent_system_type != 0) {
                if ( agent_system_info.agent_system_type !=
                     filter_info.agent_system_type)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "ASType ok!");

            boolean match = false;
            if ( filter_info.language_maps != null &&
                 filter_info.language_maps.length >= 0) {
                for (int i=0; i < filter_info.language_maps.length; i++) {
                    if ( filter_info.language_maps[i].language_id != 0) {
                        for (int k=0; k < agent_system_info.language_maps.length; k++){
                            if ( agent_system_info.language_maps[k].language_id ==
                                 filter_info.language_maps[i].language_id) {
                                for (int s=0; s < filter_info.language_maps[i].
                                                  serializations.length;  s++) {
                                    if ( filter_info.language_maps[i].
                                                     serializations[s] != 0) {
                                        for (int t=0; t < agent_system_info.
                                           language_maps[k].serializations.length;t++){
                                            if ( agent_system_info.language_maps[k].
                                                  serializations[t] ==
                                                 filter_info.language_maps[i].
                                                  serializations[s]) match = true;
                                        }
                                    }
```

```
                            else match = true;
                        }
                    }
                }
            }
            else {
                for (int k=0; k < agent_system_info.language_maps.length; k++){
                    for (int s=0; s < filter_info.language_maps[i].
                                        serializations.length; s++) {
                       if (filter_info.language_maps[i].serializations[s] !=0){
                            for (int t=0; t < agent_system_info.
                                                language_maps[k].
                                                serializations.length;t++){
                                if ( agent_system_info.language_maps[k].
                                      serializations[t] ==
                                     filter_info.language_maps[i].
                                      serializations[s]) match = true;
                            }
                        }
                        else match = true;
                    }
                }
            }
        }
        if ( !match) return false;
    }
    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                    "ASLanguage+Serialization ok!");

    if ( filter_info.agent_system_description != null &&
         filter_info.agent_system_description.length() >= 0) {
        if ( agent_system_info.agent_system_description.indexOf
             ( filter_info.agent_system_description) < 0)
            return false;
    }
    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                    "ASDescription ok!");

    if ( filter_info.major_version != 0) {
        if ( agent_system_info.major_version != filter_info.major_version)
            return false;
    }
    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                    "ASMajorVersion ok!");

    if ( filter_info.minor_version != 0) {
        if ( agent_system_info.minor_version != filter_info.minor_version)
            return false;
    }
    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                    "ASMinorVersion ok!");

    if ( filter_info.properties != null &&
         filter_info.properties.length >= 0) {
        for (int i=0; i < filter_info.properties.length; i++) {
            for (int k=0; k < agent_system_info.properties.length; k++) {
                if ( filter_info.properties[i].equal(
                      agent_system_info.properties[k]) == false)
                    return false;
            }
        }
    }
    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                    "ASProperties ok!");

    return true;
}

else {// filter is of type AgentProfile
    AgentProfile filter_profile = null;
    filter_profile = (AgentProfile) filter;

    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                    "IOR: " + _orb.object_to_string(obj));

    // fetch AgentProfile from the agent itself
    AgentService agent = AgentServiceHelper.narrow(obj);
    AgentProfile agent_profile =  agent.getAgentProfile();
```

```
            // if specific property doesn't match return false

            if ( filter_profile.language_id != 0) {
                if ( agent_profile.language_id != filter_profile.language_id)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentLanguage ok!");

            if ( filter_profile.agent_system_type != 0) {
                if ( agent_profile.agent_system_type !=
                     filter_profile.agent_system_type)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentType ok!");

            if ( filter_profile.agent_system_description != null &&
                 filter_profile.agent_system_description.length() >= 0) {
                if ( agent_profile.agent_system_description.indexOf
                     ( filter_profile.agent_system_description) < 0)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentDescription ok!");

            if ( filter_profile.major_version != 0) {
                if ( agent_profile.major_version != filter_profile.major_version)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentMajorversion ok!");

            if ( filter_profile.minor_version != 0) {
                if ( agent_profile.minor_version != filter_profile.minor_version)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentMinorVersion ok!");

            if ( filter_profile.serialization != 0) {
                if ( agent_profile.serialization != filter_profile.serialization)
                    return false;
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentSerialization ok!");

            if ( filter_profile.properties != null &&
                 filter_profile.properties.length >= 0) {
                for (int i=0; i < filter_profile.properties.length; i++) {
                    for (int k=0; k < agent_profile.properties.length; k++) {
                        if ( filter_profile.properties[i].equal(
                               agent_profile.properties[k]) == false)
                            return false;
                    }
                }
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                              "AgentProperties ok!");

            return true;
        }
    }


/**
     *  Compose a MAFURI.
     *
     *  @param iorLevel    marker to choose which ior and mafuri we have to deliver
     *  @param path        the mafuri is composed out of it
     *
     *  @return mafuri
     *
     *  @exception
     *
     *  @see
     */
```

```
private String composeMAFURI( int iorLevel, java.util.Vector path,
                              String objStr_Element)
{

String mafuri = "CosNaming:/";
org.omg.CosNaming.NameComponent name[];
org.omg.CORBA.Object obj;
String objStr = null;
String tmpmafuri = null;

try {

    switch (iorLevel) {

    case 1://Search for a agent System, return mafuri and ior of agent system
        for (int i=0; i<3; i++) {
            mafuri = mafuri + ((Binding) path.get(i)).binding_name[0].id + "!"
                            + ((Binding) path.get(i)).binding_name[0].kind + "/";
        }
        if ( mafuri.lastIndexOf( "ctx") >= 0) {
            tmpmafuri = mafuri.substring( 0, mafuri.lastIndexOf( "ctx")) + "/";
            name = NameWrapper.mafuri_to_components( tmpmafuri);
        }
        else name = NameWrapper.mafuri_to_components( mafuri);
        obj = _regionsCtx.resolve( name);
        objStr = _orb.object_to_string(obj);
        mafuri = mafuri + objStr;
        break;

    case 2://Search for a place, return mafuri and ior of agent system
        for (int i=0; i<3; i++) {
            mafuri = mafuri + ((Binding) path.get(i)).binding_name[0].id + "!"
                            + ((Binding) path.get(i)).binding_name[0].kind + "/";
        }
        if ( mafuri.lastIndexOf( "ctx") >= 0) {
            tmpmafuri = mafuri.substring( 0, mafuri.lastIndexOf( "ctx")) + "/";
            name = NameWrapper.mafuri_to_components( tmpmafuri);
        }
        else name = NameWrapper.mafuri_to_components( mafuri);
        obj = _regionsCtx.resolve( name);
        objStr = _orb.object_to_string(obj);
        mafuri = mafuri + objStr;
        break;

    case 3://Search for an agent, return mafuri of place and ior of agent
        for (int i=0; i<4; i++) {
            mafuri = mafuri + ((Binding) path.get(i)).binding_name[0].id + "!"
                            + ((Binding) path.get(i)).binding_name[0].kind + "/";
        }
        mafuri = mafuri + objStr_Element;
        break;

    case 4://Search for an agent, return mafuri of place and IOR of agent system
        for (int i=0; i<4; i++) {
            mafuri = mafuri + ((Binding) path.get(i)).binding_name[0].id + "!"
                            + ((Binding) path.get(i)).binding_name[0].kind + "/";

            if (i == 2) {
                if ( mafuri.lastIndexOf( "ctx") >= 0) {
                    tmpmafuri = mafuri.substring( 0, mafuri.lastIndexOf( "ctx"))
                                + "/";
                    name = NameWrapper.mafuri_to_components( tmpmafuri);
                }
                else name = NameWrapper.mafuri_to_components( mafuri);
                obj = _regionsCtx.resolve( name);
                objStr = _orb.object_to_string(obj);
            }
        }
        mafuri = mafuri + objStr;
        break;

    case 5://Search for an agent in 'GlobalAgents' Context only
        for (int i=0; i<3; i++) {
            mafuri = mafuri + ((Binding) path.get(i)).binding_name[0].id + "!"
                            + ((Binding) path.get(i)).binding_name[0].kind + "/";
        }
        name = NameWrapper.mafuri_to_components( mafuri);
        obj = _agentGlobalCtx.resolve( name);
```

