INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Systementwicklungsprojekt

Entwicklung einer Benutzerschnittstelle mit Java/CORBA für die Mobile Agent System Architecture (MASA)

Bearbeiter: Stefan Gerber
Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Boris Gruschke
Helmut Reiser

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Im Rahmen der Diplomarbeit von Bernhard Kempter [Kemp 98] wurde eine Agentenarchitektur entworfen, die dem verteilten Management von komplexen, verteilten Systemen dient. Dabei können Agenten – autonom handelnde Softwaresysteme – bestimmte Verwaltungsaufgaben übernehmen, z.B. aus dem Netzwerkmanagementbereich. Ihre Laufzeitumgebung sind die sog. Agentensysteme, die Programmierschnittstellen anbieten, welche es erlauben, auf die Systeme selbst und die darauf laufenden Agenten Einfluß zu nehmen. So ist es beispielsweise möglich, Agenten zu starten oder zu beenden, laufende Agenten anzuhalten, wieder fortzusetzen usw. Überdies können Agenten in andere Agentensysteme migriert werden. Diese Funktionalität mit Hilfe einer grafischen Benutzerschnittstelle überschaubar und leicht bedienbar zu machen, ist die Aufgabe des Systementwicklungsprojekts.

1.2 Szenario

Die MASA definiert eine Architektur für Agenten und Agentensysteme.

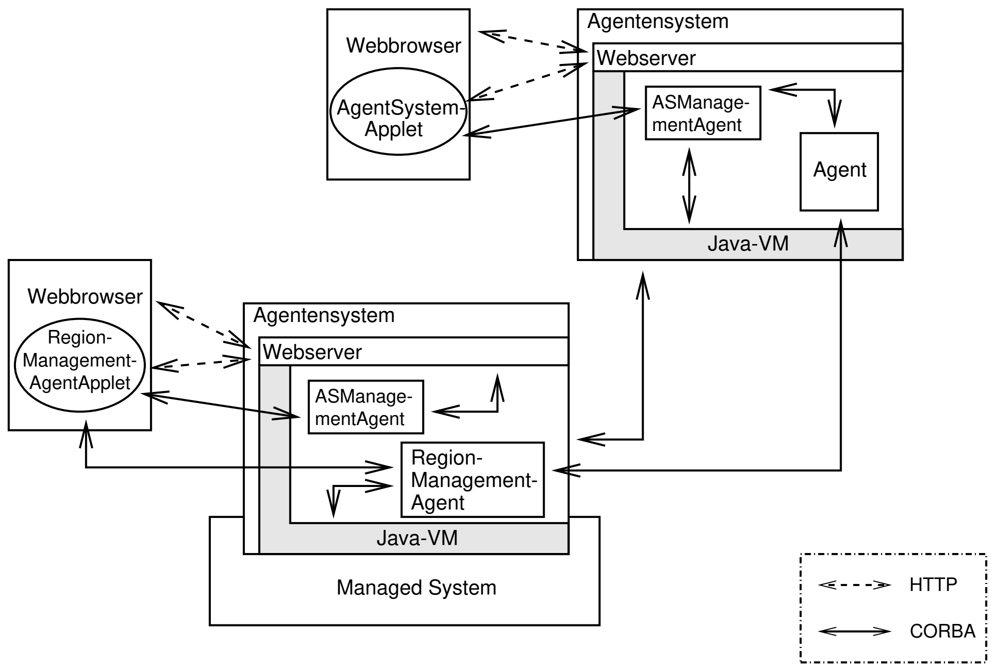

Abbildung 1.1: Die Architektur von MASA

Die Agentensysteme stellen die Laufzeitumgebungen der Agenten dar. In ihnen kann ein Webserveragent laufen, der es Java-fähigen Webbrowsern ermöglicht, die Applets der einzelnen Agenten zu laden. Die Agenten können untereinander kommunizieren (z.B. per IIOP) und, sofern sie mobil sind, in andere Agentensysteme migriert werden. Für eine ausführlichere Beschreibung der MASA siehe [Kemp 98].
Die Benutzeroberfläche, die im Rahmen dieses Systementwicklungsprojekts entwickelt wird, wird in Form zweier Java-Applets realisiert: Das sog. AgentSystemApplet, das jeweils einem Agentsystem fest zugeordnet ist und die darauf laufenden Agenten verwaltet, und das RegionManagementAgent-Applet, das einen Überblick über alle laufenden Agentensysteme und ihrer Agenten in einer MASA-Region bietet. Diesen Applets stehen zwei ebenfalls neu entwickelte Agenten zur Seite: Der ASManagementAgent und der RegionManagementAgent. Ersterer wird von beiden Applets benutzt und übernimmt dabei die Rolle eines Proxies für die CORBA-Dienste Naming Service und Event Service. Außerdem führt er die Introspektion in Agentenklassen durch, welche vom AgentSystemApplet benötigt wird. Der RegionManagementAgent sorgt als Proxy dafür, daß das ihm zugehörige RegionManagementAgent-Applet Zugriff auf den CORBA Naming Service und andere Agentensysteme als das eigene erhält.

Kapitel 2

Benutzeranleitung

2.1 Das AgentSystemApplet

Beim Start eines Agentensystems wird zusammen mit ihm u.a. auch ein Webseragent geladen. Hauptaufgabe dieses Agenten ist es, die Homepage des Agentensystems bereitzustellen. Bestandteil dieser Seite ist das *AgentSystemApplet*, das bei jedem Zugriff auf die Homepage eines Agentensystems mit einem Java-fähigen Webbrowser geladen wird und zur Verwaltung ausschließlich dieses einen Systems dient. Voraussetzung für den Betrieb des Applets ist ein laufender AS-ManagementAgent, ohne den das Applet nicht korrekt funktioniert. Das Vorhandensein dieses Agenten wird daher beim Start überprüft und gegebenenfalls eine Warnmeldung angezeigt.
Die Aufgaben des Applets sind in drei Bereiche gruppiert, die sich über Karteireiter im Applet auswählen lassen:

- Verwaltung bereits geladener Agenten (Karteireiter *Manage agents*)
- Das Starten von neuen Agenten (Karteireiter *Create agent*)
- Verwaltung des zum Applet gehörenden Agentensystems (Karteireiter *Agent System*)

Diese drei Bereiche wurden voneinander getrennt und können über Karteireiter im Applet ausgewählt werden.

2.1.1 Der Funktionsbereich „Verwaltung bereits geladener Agenten“

Eine der Hauptaufgaben des Applets ist das Management von Agenten, die bereits auf dem Agentensystem gestartet wurden. Dazu gehören:

- Auflisten der bereits geladenen Agenten und ihrer Statusinformationen
- Anhalten, Fortsetzen und Beenden von Agenten
- Transfer von Agenten auf andere Agentensysteme
- Verweis auf die Agenten-Webseiten mit den evtl. dazugehörigen Applets

- Das Schreiben in bzw. Lesen eines Agenten aus einer Datei

Die folgende Abbildung zeigt einen Screenshot des Panels, das die dafür notwendigen GUI-Elemente zusammenfaßt.