```
            objStr = _orb.object_to_string(obj);
            mafuri = mafuri + objStr;
            break;

        default://Deliver only the name components in mafuri format, no IOR
            for (int i=0; i<3; i++) {
                mafuri = mafuri + ((Binding) path.get(i)).binding_name[0].id + "!"
                    + ((Binding) path.get(i)).binding_name[0].kind + "/";
            }
            break;
        }
    } catch (Exception e) {//Error in resolving name or other reasons
        Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                          "Can't compose mafuri.", e);
    }

        return mafuri;
    }


/**
    *  List the contents of a naming context.
    *  Only for Debugging.
    *
    *  @param initContext    context which is to be listed
    *
    *  @return void
    *
    *  @exception
    *
    *  @see
    */
        public void listNameContext( org.omg.CosNaming.NamingContext initContext)

    {
        org.omg.CosNaming.NamingContext nc = initContext;

        BindingListHolder bl = new BindingListHolder();
        BindingIteratorHolder blIt= new BindingIteratorHolder();
        BindingHolder bh = new BindingHolder();

        try {

            nc.list(0, bl, blIt);
            if ( blIt.value == null){
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_INFO,
                                  "Context has no elements.");
                return;
            }

            while (blIt.value.next_one( bh) == true) {
                Binding binding = bh.value;


                // get the object reference for each binding
                //    org.omg.CORBA.Object obj = nc.resolve(binding.binding_name);
                //    String objStr = _orb.object_to_string(obj);
                int lastIx = binding.binding_name.length-1;

                // check to see if this is a naming context
                if (binding.binding_type == BindingType.ncontext) {
                    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_INFO,
                      "MASAFinder 'Context: " + binding.binding_name[lastIx].id);
                    System.out.println( "Context: " + binding.binding_name[lastIx].id);
                } else {
                    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_INFO,
                      "MASAFinder 'Object: " + binding.binding_name[lastIx].id);
                    System.out.println("Object: " + binding.binding_name[lastIx].id);
                }
            }
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_INFO,
                                  "MASAFinder finished reading context.");

            blIt.value.destroy();

        } catch (Exception e) {//Error
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                              "Can't connect to Context.", e);
```

```
            }

        }


//*****************************************************************************************
        //
        // CORBA methods defined in MAFAgentSystem (or completed in FinderService)


    /**
        *  Register an agent with the CORBA Naming Service.
        *  The method lookup_agent() from interface MAFFinder was completed with parameter
        *  is_global and redefined in interface FinderService.
        *
        *  @param callingPrincipal   for authorisation purposes
        *  @param agent_name         Name of the agent
        *  @param agent_location     contains mafuri of path to agent system and place on
        *                            which agent was started and the IOR of the agent itself
        *  @param agent_profile      Informations/properties of the agent
        *  @param is_global          indicates whether the agent is a global agent or not
        *
        *  @return void
        *
        *  @exception NameInvalid    If register is not successful
        *
        *  @see MAFAgentSystem
        */
        public void masa_register_agent( Principal_MASA_Internal callingPrincipal,
                                         Name agent_name,
                                         String agent_location,
                                         AgentProfile agent_profile,
                                         boolean is_global)
            throws NameInvalid
        {
            _callTrustDecider.DecideTrust_AS_Java();
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                              "... was called.");

            try {
                org.omg.CosNaming.NameComponent agent[] =
                                  NameWrapper.getNameComponents( agent_name);

                // get the name of agent system and place out of mafuri
                org.omg.CosNaming.NameComponent agent_address[] =
                                  NameWrapper.mafuri_to_components( agent_location);
                // add "ctx" to identify agent system identity as a context
                agent_address[2].kind = "ctx";

                //get the context for the place, where the agent has to be registered
                org.omg.CORBA.Object obj = _regionsCtx.resolve( agent_address);
                _placesCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);

                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                                  "IOR: " + _orb.object_to_string(obj));

                // get the ior of the agent out of mafuri
                String ior = NameWrapper.mafuri_to_IOR( agent_location);

                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG5,
                                  "IOR: " + ior);


                if ( is_global) {
                    try {
                        try {// bind it to the context for global agents
                            _agentGlobalCtx.bind( agent, _orb.string_to_object( ior));
                        }
                        catch ( org.omg.CosNaming.NamingContextPackage.NotFound e) {
                                // The specified contexts do not exist, try to create
                                // contexts for the agent
                          Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                                            "NotFound exception", e);
                          initAgentTree( agent_name, _agentGlobalCtx);
                          _agentGlobalCtx.bind( agent, _orb.string_to_object( ior));
                        }
                        catch(org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
```

```
                        // Another Agent is already up at this host
                      Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                                        "Another agent is already up on this host!", e);
                      throw new NameInvalid();
                    }
                }
                catch(Throwable e){// Error
                    Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                                      "Can't register agent!", e);
                    throw new NameInvalid();
                }
            }

            try {

                try {// bind it to the place context of the corresponding agent system
                  _placesCtx.bind( agent, _orb.string_to_object( ior));
                }
                catch ( org.omg.CosNaming.NamingContextPackage.NotFound e) {
                  // The specified contexts does not exist, try to create contexts for
                  // the agent
                  Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                                    "NotFound exception", e);
                  initAgentTree( agent_name, _placesCtx);
                  _placesCtx.bind( agent, _orb.string_to_object( ior));
                }
                catch(org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                  // Another Agent is already up at this host
                  Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                                    "Another agent is already up on this host!", e);
                  throw new NameInvalid();
                }
            }
            catch(Throwable e){// Error
              Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                                "Can't register agent!", e);
              throw new NameInvalid();
            }

        } catch (Exception e) {//Error
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                              "Can't register agent!", e);
            throw new NameInvalid();
        }

        // only for debugging
        this.listNameContext( _agentTypeCtx);

        return;
    }


/**
     *  Register an agent system with the CORBA Naming Service.
     *
     *  @param callingPrincipal      for authorisation purposes
     *  @param agent_system_name     Name of the agent system
     *  @param agent_system_location IOR of the agent system.
     *  @param agent_system_info     Informations about the agent system:<br>
     *
     *
     *  @return void
     *
     *  @exception NameInvalid
     *
     *  @see MAFAgentSystem
     */
    public void register_agent_system( Principal_MASA_Internal callingPrincipal,
                                       Name agent_system_name,
                                       String agent_system_location,
                                       AgentSystemInfo agent_system_info)
        throws NameInvalid
    {
        _callTrustDecider.DecideTrust_AS_Java();
        Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                          "... was called.");

        org.omg.CosNaming.NameComponent arr[] =
```

```
            NameSpace.createNameComponentArray(
                        new String(agent_system_name.authority),"",
                        String.valueOf((int)agent_system_name.agent_system_type),"",
                        new String(agent_system_name.identity),"");

        org.omg.CosNaming.NameComponent arr1[]=
            NameSpace.createNameComponentArray(
                        new String(agent_system_name.authority),"",
                        String.valueOf((int)agent_system_name.agent_system_type),"",
                        NameWrapper.toString(agent_system_name),"");

        // name component with addition of 'ctx' as identity.kind
        org.omg.CosNaming.NameComponent arr2[] =
            NameSpace.createNameComponentArray(
                        new String(agent_system_name.authority),"",
                        String.valueOf((int)agent_system_name.agent_system_type),"",
                        new String(agent_system_name.identity),"ctx");
        try {

            try {// bind agent system
                _regionsCtx.bind(arr, _orb.string_to_object( agent_system_location));

                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                                  "IOR: " + agent_system_location);

            }
            catch ( org.omg.CosNaming.NamingContextPackage.NotFound e) {
                // The specified contexts does not exist, try to create contexts for
                // the agent system
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                                 "NotFound exception", e);
                initAgentSystemTree( agent_system_name);
                _regionsCtx.bind(arr, _orb.string_to_object( agent_system_location));
            }
            catch(org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                // Another Agent-system is already up at this host, so we clean up
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                  "Another agent system is already up on this host,so I terminate", e);
                throw new NameInvalid();
            }

            // Create AgentSystemIdentity-Context for Agents and Places to register
            try {
                // create context of agent system's identity (example:
                // 'pcheger0.nm.informatik.uni-muenchen.de')
                _systemIdCtx = _regionsCtx.bind_new_context(arr2);
            }
            catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                // Agent System's Type is already  bound, get the reference of it
                org.omg.CORBA.Object obj = _regionsCtx.resolve(arr2);
                _systemIdCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);
                }

        }
        catch(Throwable e){// Error
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
            "Another agent system is already up on this host,so I terminate", e);
            throw new NameInvalid();
        }

        // only for debugging
        this.listNameContext( _systemTypeCtx);
        this.listNameContext( _systemIdCtx);

        return;

    }


/**
     *  Register a place with the CORBA Naming Service.
     *  According to MASIF place is the runtime environment for agents.
     *  For MASA place is defined as a thread group in which the thread of an agent
     *  should be started.
     *  The default place e.g. is "AllAgents-Place".
     *
     *  @param callingPrincipal   for authorisation purposes
     *  @param place_name         Name of the place.
```

```
     *  @param place_location     mafuri which contains Name of the agent system on
     *                             which the place resides.
     *
     *  @return void
     *
     *  @exception NameInvalid
     *
     *  @see
     */
     public void register_place( Principal_MASA_Internal callingPrincipal,
                                 String place_name,
                                 String place_location)
         throws NameInvalid
     {
         _callTrustDecider.DecideTrust_AS_Java();
         Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                         "... was called.");

         try {

             org.omg.CosNaming.NameComponent place[] =
                 NameSpace.createNameComponentArray( new String( place_name),"");

             // get the name of the corresponding agent system to which place belongs
             NameComponent[] nc = NameWrapper.mafuri_to_components( place_location);

             org.omg.CosNaming.NameComponent agent_system_name[] =
                                                          { nc[0], nc[1], nc[2]};
             agent_system_name[2].kind = "ctx";

             org.omg.CORBA.Object obj = _regionsCtx.resolve(agent_system_name);
             _systemIdCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);

             try {
                 // Create Place-Context for Agents to register
                 try {
                     _placesCtx = _systemIdCtx.bind_new_context( place);
                 }
                 catch( org.omg.CosNaming.NamingContextPackage.AlreadyBound e){
                      // Agent System's Type is already  bound, get the reference of it
                      org.omg.CORBA.Object objpl = _systemIdCtx.resolve( place);
                      _placesCtx = org.omg.CosNaming.NamingContextHelper.narrow(objpl);
                 }

             }
             catch(Throwable e){// Error
                  Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_FATAL,
                  "Not able to register place.", e);
                  throw new NameInvalid();
             }
         } catch (Exception e) {
             Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                             "Can't register.", e);
             throw new NameInvalid();
         }

         // only for debugging
         this.listNameContext( _systemIdCtx);
         return;
     }


/**
     *  Search for an agent in the CORBA Naming Service.
     *  The method lookup_agent() from interface MAFFinder was completed with parameter
     *  is_global and
     *  redefined in interface FinderService.
     *
     *  @param callingPrincipal   for authorisation purposes
     *  @param agent_name         Name of the agent
     *  @param agent_profile      Informations about the agent to look for, properties.
     *  @param is_global          indicates whether the agent is a global agent or not
     *
     *  @return locations  IORs of agent systems on which possible agents are running;
     *                  name or profile of the agents must match with the search filter.
     *
     *  @exception EntryNotFound There was no match of specified features
     *
```

```
     *  @see MAFAgentSystem
     */
     public String[] masa_lookup_agent( Principal_MASA_Internal callingPrincipal,
                                        Name agent_name,
                                        AgentProfile agent_profile,
                                        boolean is_global)
         throws EntryNotFound
     {
         _callTrustDecider.DecideTrust_AS_Java();
         Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                           "... was called.");

         String[] locations = null;
         // temporary vector which contains the path to the searched object in the
         // naming tree
         java.util.Vector path = new Vector(0,1);
         // temporary vector which contains the locations of objects found
         java.util.Vector result = new Vector(0,1);
         // the search filter
         AgentProfile filter = new AgentProfile();
         boolean is_system = false;

         org.omg.CosNaming.NameComponent name[] =
                           NameWrapper.getNameComponents( agent_name);

         // Lookup only GlobalAgents Context to see if the agent is already living and
         // return!
         if ( is_global) {
             try {
                 filter = null;
                 NamingContext ctx = _agentGlobalCtx;

                 // traverse the tree 'GlobalAgents'
                 list_and_compare( ctx, LEVEL_START, LEVEL_GA_AUTH, LEVEL_GA_ID,
                                   AGENTGLOBAL, name, filter, is_system, result, path);

                 // copy the vector result into locations
                 int s = result.size();
                 locations = new String[s];
                 result.copyInto( locations);

             }
             catch (Exception e) {
                 Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                                   "Can't lookup agent!", e);
                 throw new EntryNotFound();
             }
             if ( locations == null || locations.length == 0) throw new EntryNotFound();

             return locations;
         }

         try {
             if ( agent_profile != null) filter = agent_profile;
             else filter = null;

             // traverse the tree 'MAFRegions'
             list_and_compare( _regionsCtx, LEVEL_START, LEVEL_A_AUTH, LEVEL_A_ID,
                               AGENT_IORAS, name, filter, is_system, result, path);

             // copy the vector result into locations
             int s = result.size();
             locations = new String[s];
             result.copyInto( locations);