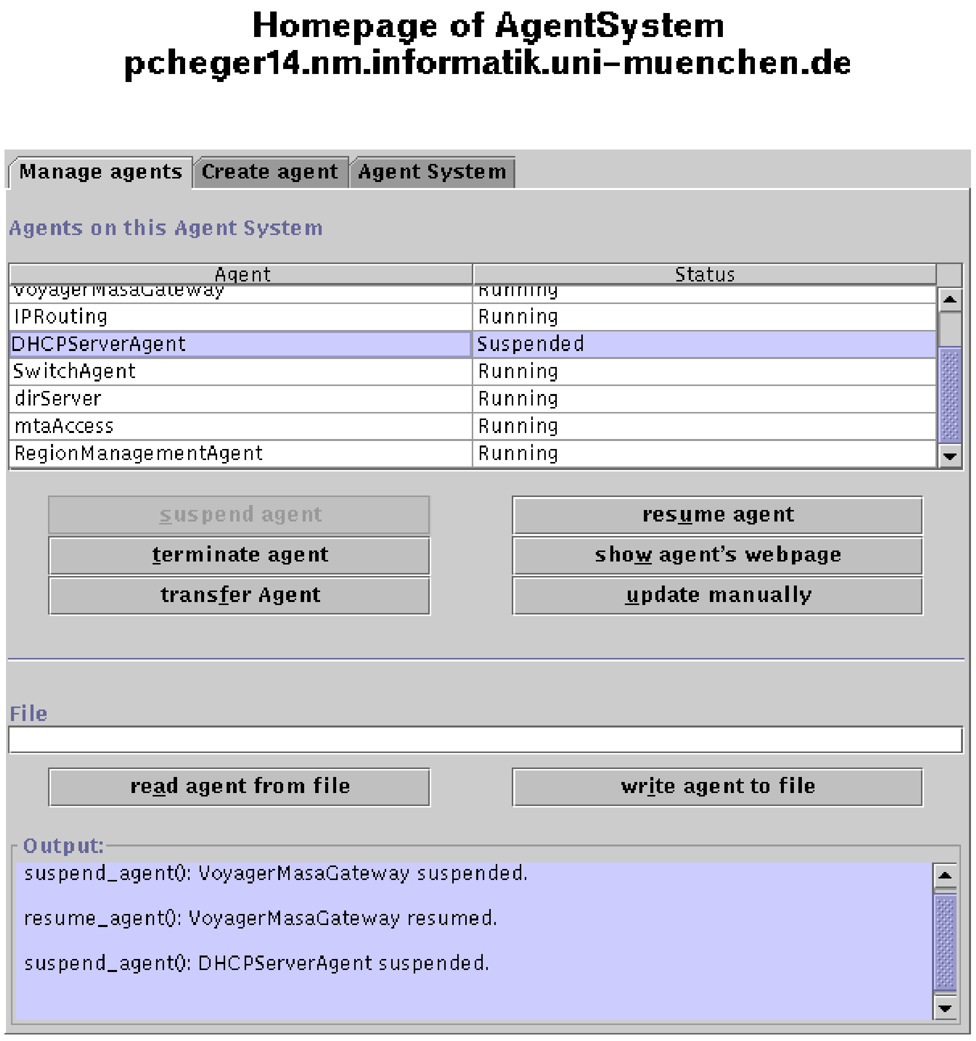

Abbildung 2.1: Erste Seite des AgentSystemApplets

Auflisten der bereits geladenen Agenten und ihrer Statusinformationen

Im Laufe der Zeit werden auf einem Agentensystem immer wieder Agenten gestartet, beendet, angehalten und fortgesetzt. Um den Überblick nicht zu verlieren, welche Agenten noch geladen sind und ob sie sich gerade in laufendem bzw. angehaltenem Zustand befinden, verfügt das Applet über eine zweispaltige Tabelle, die diese Informationen darstellt. Die erste Spalte enthält die Namen aller geladenen Agenten, die zweite Spalte deren jeweilige Statusinformationen. Der Status eines Agenten kann zwei Zuständen entsprechen:

- „running“
- „suspended“

Wurde ein Agent angehalten (siehe unten), so ist er zwar weiterhin geladen, jedoch nicht mehr aktiv. Er erhält deswegen das Attribut „suspended“. Wird er fortgesetzt, so befindet er sich wieder im ursprünglichen Zustand „running“.
Diese Tabelle dient nicht nur der Auflistung aller geladenen Agenten des Agentensystems, sondern auch der Auswahl eines Agenten zur Manipulation (anhalten, beenden, migrieren etc.). Der Benutzer muß dazu lediglich einen Eintrag der Tabelle anklicken. Sobald eine Auswahl getroffen wurde, werden sämtliche Buttons aktiviert, die bis dahin ausgeblendet waren, da sie einen ausgewählten Agenten voraussetzen. Wurde eine Aktion ausgeführt, werden die Buttons wieder deaktiviert, um unbeabsichtigte Manipulation zu vermeiden.
Die Agentenliste wird automatisch auf den aktuellen Stand gebracht, sei es, daß Agenten gestartet, beendet oder anderweitig manipuliert wurden. Falls dieser Mechanismus jedoch einmal versagen sollte – z.B. weil der dazu nötige CORBA Event Service nicht zur Verfügung steht oder ein Objekt erzeugt wurde, das bei seiner Erstellung kein CORBA Ereignis auslöst (näheres siehe Abschnitt 2.2.1 *Anzeigen aller aktiven Agentensysteme*) – kann der Button *update manually* zur manuellen Aktualisierung der Tabelle benutzt werden.

Anhalten, fortsetzen und beenden von Agenten

Alle drei Aktionen werden mit Hilfe dafür vorgesehener Buttons durchgeführt. Nachdem der Benutzer aus der Tabelle den Agenten ausgewählt hat, auf den sich die nachfolgende Aktion beziehen soll, kann er einen der entsprechenden Buttons betätigen.
Wird der Button *suspend agent* gedrückt, während ein laufender Agent (Status „running“) ausgewählt ist, so stellt dieser sämtliche Tätigkeiten ein. Darüber hinaus ändert sich automatisch sein Status in der Tabelle von „running“ nach „suspended“.
Der Vorgang läßt sich rückgängig machen, indem der angehaltene Agent ausgewählt und anschließend der Button *resume agent* betätigt wird. Wieder wird die Anzeige in der Tabelle auf den aktuellen Stand gebracht.
Ein Agent kann auch gezwungen werden, alle Aktivitäten abzubrechen und sich aus dem Speicher zu entfernen. Dazu muß bei ausgewähltem Agent der Button *terminate agent* gedrückt werden. Der Agent beendet sich daraufhin und sein Eintrag wird aus der Tabelle mit laufenden Agenten entfernt.
Ausnahmen bilden der Webserveragent und der ASManagementAgent. Diese sollten auf keinen Fall beendet oder angehalten werden, da sie für den Betrieb des Applets notwendig sind. Der Benutzer wird daher vorher explizit zur Bestätigung der Aktion aufgefordert.

Transfer von Agenten auf andere Agentensysteme

Das Verschieben von Agenten von einem Agentensystem in ein anderes ist eine Funktion, die mit beiden in dieser Dokumentation beschriebenen Applets möglich ist. Um dies zu bewerkstelligen, muß der entsprechende Agent aus der Agententabelle ausgewählt werden und anschließend der Button *transfer Agent* gedrückt werden. Dieser Button ist nur aktiviert, falls es sich bei dem ausgewählten Agenten um einer mobilen Agenten handelt. Wurde der Button gedrückt erscheint ein Dialogfenster, das eine Auswahlliste aller momentan verfügbaren Agentensysteme bietet, in die der Agent verschoben werden kann.