         } catch (Exception e) {
             Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                               "Can't lookup agent!", e);
             throw new EntryNotFound();
         }

         return locations;
     }


/**
     *  Search for an agent system in the CORBA Naming Service.
     *
```

```
     *  @param callingPrincipal      for authorisation purposes
     *  @param agent_system_name     Name of the agent system
     *  @param agent_system_info     Informations about the agent system, properties.
     *
     *  @return locations    IOR's of agent systems which match name or info.
     *
     *  @exception EntryNotFound There was no match of specified features.
     *
     *  @see MAFAgentSystem
     */
     public String[] lookup_agent_system( Principal_MASA_Internal callingPrincipal,
                                          Name agent_system_name,
                                          AgentSystemInfo agent_system_info)
         throws EntryNotFound
     {
         _callTrustDecider.DecideTrust_AS_Java();

         String[] locations = null;
         // temporary vector which contains the path to the searched object in the
         // naming tree
         java.util.Vector path = new Vector();
         // temporary vector which contains the locations of objects found
         java.util.Vector result = new Vector();
         // the search filter
         AgentSystemInfo filter = new AgentSystemInfo();
         boolean is_system = true;

         org.omg.CosNaming.NameComponent name[] =
                           NameWrapper.getNameComponents( agent_system_name);

         try {
             if ( agent_system_info != null) filter = agent_system_info;
             else filter = null;

             // traverse the tree 'MAFRegions'
             list_and_compare( _regionsCtx, LEVEL_START, LEVEL_AS_AUTH, LEVEL_AS_ID,
                               AGENTSYSTEM, name, filter, is_system, result, path);

             // copy the vector result into locations
             int s = result.size();
             locations = new String[s];
             result.copyInto( locations);

         } catch (Exception e) {
             Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                               "Can't lookup agent system!", e);
             throw new EntryNotFound();
         }

         return locations;
     }


/**
     *  Search for a place in the CORBA Naming Service.
     *
     *  @param callingPrincipal   for authorisation purposes
     *  @param place_name         Name of the place.
     *
     *  @return  locations      IOR's of agent systems for which the place name matches.
     *
     *  @exception  EntryNotFound There was no match of specified features.
     *
     *  @see MAFAgentSystem
     */
     public String[] lookup_place( Principal_MASA_Internal callingPrincipal,
                                   String place_name)
         throws EntryNotFound
     {
         _callTrustDecider.DecideTrust_AS_Java();

         String[] locations = null;
         // temporary vector which contains the path to the searched object in the
         // naming tree
         java.util.Vector path = new Vector();
         // temporary vector which contains the locations of objects found
         java.util.Vector result = new Vector();
         java.lang.Object filter = null;
```

```
        boolean is_system = false;

        org.omg.CosNaming.NameComponent name[] =
                        NameSpace.createNameComponentArray( place_name,"");

        try {
            // traverse the tree 'MAFRegions'
            list_and_compare( _regionsCtx, LEVEL_START, LEVEL_PLACE, LEVEL_PLACE,
                              PLACE, name, filter, is_system, result, path);

            // copy the vector result into locations
            int s = result.size();
            locations = new String[s];
            result.copyInto( locations);

        } catch (Exception e) {
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                            "Can't lookup agent system!", e);
            throw new EntryNotFound();
        }

        return locations;
    }



/**
     *  Unregister an agent with the CORBA Naming Service.
     *  It is assumed that the agent lives on the same agent system as the MASAFinder.
     *
     *  @param callingPrincipal   for authorisation purposes
     *  @param agent_name         Name of the agent
     *
     *  @return  void
     *
     *  @exception EntryNotFound  If the name is not registered.
     *
     *  @see MAFAgentSystem
     */
    public void unregister_agent( Principal_MASA_Internal callingPrincipal,
                                  Name agent_name)
        throws EntryNotFound
    {
      _callTrustDecider.DecideTrust_AS_Java();
      Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                        "... was called.");

      org.omg.CosNaming.NameComponent agent[] =
                        NameWrapper.getNameComponents( agent_name);

      org.omg.CosNaming.NameComponent arr1[] =
        NameSpace.createNameComponentArray( new String( agent_name.authority),"");

      org.omg.CosNaming.NameComponent arr2[] =
        NameSpace.createNameComponentArray( new String( agent_name.authority),"",
                        String.valueOf((int) agent_name.agent_system_type),"");

      org.omg.CosNaming.NameComponent arr3[] =
        NameSpace.createNameComponentArray(
                        new String( _agent_system_name.authority),"",
                        String.valueOf((int) _agent_system_name.agent_system_type),"",
                        new String( _agent_system_name.identity),"ctx");

      try {

        org.omg.CORBA.Object obj = _regionsCtx.resolve( arr3);
        _systemIdCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);

        java.util.Vector path = new Vector();
        java.util.Vector result = new Vector();

        // lookup the agent for getting the corresponding place;
        // subtract '3' from minLevel and maxLevel because we start at _systemIdCtx
        // (level = 3)
        list_and_compare( _systemIdCtx, LEVEL_START, LEVEL_A_AUTH - 3 , LEVEL_A_ID - 3,
                          DEFAULT, agent, null, false, result, path);

        // copy the vector result into locations
```

```
  int s = result.size();
  String[] locations = new String[s];
  result.copyInto( locations);

  org.omg.CosNaming.NameComponent address[] =
                    NameWrapper.mafuri_to_components( locations[0]);
  org.omg.CosNaming.NameComponent place[] =
      NameSpace.createNameComponentArray( address[0].id, address[0].kind);

  org.omg.CORBA.Object objpl = _systemIdCtx.resolve( place);
  _placesCtx = org.omg.CosNaming.NamingContextHelper.narrow(objpl);

  // unbind the identity of the agent
  _placesCtx.unbind( agent);

  org.omg.CORBA.Object objauth = _placesCtx.resolve(arr1);
  _agentAuthCtx = org.omg.CosNaming.NamingContextHelper.narrow(objauth);

  org.omg.CORBA.Object objtype = _placesCtx.resolve(arr2);
  _agentTypeCtx = org.omg.CosNaming.NamingContextHelper.narrow(objtype);

  BindingListHolder bl = new BindingListHolder();
  BindingIteratorHolder blIt= new BindingIteratorHolder();

  // check if there is a binding in the the type context of the agent
  _agentTypeCtx.list(10, bl, blIt);
  Binding bindings[] = bl.value;

  // if not unbind context
  if (bindings.length == 0) {
    _placesCtx.unbind( arr2);
    _agentTypeCtx.destroy();
  }

  // check if there is a binding in the the authority context of the agent
  _agentAuthCtx.list(10, bl, blIt);
  Binding bindings2[] = bl.value;

  // if not unbind context
  if (bindings2.length == 0) {
    _placesCtx.unbind( arr1);
    _agentAuthCtx.destroy();
  }

}
catch(Throwable e){// Error
  Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                    "Error, can't proceed unbinding", e);
  throw new EntryNotFound();
}

// look whether agent is registered in context 'GlobalAgents' and unbind agent
try {
  // unbind the identity of the agent
  _agentGlobalCtx.unbind( agent);

  org.omg.CORBA.Object objtype = _agentGlobalCtx.resolve(arr2);
  _agentTypeCtx = org.omg.CosNaming.NamingContextHelper.narrow(objtype);

  org.omg.CORBA.Object objauth = _agentGlobalCtx.resolve(arr1);
  _agentAuthCtx = org.omg.CosNaming.NamingContextHelper.narrow(objauth);