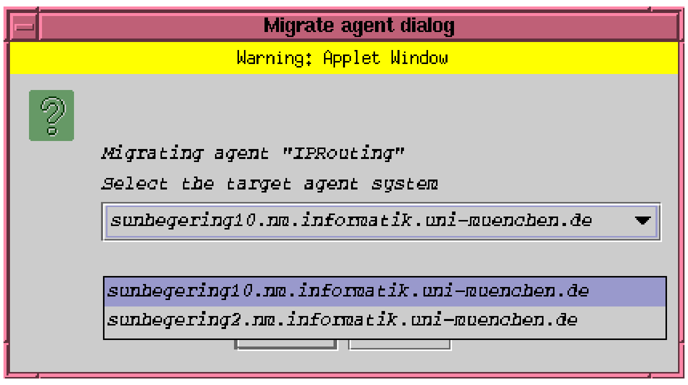

Abbildung 2.2: Auswahl des Migrationsziels

Aus der Liste wird nun das Ziel-Agentensystem ausgewählt und *OK* gedrückt. Der Agent wird in das ausgewählte Agentensystem verschoben und sein Eintrag aus der Agentenliste automatisch gelöscht. Von anderen Agentensystemen eingehende Agenten werden ebenfalls automatisch in der Agententabelle aufgeführt.

Anzeigen der Agenten-Webseiten mit den evtl. dazugehörigen Applets

Die meisten Agenten besitzen eigene Homepages mit Applets, die sich mit deren individuellen Aufgaben befassen. Das AgentSystemApplet bietet zwei Möglichkeiten, diese Seiten anzuzeigen.
Die erste Alternative besteht darin, den Agenten, dessen Homepage geladen werden soll, aus der Agententabelle auszuwählen und anschließend den Button *Show agent's webpage* zu betätigen. Es wird daraufhin ein weiteres Browserfenster geöffnet, in dem die betreffende Webseite angezeigt wird.
Das gleiche erreicht der Benutzer, indem er den betreffenden Agenten in der Tabelle mit einem Doppelklick auswählt, was meist die schnellere Methode darstellt.

Das Schreiben in bzw. Lesen eines Agenten aus einer Datei

Manchmal kann es sinnvoll sein, den Zustand eines Agenten in einer Datei zu speichern. Zu diesem Zweck ist im Applet ein Eingabefeld vorgesehen, in das der Benutzer Namen und Pfad einer Datei eingeben kann.
Der aus der Tabelle ausgewählte Agent wird auf Betätigen des Buttons *write agent to file* in die angegebene Datei geschrieben und aus der Agententabelle entfernt.
Ein derart abgespeicherter Agent kann auf ähnliche Weise wieder geladen werden. Die Datei muß mit ihrem Pfad wieder in dem Eingabefeld angegeben werden, woraufhin die Betätigung des Buttons *read agent from file* das Laden und aktivieren des Agenten zur Folge hat.

2.1.2 Der Funktionsbereich „Das Starten von neuen Agenten“

Die zweite Seite des Applets beschäftigt sich mit dem Laden eines Agenten in das Agentensystem.

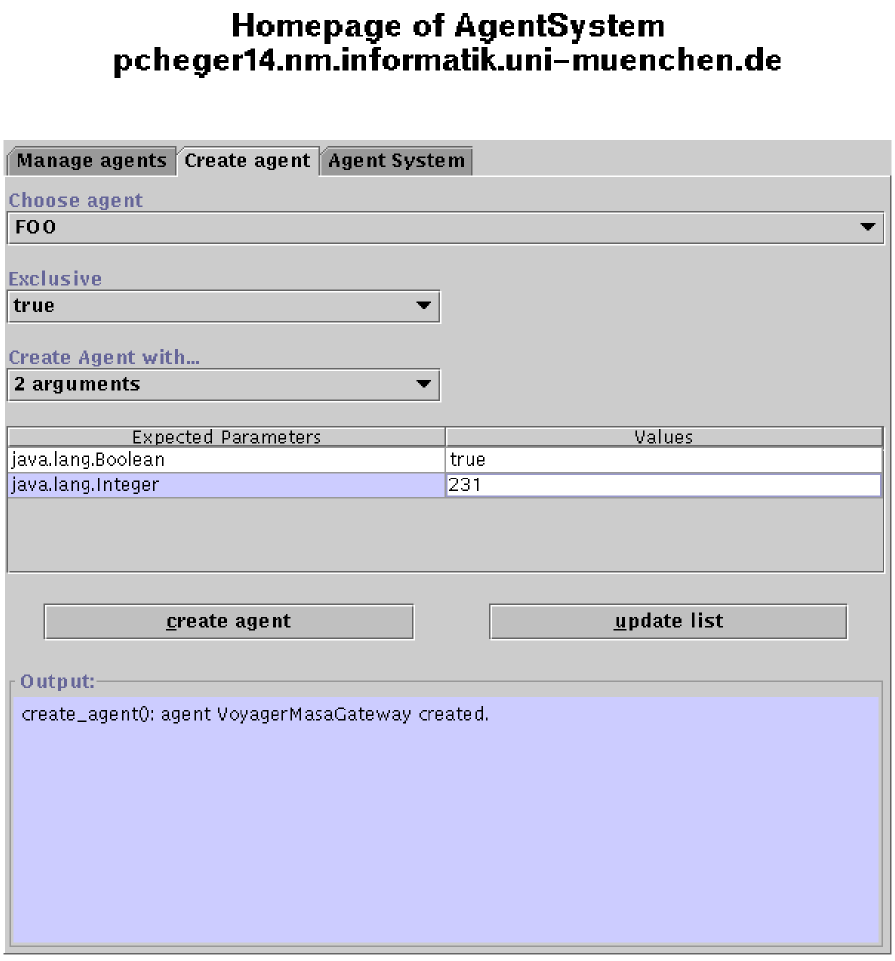

Abbildung 2.3: „Create Agent“-Seite des AgentSystemApplets

Verschiedene Vorarbeiten müssen vorgenommen werden, bevor ein Agent geladen werden kann:

- **Auswahl des Agenten**
  Am Kopf der Seite befindet sich eine mit „Choose Agent“ beschriftete Auswahlbox. Sie beinhaltet eine Liste von mit dem Applet registrierten Agenten, zwischen denen der Benutzer wählen kann.
  Die Registrierung eines Agenten mit dem Applet und damit sein Erscheinen in der Auswahlbox, geschieht über die Datei *ImplementedAgents.txt*, die sich normalerweise im MASA-Verzeichnis befindet, aber, je nach Einstellung durch die Konfigurationsdatei *masa.properties*, auch an einer anderen Stelle im Dateisystem liegen kann. Sie enthält eine Aufzählung von bereits implementierten Agenten, die über das AgentSystemApplet geladen werden können. Falls *ImplementedAgents.txt* während der Arbeit mit dem Applet verändert wird, so kann sie mit dem Button *update list* neu eingelesen werden. Die Liste der Agenten wird daraufhin an die neuen Da-

ten angepaßt.
Sobald der Benutzer einen Agenten ausgewählt hat, wird dessen Klassendatei untersucht, um auf die von ihm erwarteten Parameter zu schließen. Falls diese Klasse vom Programm nicht gefunden werden kann, so wird eine Warnmeldung angezeigt. In diesem Fall sollte kontrolliert werden, ob der Agent tatsächlich installiert wurde und sich an der Stelle im Verzeichnisbaum befindet, die durch *ImplementedAgents.txt* spezifiziert wird.
Nähere Informationen zu *ImplementedAgents.txt* siehe Abschnitt *3.1 Erstellung der Übersicht ladbarer Agenten*.

- **Exklusiver Agent?**
  Unter der Auswahlbox für den Agenten befindet sich eine weitere mit der Bezeichnung „Exclusive". Sie enthält lediglich zwei Einträge, „true" und „false". Mit dieser Einstellung wird festgelegt, ob der Agent als sogenannter „exklusiver Agent" gestartet werden soll. Dabei handelt es sich um Agenten, die in einer MASA-Region nur einmal gestartet werden dürfen, da sie z.B. eine Ressource exklusiv beanspruchen müssen.

- **Mit welchen Parametern soll der Agent gestartet werden?**
  Die nächste Auswahlbox mit der Bezeichnung „Create Agent with..." enthält, je nach Auswahl eines Agenten aus „Choose agent", eine Liste von verschiedenen Möglichkeiten den Agenten zu starten. Jeder Eintrag gibt dabei eine Anzahl von Parametern an, mit denen ein Agent geladen werden kann. Diese Liste ergibt sich durch Introspektion der Klasse dieses Agenten.
  Die Auswahlbox ist gekoppelt mit dem nächsten Element des Applets, einer Tabelle mit der Beschriftung „Parameters for this constructor". In dieser Tabelle werden Zeile für Zeile die Datentypen der Parameter für den gewählten Konstruktor aufgelistet. Besitzt ein Agent z.B. einen Konstruktor, der zwei String-Parameter erwartet, und wird dieser Konstruktor über die Auswahlbox „Create Agent with..." ausgesucht, so wird die Tabelle etwa wie in obiger Abbildung aussehen. Die zweite Spalte dient dem Benutzer dazu, Werte für die jeweils links stehenden Parameter einzugeben. Sie werden standardmäßig vom Applet mit „none" vorbelegt.
  Da die eingegebenen Werte der Tabelle nur in Form eines Strings entnommen werden können, müssen sie vom Applet explizit in die entsprechenden Datentypen der Parameter umgewandelt werden. Daher können leider nicht alle Parametertypen in einem Agentenkonstruktor akzeptiert werden. Zum Zeitpunkt der Entstehung dieser Dokumentation beschränken sich die Möglichkeiten auf *java.lang.String*, *java.lang.Integer*, *java.lang.Float* und *java.lang.Double*. Falls in einem Konstruktor ein ungültiger Parameter-Datentyp gefunden wird, so wird dies mit einer Warnmeldung angezeigt.

Sobald ein Agent ausgewählt wurde, ferner entschieden wurde, ob er exklusiv sein soll und seine Parameter festgelegt wurden, kann er gestartet werden. Dies geschieht durch Drücken des Buttons *create agent*. Der Agent wird daraufhin im Agentensystem gestartet und erscheint in der Agententabelle auf der ersten Seite des Applets. Etwaige Fehler- bzw. Erfolgsmeldungen werden im blau hinterlegten Ausgabebereich angezeigt.

2.1.3 Der Funktionsbereich „Verwaltung des zum Applet gehörenden Agentensystems“

Die dritte Seite des AgentSystemApplets dient der Verwaltung des Agentensystems, dem das Applet zugeordnet ist.

Abbildung 2.4: Dritte Seite des AgentSystemApplets

Der größte Teil dieser Seite besteht aus einem Ausgabefeld. Dort werden Informationen über das Agentensystem angezeigt, sobald der Benutzer den Button *get agent system info* betätigt. Diese Informationen umfassen:

- Agent System Type
- Authority

- Identity
- Description

Das Applet bietet in diesem Kontext überdies noch die Möglichkeit, das dazugehörige Agentensystem zu beenden. Sobald der Benutzer den Button *terminate agent system* betätigt hat, wird – nach Bestätigung – das Agentensystem mit allen darauf laufenden Agenten terminiert. Das Applet reagiert daraufhin nicht mehr auf Benutzereingaben und muß beendet werden.

2.2 Das RegionManagementAgent-Applet

Im Rahmen dieses Systementwicklungsprojekts entstand u.a. der RegionManagementAgent (siehe 4.2 *Der RegionManagementAgent*), dessen Aufgabe die Bereitstellung von Proxyfunktionalität zum Zugriff auf CORBA Objekte fremder Rechner für das nachfolgend beschriebene RegionManagementAgent-Applet ist. Dieses Applet ist dem RegionManagementAgent zugeordnet und kann z.B. vom AgentSystemApplet aus aufgerufen werden.

Abbildung 2.5: Das RegionManagementAgent-Applet mit geöffnetem Agenten-Kontextmenü

Die Aufgaben des Applets umfassen:

- Anzeigen aller aktiven Agentensysteme
- Anzeigen der Agenten eines beliebigen Agentensystems

- Verschieben eines Agenten in ein anderes Agentensystem
- Anzeige der Webseiten von Agenten oder Agentensystemen
- Beenden von Agentensystemen

2.2.1 Anzeigen aller aktiven Agentensysteme

Die linke Hälfte des Applets zeigt einen Baum. In diesem Baum mit der Bezeichnung „Agent Systems & Co“ werden die gerade aktiven Agentensysteme namentlich aufgeführt. Die Wurzel wird von einem Knoten namens „Agent System Context“ gebildet in Anlehnung an den CORBA Namensbereich, dem die Namen der Agentensysteme entnommen werden. Die Blätter des Baums tragen die Namen der Agentensysteme, die gerade im Netzwerk aktiv sind, mit einer Ausnahme: „global“. Dabei handelt es sich nicht um ein besonderes Agentensystem, sondern um eine Gruppe, in der alle „exklusiven“ Agenten eingetragen sind, die also mit der Option „exclusive = true“ im AgentSystemApplet gestartet wurden. Der Eintrag eines Agenten in „global“ erfolgt zusätzlich zum Eintrag in das Agentensystem, in dem der Agent tatsächlich läuft.
Die Auflistung der Agentensysteme erfolgt sofort beim Start des Applets. Falls Agentensysteme nach Aufruf des Applets gestartet oder beendet werden, wird der Baum automatisch entsprechend ergänzt oder berichtigt. Falls dieser Mechanismus jedoch einmal versagen sollte, kann der Benutzer mit dem unter dem Baum befindlichen Button mit der Bezeichnung *update manually* eine Aktualisierung erzwingen. Dies kann z.B. dann nötig sein, wenn in den entsprechenden CORBA Namensbereich ein Element eingetragen wurde, das kein MASA-Element ist und daher kein CORBA Ereignis erzeugt, welches eine automatische Aktualisierung auslösen würde. Ein Beispiel dafür ist der Namensbereich „voyager“, der eine Liste der geladenen Agenten des Voyager-Systems beeinhaltet (siehe dazu [Bran 99]).

2.2.2 Anzeigen der Agenten eines beliebigen Agentensystems

Zu jedem Agentensystem, das in dem Baum auf der linken Seite aufgeführt ist, können in der rechten Hälfte des Applets die darin geladenen Agenten angezeigt werden. Sobald der Benutzer ein Agentensystem des Baums markiert, wird auf der rechten Seite des Applets eine Tabelle aufgebaut, die alle Agenten dieses Agentensystems enthält.
Diese Liste wird automatisch auf den neuesten Stand gebracht, wenn im betroffenen Agentensystem weitere Agenten gestartet oder bereits geladene beendet werden. Die Aktualisierung kann aber auch aus obigen Gründen mit Hilfe des Buttons *update manually* erzwungen werden.

2.2.3 Verschieben eines Agenten in ein anderes Agentensystem

Klickt der Benutzer mit der rechten Maustaste auf einen Agenten der Agententabelle, so wird ein Kontextmenü geöffnet, das verschiedene Funktionen bietet. In diesem Menü befindet sich ein Eintrag „Migrate agent“, welcher nur aktiviert ist, falls es sich bei dem ausgewählten Agenten um einen mobilen Agenten handelt. Wird dieser Eintrag angeklickt, so öffnet sich ein Dialogfenster, in dem der Benutzer aus einer Liste der aktiven Agentensysteme das Agentensystem auswählen kann, in das der betreffende Agent verschoben werden soll. Es ist das gleiche Dialogfenster, das bereits im AgentSystemApplet zum Einsatz kam. Sobald das Zielsystem gewählt und *OK* gedrückt wurde, wird der Agent in das Agentensystem transferiert. Dabei wird er automatisch aus der gerade angezeigten Agententabelle entfernt. Wenn es sich bei dem zu verschiebenden Agenten um einen RegionManagementAgenten handelt, so wird der Benutzer vor Durchführung der Migration davor gewarnt, daß das Applet evtl. nicht mehr richtig funktionieren wird, falls es diesem Agenten zugeordnet ist. Dies hat seine Begründung in der funktionalen Abhängigkeit des Applets von seinem Agenten.