  BindingListHolder bl = new BindingListHolder();
  BindingIteratorHolder blIt= new BindingIteratorHolder();

  // check if there is a binding in the type context
  _agentTypeCtx.list(10, bl, blIt);
  Binding bindings[] = bl.value;

  // if not unbind context
  if (bindings.length == 0) {
    _agentGlobalCtx.unbind( arr2);
    _agentTypeCtx.destroy();
  }

  // check if there is a binding in the authority context
  _agentAuthCtx.list(10, bl, blIt);
  Binding bindings2[] = bl.value;
```

```
        // if not unbind context
        if (bindings2.length == 0) {
          _agentGlobalCtx.unbind( arr1);
          _agentAuthCtx.destroy();
        }

      }
      catch( org.omg.CosNaming.NamingContextPackage.NotFound e){
        // Agent was not of type global, ok nothing to do here
        Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                          "Error, can't proceed unbinding!", e);
      }
      catch(Throwable e){// Error
        Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                          "Error, can't proceed unbinding!", e);
      }

      // only for debugging
      listNameContext( _placesCtx);
      listNameContext( _agentGlobalCtx);

      return;
    }


/**
    *  Unregister an agent system with the CORBA Naming Service.
    *
    *  @param callingPrincipal    for authorisation purposes
    *  @param agent_system_name   Name of the agent system
    *
    *  @return void
    *
    *  @exception EntryNotFound  If the name is not registered.
    *
    *  @see MAFAgentSystem
    */
    public void unregister_agent_system( Principal_MASA_Internal callingPrincipal,
                                         Name agent_system_name)
        throws EntryNotFound
    {
        _callTrustDecider.DecideTrust_AS_Java();

        org.omg.CosNaming.NameComponent system[] =
                          NameWrapper.getNameComponents( agent_system_name);

        try {
            // unbind identity of agent system
            _regionsCtx.unbind( system);

            org.omg.CosNaming.NameComponent arr3[] =
                NameSpace.createNameComponentArray(
                          new String( agent_system_name.authority),"",
                          String.valueOf((int) agent_system_name.agent_system_type),"",
                          new String( agent_system_name.identity),"ctx");

            org.omg.CORBA.Object obj = _regionsCtx.resolve( arr3);
            _systemIdCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);

            org.omg.CosNaming.NameComponent arr2[] =
                NameSpace.createNameComponentArray(
                         new String( agent_system_name.authority),"",
                         String.valueOf((int) agent_system_name.agent_system_type),"");
            org.omg.CORBA.Object objtype = _regionsCtx.resolve(arr2);
            _systemTypeCtx = org.omg.CosNaming.NamingContextHelper.narrow(objtype);

            org.omg.CosNaming.NameComponent arr1[] =
                NameSpace.createNameComponentArray(
                          new String( agent_system_name.authority),"");
            org.omg.CORBA.Object objauth = _regionsCtx.resolve(arr1);
            _systemAuthCtx = org.omg.CosNaming.NamingContextHelper.narrow(objauth);

            // unbind the context of agent systems identity component
            _regionsCtx.unbind( arr3);
            _systemIdCtx.destroy();

            BindingListHolder bl = new BindingListHolder();
```

```
            BindingIteratorHolder blIt= new BindingIteratorHolder();

            // check if there is a binding in the type context
            _systemTypeCtx.list(10, bl, blIt);
            Binding bindings[] = bl.value;

            // if not unbind context
            if (bindings.length == 0) {
                _regionsCtx.unbind( arr2);
                _systemTypeCtx.destroy();
            }

            // check if there is a binding in the authority context
            _systemAuthCtx.list(10, bl, blIt);
            Binding bindings2[] = bl.value;

            // if not unbind context
            if (bindings2.length == 0) {
                _regionsCtx.unbind( arr1);
                _systemAuthCtx.destroy();
            }

        }
        catch(Throwable e){// Error
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                              "Error, can't proceed unbinding!", e);
            throw new EntryNotFound();
        }

        // only for debugging
        listNameContext( _regionsCtx);

        return;
    }


/**
     *  Unregister a place with the CORBA Naming Service.
     *
     *  @param callingPrincipal   for authorisation purposes
     *  @param place_name         Name of the place.
     *
     *  @return void
     *
     *  @exception EntryNotFound  If the name is not registered.
     *
     *  @see MAFAgentSystem
     */
    public void unregister_place( Principal_MASA_Internal callingPrincipal,
                                  String place_name)
        throws EntryNotFound
    {
        _callTrustDecider.DecideTrust_AS_Java();

        org.omg.CosNaming.NameComponent place[] =
                          NameSpace.createNameComponentArray( place_name,"");

        org.omg.CosNaming.NameComponent arr3[] =
            NameSpace.createNameComponentArray(
                      new String( _agent_system_name.authority),"",
                      String.valueOf((int) _agent_system_name.agent_system_type),"",
                      new String( _agent_system_name.identity),"ctx");

        try {
            org.omg.CORBA.Object objsys = _regionsCtx.resolve( arr3);
            _systemIdCtx = org.omg.CosNaming.NamingContextHelper.narrow(objsys);

            org.omg.CORBA.Object obj = _systemIdCtx.resolve( place);
            _placesCtx = org.omg.CosNaming.NamingContextHelper.narrow(obj);

            BindingListHolder bl = new BindingListHolder();
            BindingIteratorHolder blIt= new BindingIteratorHolder();

            // check if there is a binding in the context
            _placesCtx.list(10, bl, blIt);
            Binding bindings[] = bl.value;

            // if not unbind context
```

```
            if (bindings.length == 0) {
                _systemIdCtx.unbind( place);
                _placesCtx.destroy();
            }

        }
        catch(Throwable e){// Error
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_ERROR,
                              "Error, can't proceed unbinding!", e);
            throw new EntryNotFound();
        }

        // only for debugging
        listNameContext( _systemIdCtx);

        return;
    }


//*****************************************************************************************
    //
    // CORBA methods defined in FinderService only


  /**
    *  Get Number to identify names globally.
    *  Not supported at the moment.
    *
    *  @param callingPrincipal   for authorisation purposes
    *  @param name               name of the agent for which we need a Id-number
    *
    *  @return
    *  @see
    */
    public String getNameCount( Principal_MASA_Internal callingPrincipal,
                                Name name)
        throws EntryNotFound
    {
        _callTrustDecider.DecideTrust_AS_Java();

        String test = null;

        return test;
    }


  /**
    *  Search for an agent in the CORBA Naming Service.
    *  Returns a list of IORs of the agents found instaed of the agent systems IORs.
    *
    *  @param callingPrincipal   for authorisation purposes
    *  @param agent_name         Name of the agent
    *  @param agent_profile      Informations about the agent to look for, properties.
    *
    *
    *  @return locations         IORs of agents which match name or profile.
    *
    *  @exception EntryNotFound  There was no match of specified features
    *
    *  @see
    */
    public String[] lookup_agentIOR( Principal_MASA_Internal callingPrincipal,
                                     Name agent_name,
                                     AgentProfile agent_profile)
        throws EntryNotFound
    {
        _callTrustDecider.DecideTrust_AS_Java();