2.2.4 Anzeige der Webseiten von Agenten oder Agentensystemen

Im Kontextmenü eines Agenten findet sich auch der Eintrag *Show webpage*, der dafür sorgt, daß ein weiteres Browserfenster geöffnet wird. In diesem wird die Webseite des Agenten mit seinem Applet angezeigt.
Ebenso wie die Agenten in der Tabelle verfügen auch die Agentensysteme im Baum über ein eigenes Kontextmenü. Dieses enthält ebenfalls den Eintrag *Show webpage*, der ein Browserfenster mit der Homepage des Agentensystems und dem dazugehörigen AgentSystemApplet öffnet.
In beiden Fällen kann die entsprechende Webpage auch durch einen Doppelklick auf den Agenten bzw. das Agentensystem abgerufen werden.

2.2.5 Beenden von Agentensystemen

Der Menüpunkt *terminate agent system* im Kontextmenü eines Agentensystems sorgt dafür, daß das betreffende Agentensystem – nach einer Sicherheitsabfrage – heruntergefahren wird. Dabei werden auch sämtliche darin enthaltenen Agenten beendet. Der Baum, der die Agentensysteme auflistet, wird automatisch aktualisiert.

Kapitel 3

Realisierung

3.1 Erstellung der Übersicht ladbarer Agenten

Soll ein Agent über das AgentSystemApplet gestartet werden, so sollte dem Benutzer eine Übersicht über das Angebot der möglichen Agenten präsentiert werden. Dies geschieht, wie bereits beschrieben, über eine dafür vorgesehene Auswahlbox, die die Namen der bereits implementierten Agenten enthält. Woher jedoch stammen die Daten, die dieser Box zu Grunde liegen?
Die Lösung ist eine spezielle Registrierungsdatei namens *ImplementedAgents.txt*. Sie wird – auf Anfrage – vom Webserveragenten gelesen und ihr Inhalt dem anfragenden Applet zur Verfügung gestellt.

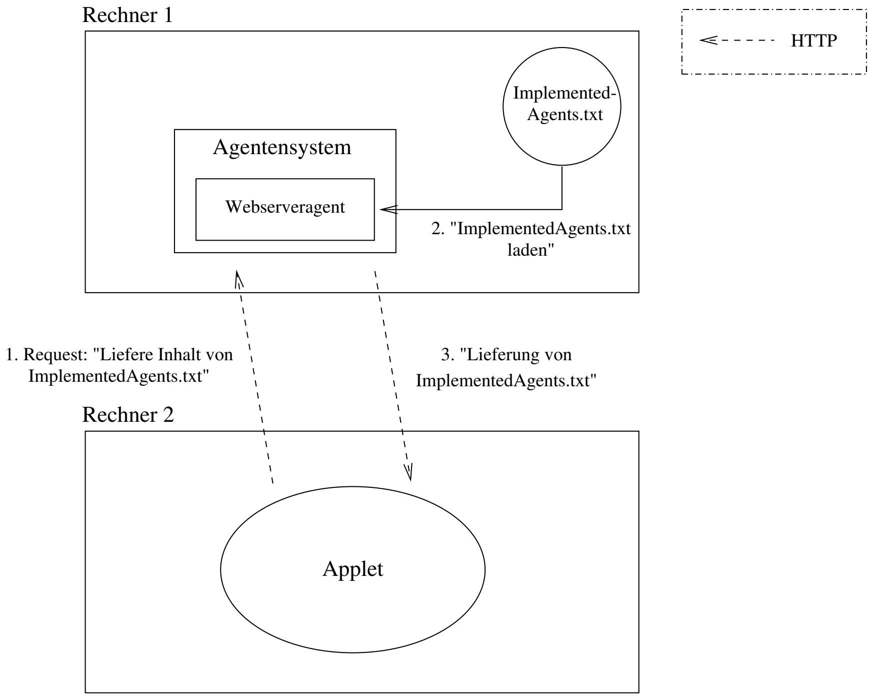

Abbildung 3.1: Zugriff auf die Datei „ImplementedAgents.txt“

Die Datei wird mit Hilfe des Eintrags „de.unimuenchen.informatik.mnm.masa.implagents“ in der Konfigurationsdatei *masa.properties* lokalisiert. *ImplementedAgents.txt* enthält eine Aufzählung von bereits implementierten Agenten, die über das AgentSystemApplet geladen werden können. Das nachfolgende Beispiel einer solchen Datei zeigt die Syntax, die ihr zu Grunde liegt:

```
FOO de.unimuenchen.informatik.mnm.masa.agent.foo MobileAgent
IPRouting de.unimuenchen.informatik.mnm.masa.agent.iprouting MobileAgent
```

Jede Zeile repräsentiert einen Agenten. Die erste Spalte enthält seinen Namen, die zweite das Java-Package, in dem er enthalten ist und die dritte Spalte gibt an, ob es sich um einen mobilen (MobileAgent), oder um einen stationären Agenten (StationaryAgent) handelt. Diese Informationen werden vom Applet benötigt, um einen Agenten in ein Agentensystem laden zu können. Falls ein neuer Agent entwickelt wird, so muß er in obiger Form in *ImplementedAgents.txt* eingetragen werden, bevor er durch das Applet gestartet werden kann.

3.2 Parameter zum Start eines Agenten

Jeder Agent besitzt einen oder mehrere Konstruktoren, von denen jeder eine unterschiedliche Anzahl von Parametern besitzen kann, die darüber hinaus verschiedenen Typs sein können. Es muß also dem Benutzer des AgentSystemApplets eine Möglichkeit geboten werden, zwischen den verschiedenen Konstruktoren eines Agenten zu wählen. Überdies müssen die Datentypen der Parameter ersichtlich sein, damit der Benutzer die Werte entsprechend eingeben kann.
Das AgentSystemApplet bietet dazu eine Kombination aus einer Auswahlbox, die für einen Agenten die möglichen Konstruktoren anhand der Parameteranzahl zur Auswahl stellt, und einer zweispaltigen Tabelle, die links die Datentypen aller Parameter des gewählten Konstruktors enthält und rechts entsprechend die Eingabe der Werte ermöglicht.
Sobald im AgentSystemApplet ein Agent ausgewählt wurde, der gestartet werden soll, ruft das Applet die Methode *get_agent_parameters(...)* des ASManagementAgenten auf. Diese erstellt mittels Introspektion in die Agentenklasse über die Java *Reflection-API* eine Liste aller Parametertypen sämtlicher Konstruktoren. Diese Agentenklasse kann sich auch in einem entfernten Code-Repository befinden.