        String[] locations = null;
        // temporary vector which contains the path to the searched object in the
        // naming tree
        java.util.Vector path = new Vector();
        // temporary vector which contains the locations of objects found
        java.util.Vector result = new Vector();
        // the search filter
        AgentProfile filter = new AgentProfile();
        boolean is_system = false;
```

```
            org.omg.CosNaming.NameComponent name[] =
                                NameWrapper.getNameComponents( agent_name);

            if ( agent_profile != null) filter = agent_profile;
            else filter = null;

            // traverse the tree 'MAFRegions'
            list_and_compare( _regionsCtx, LEVEL_START, LEVEL_A_AUTH, LEVEL_A_ID, AGENT,
                                name, filter, is_system, result, path);

            // copy the vector result into locations
            int s = result.size();
            locations = new String[s];
            result.copyInto( locations);

            return locations;
        }


//*****************************************************************************************
        //
        // package methods
        // at the moment only some methods for testing purposes


   /**
        *  This is a test method!
        *
        *  @param
        *
        *  @return
        *
        *  @see
        */
        protected void test()
        {

            try {
                Thread.currentThread().sleep(10000);
            }
            catch(Throwable e){// Error
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                "MASAFinder Thread sleeps", e);
            }

            String authority = "NO_CERT_PRESENT";
            String identity  = "";
            short type       = 4;

            Name agent_name = new Name( authority.getBytes(), identity.getBytes(), type);

            AgentProfile ap = new AgentProfile();
            ap.properties = new org.omg.CORBA.Any[0];
            ap.agent_system_description = "";
            ap = null;


            String[] loc = null;

            try {
                loc = this.masa_lookup_agent( null, agent_name, ap, false);
            }
            catch(Throwable e){// Error
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                "MASAFinder didn't find entry", e);
            }

            int s = loc.length;

            for (int k=0; k< s; k++){
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1, "Look: " +
                                loc[k]);
            }


            try {
                Thread.currentThread().sleep(3000);
```

```
            }
            catch(Throwable e){// Error
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                "MASAFinder Thread sleeps", e);
            }


            authority = "UID:gigl";
            identity  = "pcheger0.nm.informatik.uni-muenchen.de";
            type       = 4;

            Name agentSys_name = new Name( authority.getBytes(),
                                           identity.getBytes(), type);

            AgentSystemInfo    tmpAgentSystemInfo    = null;
            org.omg.CORBA.Any  emptyProperty       = org.omg.CORBA.ORB.init().create_any();
            org.omg.CORBA.Any  propertyList[]        = { emptyProperty };
            LanguageMap        languageMapList[]     = new LanguageMap[1];
            short              serializationList[]   = new short[1];
            serializationList[0] = SerializationID.JAVA_OBJECT_SERIALIZATION;
            languageMapList[0]   = new LanguageMap( LanguageID.JAVA, serializationList);
            tmpAgentSystemInfo   = new AgentSystemInfo(
                                       agentSys_name,                      //system_name
                                       AgentSystemType.MASA,               //system_type
                                       languageMapList,                    //language_maps
                                       GlobalConstants.masa_system_description,
                                                                           //system_description
                                       GlobalConstants.masa_major_version, //major_version
                                       GlobalConstants.masa_minor_version, //minor_version
                                       propertyList                        //properties
                                                   );

            loc = null;

            try {
                loc = this.lookup_agent_system( null, agentSys_name, tmpAgentSystemInfo);
            }
            catch(Throwable e){// Error
                Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                "MASAFinder didn't find entry", e);
            }

            s = loc.length;

            for (int k=0; k< s; k++){
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1, "Look: " +
                              loc[k]);
            }

            return;

        }


//*****************************************************************************************
        //
        // methods inherited from run interface

   /**
        *  Running thread of this MASAFinder.
        *
        *  @see AgentSystem
        */
        public void run()
        {
            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1,
                             "Begin run MASAFinder.");

            this.test();

            Debug.logMessage( Debug.FACILITY_MASAFINDER, Debug.LEVEL_DEBUG1, "run END");

        }

}
```

D) Die erweiterte Klasse NameWrapper

```
package __MASA_PACKAGE__(tools);

import __MASA_PACKAGE_CfMAF__(*);
import __MASA_PACKAGE_agentSystem__(AgentKind);

import org.omg.CosNaming.NameComponent;

/**
 *  Helper class for <CODE>CfMAF.Name</CODE> CORBA classes.
 *  <P>
 *  This class is needed for three purposes: <br>
 *  <OL>
 *    <LI>This class is taken as the key for the agent table
 *        therefore it implements a method <CODE>equals()</CODE>
 *        and <CODE>hashCode()</CODE> <br>
 *    <LI>Convert the <CODE>Name</CODE> in a <CODE>NameComponent</CODE>i
 *        array for use with the CORBA-Naming-Service.
 *    <LI>Convert <CODE>Name</CODE> into a String an back.
 *
 *  @author Bernhard Kempter
 *  @author Enhanced by Harald Roelle
 *  @author further developped by Josef Gigl
 */
public class NameWrapper
{
//*****************************************************************************************
        // Private attributes

   /**
        *  Start marker for <CODE>Name</CODE> String representation.
        */
        private final static String BEGIN_MARKER = "[";

   /**
        *  End marker for <CODE>Name</CODE> String representation.
        */
        private final static String END_MARKER = "]";

   /**
        *  Marker for <CODE>Name</CODE> part boundaries in stringified representation.
        */
        private final static String PART_DELIM = "!";

   /**
        *  Marker for mafuri part boundaries in stringified representation.
        */
        private final static String SLASH = "/";

//*****************************************************************************************
        //
        // Public attributes

   /**
        *  Reference to the original name.
        */
        public final Name _name;

//*****************************************************************************************
        //
        // Constructors

   /**
        *  Constructor.
        *
        *  @param name name to wrap
        */
        public NameWrapper(Name name)
        {
                _name= name;
        }


   /**
        *  Constructor.
        *
```

```
        *  @param inAuthority        authority element of name to wrap
        *  @param inIdentity         identity element of name to wrap
        *  @param inAgentSystemType  agentSystemType element of name to wrap
        */
        public NameWrapper( String inAuthority, String inIdentity, short inAgentSystemType)
        {
         _name = new Name( inAuthority.getBytes(), inIdentity.getBytes(),
                                                              inAgentSystemType);
        }


    /**
        *  Constructor.
        *  <P>
        *  This class contains a compatibility hack for the old semantics of this
        *  constructor.<BR>
        *  When <CODE>inNameStr</CODE> is enclosed by <CODE>BEGIN_MARKER</CODE> and
        *  <CODE>END_MARKER</CODE>, then <CODE>inNameStr</CODE> is interpreted as a String
        *  representation of <CODE>Name</CODE>. Then it acts as the inverse function of
        *  the encoding done in the <CODE>toString()</CODE> method.<BR>
        *  Else the old semantics apply and <CODE>inNameStr</CODE> is taken as the
        *  <CODE>identity</CODE> part of the <CODE>Name</CODE>.
        *
        *  @param inNameStr String representation of <CODE>Name</CODE> to wrap.
        */
        public NameWrapper( String inNameStr)
        {
                //(roelle): This is a hack for compatibility with the old implementation
                //          of this method.
        if ( inNameStr.startsWith( BEGIN_MARKER) && inNameStr.endsWith( END_MARKER) )
        {
             inNameStr = inNameStr.substring( BEGIN_MARKER.length(),
                                              inNameStr.length()-END_MARKER.length());

             String tokens[] = new String[3];
             tokens[0] = inNameStr.substring( 0, inNameStr.indexOf( PART_DELIM));
             tokens[1] = inNameStr.substring( inNameStr.indexOf( PART_DELIM)+1,
                                              inNameStr.lastIndexOf( PART_DELIM));
             tokens[2] = inNameStr.substring( inNameStr.lastIndexOf( PART_DELIM)+1);

             _name = new Name( tokens[1].getBytes(), tokens[0].getBytes(),
                               Short.parseShort( tokens[2]));
        }
        else
        {
             _name = new Name( new byte[0], inNameStr.getBytes(),
                               AgentSystemType.MASA);
        }
        }


//*****************************************************************************************
        //
        // Public instance methods

    /**
        *  Compare two Objects if they are equal.
        *
        *  @param obj the object to compare to
        *
        *  @return wether obs is equal to this.
        */
        public boolean equals(Object obj)
        {
                if ( !(obj instanceof NameWrapper) )//Correct Type of Object
                        return false;

                //####(roelle): Correct semantics????
                //####(roelle): PROBLEM FOR MULTIPLE INSTANCES OF ONE AGENT
                NameWrapper testName= (NameWrapper)obj;
                int i= 0;
                while (i < _name.identity.length )
                        if ( _name.identity[i] != testName._name.identity[i++] )
                        {
                                return false;
                        }
                return true;
        }
```

```
    /**
         *  Return a hash code for this object.
         *
         *  @return   the hash code for this object
         */
         public int hashCode()
         {
                 //####(roelle): PROBLEM FOR MULTIPLE INSTANCES OF ONE AGENT
                 return (new String( _name.identity)).hashCode();
         }


    /**
         *  String representation of this object.
         *
         *  @return   String
         */
         public String toString()
         {
                 return toString( _name);
         }

    /**
         *  Name representation of this object.
         *
         *  @return   Name
         */
         public Name toName()
         {
                 return _name;
         }


//*************************************************************************************
         //
         // Public static methods

    /**
         *  Create a NameComponent out of a <CODE>Name</CODE>.
         *
         *  @param name          the Name
         *  @param isAgentSystem if this name specifies an agent system
         *
         *  @return    NameComponent
         */
         public static NameComponent[] getNameComponents( Name name, boolean isAgentSystem)
         {
                 NameComponent nc3 = new NameComponent( toString(name), "");

                 if ( isAgentSystem ) {
                         NameComponent nc0= new NameComponent( "AgentSystemService", "");
                         NameComponent nc1= new NameComponent(
                                                 String.valueOf((int)name.agent_system_type),
"");
                         NameComponent nc2= new NameComponent( new String(name.authority),
"");

                         NameComponent ncArr[]= {nc0, nc1, nc2, nc3};
                         return ncArr;
                 }
                 else {
                         NameComponent ncArr[]= {nc3};
                         return ncArr;
                 }
         }


         public static NameComponent[] getNameComponents4Global( AgentKind inAgentKind)
         {
                 NameComponent nc = new NameComponent( inAgentKind.packageName+ "."
                                         +inAgentKind.bodyName+inAgentKind.typeString, "");
                 NameComponent ncArr[]= {nc};
                 return ncArr;
         }
```

```
  /**
       *  String representation of the specified argument.
       *
       *  @param name Name
       *  @return    String representation of the specified argument
       */
       public static String toString( Name name)
       {
             String auth = new String( name.authority);
             String id   = new String( name.identity);
             String ast  = String.valueOf( (int)name.agent_system_type);
             return BEGIN_MARKER + id + PART_DELIM + auth + PART_DELIM + ast + END_MARKER;
       }


/**
     *  Create a NameComponent[] out of a <CODE>Name</CODE>.
     *  Take care about the order "authority, agent_systemType, identity".
     *  It's the right sequence for a search in the naming tree.
     *
     *  @param name          the Name
     *
     *  @return    NameComponent[]
     */
     public static NameComponent[] getNameComponents( Name name)
     {
      NameComponent nc0= new NameComponent( new String(name.authority), "");
      NameComponent nc1=
                    new NameComponent( String.valueOf((int)name.agent_system_type), "");
      NameComponent nc2= new NameComponent( new String(name.identity),"");

      NameComponent ncArr[]= {nc0, nc1, nc2};
      return ncArr;
     }


  /**
       *  Create a mafuri out of a <CODE>NameComponent[]</CODE>.
       *
       *  @param nc  NameComponent[]
       *
       *  @return    String mafuri
       */
       public static String name_to_mafuri( NameComponent[] nc)
       {
           String mafuri = "CosNaming:/";
           int i = 0;

           while ( i < nc.length) {

               mafuri = mafuri + nc[i].id + "!" + nc[i].kind + "/";
               i++;
           }
           return mafuri;
       }


  /**
       *  Create a mafuri out of a <CODE>NameComponent[]</CODE>, append a
       *  <CODE>NameComponent[]</CODE>.
       *
       *  @param nc       the NameComponent[]
       *  @param mafuri   mafuri to append a NameComponent[]
       *
       *  @return    String mafuri
       */
       public static String name_to_mafuri( NameComponent[] nc, String mafuri)
       {
           String _mafuri = mafuri;
           int i = 0;

           while ( i < nc.length) {

               _mafuri = _mafuri + nc[i].id + "!" + nc[i].kind + "/";
               i++;
           }
           return _mafuri;
       }
```

```
/**
     *  Create a <CODE>NameComponent[]</CODE> out of mafuri, isolate the IOR.
     *
     *  @param mafuri   mafuri
     *
     *  @return nc      NameComponent[]
     */
     public static  NameComponent[] mafuri_to_components( String mafuri)
         throws ArgumentInvalid
     {
         java.util.Vector nameComponents = new java.util.Vector(0,1);
         int i= 0;

         if ( mafuri.startsWith( "CosNaming")){
                 try {
                     mafuri = mafuri.substring( mafuri.indexOf( SLASH)+1);
                     Debug.logMessage( Debug.FACILITY_NAMEWRAPPER, Debug.LEVEL_DEBUG5,
                                       "NameWrapper, mafuri:" + mafuri);
                 }
                 catch( Exception e){// e.g. missing slash
                     mafuri = null;
                     throw new ArgumentInvalid();
                 }

             while ( mafuri.length() > 0){
                 if ( mafuri.startsWith( "IOR")) break;
                 if ( mafuri.indexOf( SLASH) < 0) throw new ArgumentInvalid();
                                                  // wrong structure of mafuri

                 NameComponent component = new NameComponent();