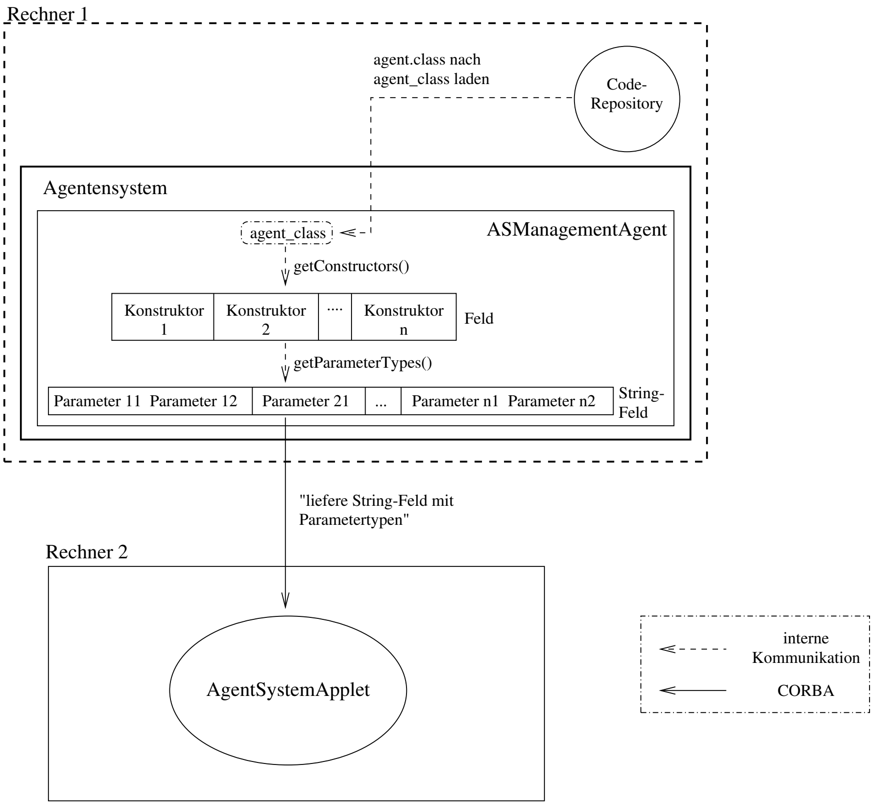

Abbildung 3.2: Ermittlung der Datentypen der Parameter

Wie die Abbildung zeigt, werden die Parametertypen eines jeden Konstruktors zusammen gruppiert. Ein String-Feld, in dem jedes Element die Parametertypen eines bestimmten Konstruktors enthält, wird an das aufrufende Applet zurückgeliefert und dort weiter verwertet. Durch die Gruppierung der Parameterlisten in verschiedenen Feldelementen, ist es im Applet möglich, für jeden Konstruktor die dazugehörige Liste von Parametern bereitzustellen. Daher kann das Programm auch die Anzahl der Parameter pro Konstruktor ermitteln, welche als Basis für die Auswahl eines Konstruktors durch den Benutzer mit Hilfe der dafür vorgesehenen Auswahlbox dient. Sobald der Benutzer den Konstruktor mit der gewünschten Anzahl Parameter ausgewählt hat, werden die Parametertypen, die zu diesem Konstruktor gehören, in der Tabelle angezeigt.
Da die Festlegung der Parameterwerte über die Oberfläche durch Eingabe in die rechte Spalte der Parametertabelle geschieht, ist die Auswahl an erlaubten Parameterdatentypen eines Agentenkonstruktors eingeschränkt auf solche, welche sich mittels eines Strings in ihrem Wert festlegen lassen. Außerdem verbietet sich der Einsatz von zusammengesetzten Datentypen und elementaren Datentypen (wie *int*, *float* etc.), da die Methode *AgentManager.create_ agent(...)*, zuständig für den Start eines Agenten im Agentensystem, die übergebenen Parameter in einem *java.util.Vector*-Objekt gruppiert, was mit elementaren Datentypen jedoch erst nach einer expliziten Umwandlung in die entsprechenden elementaren

Objekttypen (Integer, Float, etc.) möglich wäre. Diese fehlt jedoch bislang im AgentManager. Da überdies die Praxis zeigt, daß neben *Strings* und *Booleans* lediglich ganze bzw. Gleitkommazahlen als Parameterdatentypen zum Einsatz kommen, wird die Auswahl an akzeptierten Parameterdatentypen auf folgende eingeschränkt:

- *java.lang.String*
- *java.lang.Boolean*
- *java.lang.Integer*
- *java.lang.Float*
- *java.lang.Double*

Alle anderen Datentypen werden durch das Applet abgelehnt, was dem Benutzer durch eine Meldung mitgeteilt wird. Der Autor des Agenten muß in einem solchen Fall den Konstruktor entsprechend anpassen.
Hat der Benutzer die Werte der Parameter in die Tabelle eingegeben und drückt den *create agent* Button, so wird für jeden der Parameter eine Instanz des entsprechenden Datentypen mit dem eingegebenen String initialisiert und an *AgentManager.create_ agent(...)* übergeben.

3.3 Automatische Benachrichtigung der Applets bei Veränderungen des Agentensystemzustands

Um die in den Applets angezeigten Daten aktuell zu halten, beispielsweise wenn neue Agenten geladen/beendet, weitere Agentensysteme gestartet oder beendet werden etc., war es nötig, eine Möglichkeit zu finden, die Applets über diese Ereignisse „zu informieren“. Die Implementierung der MASA sieht bereits vor, daß beim Starten bzw. Beenden von Agenten und Agentensystemen entsprechende Ereignisse über einen CORBA Event Channel ausgelöst werden. Zusammen mit diesen Ereignissen werden auch Daten versendet, welche die genaue Art des Ereignisses spezifizieren. Auf diese Weise können Applets, die diese Ereignisse registrieren, entsprechend reagieren, also etwa eine Liste mit laufenden Agenten aktualisieren, falls ein weiterer Agent im betreffenden Agentensystem geladen wird, etc.
Um diese Ereignisse an Applets weiterzureichen, müßten diese direkten Zugang zum CORBA Event Service besitzen. Problematisch ist dabei jedoch, daß Applets einem strengen Sicherheitsmodell unterliegen, das es verbietet, auf Dienste zuzugreifen, die nicht auf dem gleichen Rechner im Netz laufen wie der Webserver, über den das Applet geladen wurde. Daher muß ein Weg gefunden werden, wie die Applets trotzdem benachrichtigt werden können.
Die Lösung besteht darin, einen Proxy-Agenten zu definieren, der direkten Zugang zum CORBA Event Channel besitzt. Die Applets müssen mit diesem kommunizieren können, ohne die an sie gestellten Sicherheitsanforderungen zu verletzen.

Die erste Forderung kann erfüllt werden, da ein MASA-Agent nicht den strengen Sicherheitsanforderungen unterliegt, die an Applets gestellt werden. Er ist weitgehend frei in seinen Kommunikationsmöglichkeiten. Der zweiten Bedingung wird entsprochen, indem der Agent in jedem Agentensystem gestartet wird, für das ein solches Applet geladen werden soll. Dadurch befindet er sich auf der gleichen Maschine wie der Webserveragent, über den das Applet geladen wird, welches auf diese Weise ungehindert mit dem Proxy-Agenten kommunizieren kann.
Der Agent, der diese Proxy-Funktionalität übernimmt, ist der ASManagement-Agent. Durch „Horchen“ am CORBA Event Channel registriert er eingehende Ereignisse und gibt sie an Applets weiter. Das allgemeine Schema, nach dem der Agent diese Ereignisse empfängt, ist das sogenannte *Push Modell*.