                 // PART_DELIM is missing or there is no kind-component
                 if ( mafuri.indexOf( PART_DELIM) >= 0 &&
                      mafuri.indexOf( PART_DELIM) < mafuri.indexOf( SLASH)) {
                     // for secure masa where authority is a certificate,
                     // can begin with '[' and can contain several '!'
                     if ( mafuri.startsWith( BEGIN_MARKER)) {
                         component.id = mafuri.substring( 0,
                                                  mafuri.indexOf(END_MARKER+1))
                                     + mafuri.substring( mafuri.indexOf( END_MARKER+1),
                                                 mafuri.indexOf( PART_DELIM));
                         Debug.logMessage( Debug.FACILITY_NAMEWRAPPER,
                           Debug.LEVEL_DEBUG5, "NameWrapper, id: " + component.id);
                         String tmpString = mafuri.substring(
                                                mafuri.indexOf(END_MARKER+1));
                         component.kind = tmpString.substring( tmpString.indexOf(
                                               PART_DELIM)+1, tmpString.indexOf( SLASH));
                         Debug.logMessage( Debug.FACILITY_NAMEWRAPPER,
                           Debug.LEVEL_DEBUG5, "NameWrapper, kind: " + component.kind);
                     }
                     else {
                        component.id = mafuri.substring( 0, mafuri.indexOf(PART_DELIM));
                        Debug.logMessage( Debug.FACILITY_NAMEWRAPPER,
                          Debug.LEVEL_DEBUG5, "NameWrapper, id: "
                                + mafuri.substring( 0, mafuri.indexOf( PART_DELIM)));
                         component.kind = mafuri.substring(
                                mafuri.indexOf( PART_DELIM)+1, mafuri.indexOf( SLASH));
                         Debug.logMessage( Debug.FACILITY_NAMEWRAPPER,
                           Debug.LEVEL_DEBUG5, "NameWrapper, kind: " + mafuri.substring(
                               mafuri.indexOf( PART_DELIM)+1, mafuri.indexOf( SLASH)));
                     }
                 }
                 else {
                   component.id   = mafuri.substring( 0, mafuri.indexOf( SLASH));
                   Debug.logMessage( Debug.FACILITY_NAMEWRAPPER, Debug.LEVEL_DEBUG5,
                    "NameWrapper, id: " + mafuri.substring( 0, mafuri.indexOf( SLASH)));
                   component.kind = "";
                   Debug.logMessage( Debug.FACILITY_NAMEWRAPPER, Debug.LEVEL_DEBUG5,
                                     "NameWrapper, kind :" + component.kind);
                 }

                 nameComponents.addElement( component);

                 i++;

                 try {
```

```
                    mafuri = mafuri.substring( mafuri.indexOf( SLASH)+1);
                    Debug.logMessage( Debug.FACILITY_NAMEWRAPPER, Debug.LEVEL_DEBUG5,
                                      "NameWrapper: " + mafuri);
                }
                catch( Exception e){
                       // wrong structure of mafuri or end of mafuri, e.g. no IOR added
                    mafuri = null;
                }
            }
        }
        else throw new ArgumentInvalid();

        int s = nameComponents.size();
        NameComponent[] nc = new NameComponent[s];

        nameComponents.copyInto( nc);

        // only for debugging
        for (int k=0; k< s; k++){
        Debug.logMessage( Debug.FACILITY_NAMEWRAPPER, Debug.LEVEL_DEBUG5, "NameWrapper,
                          comp: " + nc[k].id + " " +  nc[k].kind);
        }
        return nc;
    }


/**
     *  Get the IOR out of mafuri.
     *
     *  @param mafuri   mafuri
     *
     *  @return String  IOR
     */
    public static  String mafuri_to_IOR( String mafuri)
        throws ArgumentInvalid
    {
        try {
            mafuri = mafuri.substring( mafuri.indexOf( "IOR"));
        }
        catch( Exception e){
            Debug.logMessage( Debug.FACILITY_NAMEWRAPPER, Debug.LEVEL_DEBUG5,
                              "NameWrapper: No IOR added!", e);
            throw new ArgumentInvalid();
        }
        return mafuri;
    }


/**
     *  Compares two org.omg.CfMAF.Name.
     *
     *  @param name1     Name of first entity
     *  @param name2     Name of second entity
     *
     *@return boolean  true, if each id- and kind-component of the name-components match
     *
     *  @exception EntryNotFound There was no match of specified features
     *
     *  @see
     */
    public static boolean compare_names( Name name1, Name name2)
    {
        int comp;

        comp = ( new String (name1.identity)).compareTo( new String (name2.identity));
        if ( comp != 0) return false;

        if ( name1.agent_system_type != name2.agent_system_type) return false;

        comp = ( new String (name1.authority)).compareTo(
                                                    new String (name2.authority));
        if ( comp != 0) return false;

        return true;
    }
}
```

E) Literaturverzeichnis

[BGHO 99] Brandl A., Göbel M., Harlfinger C., von Oheimb D.: Skript zum JAVA-Kurs; Technische Universität München, Mai 1999

[Fla 98] Flanagan, D.: Java in a Nutshell; O'Reilly, 2. Auflage, 1998; Deutsche Ausgabe für Jaca 1.1

[GHR 99] Gruschke, B., Heilbronner, S., Reiser, H.: Mobile Agent System Architecture, Eine Plattform für flexibles IT-Management; Technischer Bericht 9902, Münchner Netzmanagement Team, Ludwigs-Maximilians-Universität München, München, 1999

[Heun 97] Heun, V.: Skriptum zur Vorlesung Grundlegende Algorithmen; Technische Universität München, Lehrstuhl für Effiziente Algorithmen, 22.10.97

[Kemp 98] Kempter, B.: Entwurf eines Java/CORBA-basierten Mobilen Agenten; Diplomarbeit, Technische Universität München, August 1998

[Kühn 97] Kühnel R.: Die Java 1.1 Fibel, 2. Auflage; Addison-Wesley, 1997

[OMG 99-10-07] The Common Object Request Broker: Architecture and Specification, Version 2.3.1; Object Management Group, October 1999; OMG 99-10-07

[OMG 98-10-11] CORBA Services: Common Object Services Specification, Interoperable Naming Service ; Object Management Group, 1998; OMG TC Document orbos/98-10-11

[OMG 98-03-09] CORBA Facilities: Mobile Agent System Interoperability Facilities, Submission; MASIF-RTF Results; Object Management Group, 1998; OMG orbos/98-03-09

[OMG 97-12-20] CORBA Services: Common Object Services Specification, Property Service; Object Management Group, 1997; OMG document 97-12-20

[OOC 99] ORBacus ™ for C++ and Java, Version 3.2.1; Object Oriented Concepts, Inc., 1999

[Roel 99] Rölle, H.: Authentisierung und Autorisierung für das Java/CORBA-Agentensystem MASA; Diplomarbeit, Technische Universität München, August 1999

[RRZN 98] Java, Begleitmaterial zu Vorlesungen/Kursen, 4. Auflage; Regionales Rechenzentrum für Niedersachsen / Universität Hannover, Dezember 1998

[RSC 97] Unified Modeling Language, Notation Guide, Version 1.0, 13.1.97; Rational Software Corporation

[Scho 99] Scholze, M.: Plattform(en) für Mobile Agenten in heterogenen, verteilten Systemen; Diplomarbeit, Technische Universität Berlin, Juni 1999

F) www-Links

- Homepage der Object Management Group:
  www.omg.org
- Homepage von Object Oriented Concepts, Inc.
  www.ooc.com
- SUN-Homepage (für Java API Online-Dokumentation)
  www.java.sun.com