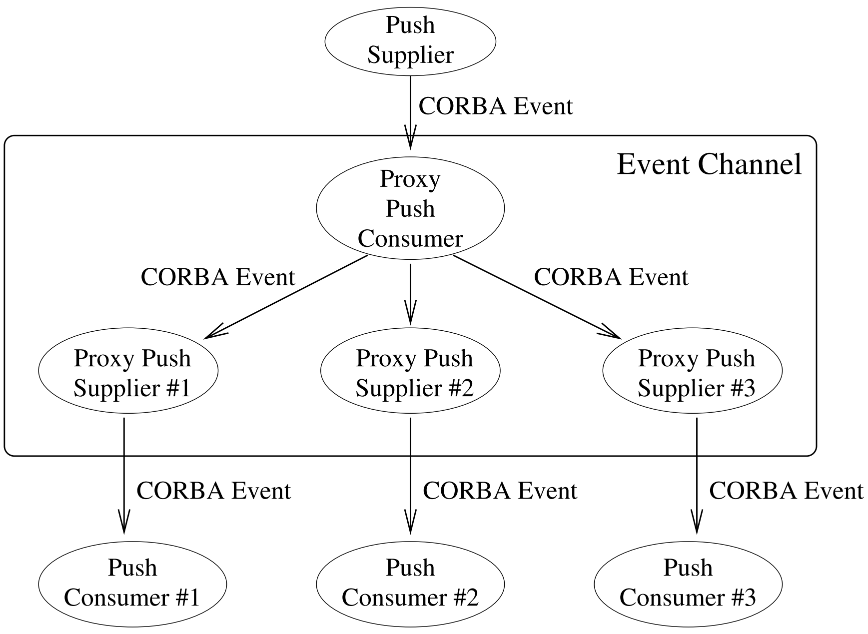

Abbildung 3.3: Allgemeines Push-Modell

Dabei sendet der *PushSupplier* (hier das Agentensystem) Daten an seinen *ProxyPushConsumer*. Auf der anderen Seite verbringt der *PushConsumer*, der vom ASManagementAgent erzeugt und bei einem *ProxyPushSupplier* registriert wird, die meiste Zeit in einer Event-Schleife und wartet darauf, daß Daten von dem *ProxyPushSupplier* ankommen, bei dem er als Empfänger registriert ist. Der Event Channel sorgt für die Übertragung von Daten vom *ProxyPushConsumer* zum *ProxyPushSupplier*.
Damit die Ereignisse und die damit verbundenen Daten auch zu den Applets gelangen, definiert jedes Applet eine eigene *PushConsumer* Klasse. Eine Instanz dieser Klasse wird beim Start des Applets erzeugt, beim CORBA ORB registriert und dem ASManagementAgenten mit dessen Methode *connect_push_consumer(...)* übergeben. Dieser erhält dadurch CORBA-

Referenzen der *PushConsumer* der Applets. Jeder *PushConsumer* besitzt definitionsgemäß die Methode *push(...)*, die bei eingehenden Events automatisch aufgerufen wird. Sie definiert daher das Verhalten, mit dem auf Ereignisse reagiert wird.
Der ASManagementAgent hält die übergebenen Objektreferenzen in einer eigens dafür vorgesehenen Datenstruktur. Wird nun von seinem eigenen *PushConsumer* ein Ereignis vom CORBA Event Service empfangen, so ruft dessen *push(...)*-Methode wiederum die *push(...)*-Methoden aller *PushConsumer*-Objektreferenzen auf, die von Applets an den Agenten übergeben wurden, und übermittelt damit die CORBA Events an die *PushConsumer* der Applets. Da die Applets ihre eigenen *PushConsumer*-Klassen und die damit verbundenen *push(...)*-Methoden definieren, kann jedes Applet individuell auf die Events reagieren, die ihnen vom ASManagementAgenten übergeben werden. Die nachstehende Abbildung erläutert das eben beschriebene nocheinmal.

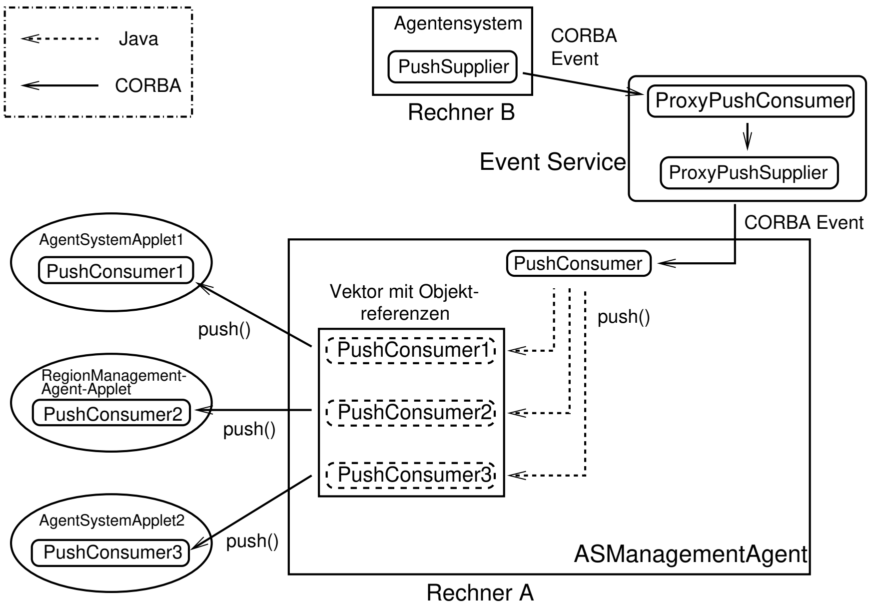

Abbildung 3.4: Weiterleitung von CORBA Events durch den ASManagement-Agenten

Auf diese Weise werden alle beim Agenten eingehenden CORBA Ereignisse über CORBA-Kommunikationsmechanismen an die Applets weitergeleitet. Die damit verbundenen Daten werden dabei ebenfalls weitergegeben. Sie werden von den *push(...)*-Methoden der Applet-*PushConsumer* ausgewertet, um auf die Ereignisse der Situation entsprechend zu reagieren.

3.4 Zugriff auf den CORBA Naming Service

Beide in dieser Dokumentation beschriebenen Applets benötigen den CORBA Naming Service. Das AgentSystemApplet, um eine Liste der gerade aktiven Agentensysteme zu erstellen, aus welcher der Benutzer das Zielsystem für einen Agententransfer auswählt. Das RegionManagementAgent-Applet, um beispielsweise seine Listen der Agentensysteme und Agenten zu erhalten. Doch wie auch schon beim CORBA Event Service haben Applets das Problem, nicht direkt auf den Naming Service zugreifen zu können, falls sich dieser nicht auf dem gleichen Rechner befindet wie der Webserver, von dem das Applet stammt.
Die Lösung besteht darin, die beiden Agenten, ASManagementAgent und RegionManagementAgent, die Naming Service Zugriffe erledigen zu lassen. Diese liefern dann die Ergebnisse an die Applets zurück. Dabei greift der ASManagementAgent nur auf den Agent-Kontext zu, während der RegionManagementAgent zusätzlich auch den AgentenSystem-Kontext kontaktiert.

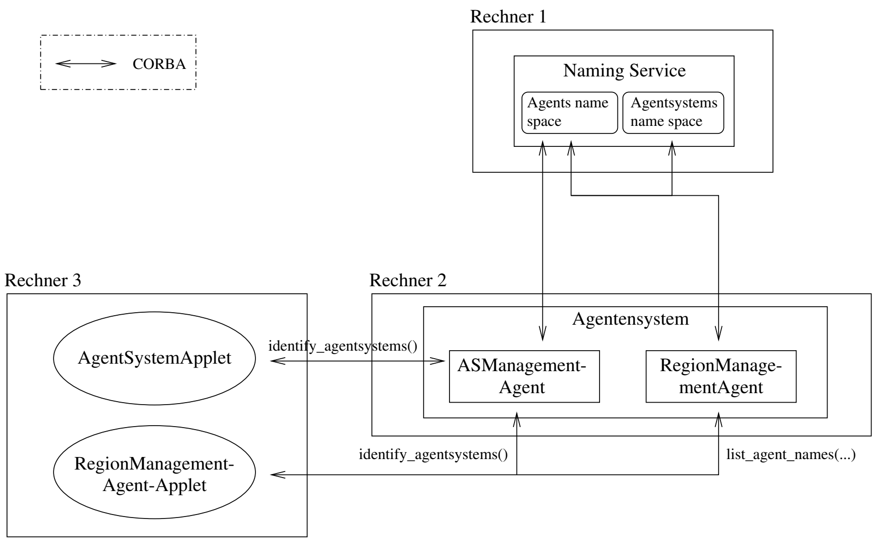

Abbildung 3.5: Kommunikation mit dem CORBA Naming Service über die Agenten

So bietet der ASManagementAgent beispielsweise die Methode *identify_agentsystems()* an, die eine Liste der Namen aller aktiven Agenten zurückliefert. Die Methode *list_agent_names(...)* des RegionManagementAgenten dagegen gibt für ein bestimmtes Agentensystem die Namen der darin geladenen Agenten zurück. Beide Informationen stammen aus dem CORBA Naming Service.

3.5 Zugriff auf fremde Agentensysteme und deren Agenten

Applets verbietet sich auch der Zugriff auf Agentensysteme, die nicht dem entsprechen, in dem der eigene Webserveragent gestartet wurde. Das RegionManagementAgent-Applet muß jedoch mit „fremden" Agentensystemen und Agenten arbeiten, etwa um Agenten zu verschieben, Agentensysteme zu beenden oder um die URL eines Agentensystems zu bestimmen. Diese Tätigkeiten werden daher vom Applet an den RegionManagementAgenten delegiert, der nach der Ausführung eventuelle Ergebniswerte zurückliefert. Die nachfolgende Abbildung verdeutlicht das eben gesagte nochmals.

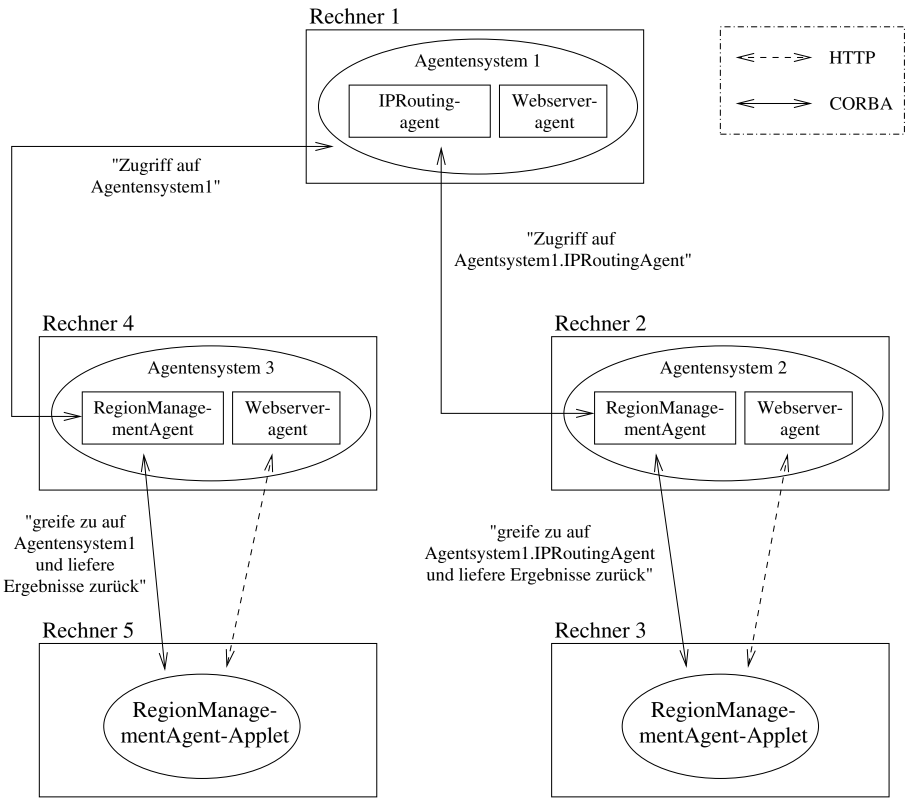

Abbildung 3.6: Zugriff auf Agenten bzw. Agentensysteme über den RegionManagementAgenten

3.6 Die IDL-Schnittstellen der Agenten

Damit andere Programme Methoden eines CORBA-Objektes aufrufen können, müssen dessen Schnittstellen mit Hilfe der sog. *Interface Definition Language (IDL)* beschrieben werden. Diese Beschreibung ist programmiersprachenneutral. Ein IDL-Präcompiler kann daraus die Client-Stubs und die serverseitigen Skeletons erstellen, die für eine Nutzung der Objektmethoden durch einen Client

notwendig sind.
Auch die Schnittstellen des ASManagementAgenten und des RegionManagementAgenten werden mittels IDL definiert, damit die Applets deren Methoden nutzen können.

3.6.1 ASManagementAgent

Es werden folgende Datentypen deklariert:

- *typedef string parameterlist*
  Dieser Datentyp stellt lediglich einen String dar und wird im nachfolgenden Datentyp verwendet. Er enthält die Namen der Typen der Parameter, die ein Konstruktor eines Agenten erwartet. Die einzelnen Namen werden dabei im String durch Leerzeichen voneinander getrennt.

- *typedef sequence< parameterlist > parametersets*
  Parametersets ist eine Sequenz von Parameterlisten (Strings). Parametersets enthält also in jedem Element eine Liste von durch Leerzeichen getrennten Parametertypnamen eines Agentenkonstruktors.

- *typedef string agentsystemname*
  Dieser Datentyp entspricht einem String, der den Namen eines Agentensystems aufnehmen soll.

- *typedef sequence< agentsystemname > agentsystem_list*
  Ein Feld von Agentensystemnamen.

- *typedef string packagename*
  Ein String, der den Namen eines Java-Packages aufnehmen soll.

- *typedef string agenttype*
  Ein String, der den Typ eines MASA-Agenten aufnehmen soll („StationaryAgent“ oder „MobileAgent“).

- *typedef long pushConsumer_idnr*
  Ein Integer, der die ID-Nummer eines PushConsumer-Objektes enthalten soll, das von einem Applet an den ASManagementAgenten mit Hilfe von *connect_push_consumer(...)* übergeben wird.

Folgende Methoden werden Clients zur Verfügung gestellt:

- *agentsystem_list identify_agentsystems()*
  Liefert eine Liste der geladenen Agentensysteme an die aufrufende Methode zurück.

- *parametersets get_agent_parameters(in string agent_name, in packagename agent_package, in agenttype agent_type)*
  Liefert eine Liste der Parametertypen aller Konstruktoren der Agentenklasse zurück, die durch die Argumente der Methode spezifiziert wird.

- *pushConsumer_idnr connect_push_consumer(in CosEventComm::PushConsumer pushConsumer)*
  Registriert einen PushConsumer (i.d.R. eines Applets) beim ASManagementAgenten und liefert die ID-Nummer zurück, die diesem PushConsumer vom ASManagementAgenten zugewiesen wurde.

- *void disconnect_push_consumer(in pushConsumer_idnr id)*
  Deregistriert anhand der übergebenen ID-Nummer den entsprechenden PushConsumer beim ASManagementAgenten.

Für eine nähere Beschreibung dieser Methoden siehe Kapitel 4.1.1 *Klasse ASManagementAgentStationaryAgent*.

3.6.2 RegionManagementAgent

Es werden folgende Datentypen deklariert:

- *typedef string agentsystem_url*
  Nimmt die String-Repräsentation einer URL eines Agentensystems auf.

- *typedef string agentsystem_name*
  String, der den Namen eines Agentensystems aufnimmt

- *typedef string agentname*
  String, der den Namen eines Agenten aufnimmt.

- *typedef sequence$<$ agentname $>$ agentname_list*
  Ein Feld von Agentennamen.

Folgende Methoden werden Clients zur Verfügung gestellt:

- *agentsystem_url get_AS_URL(in agentsystem_name as)*
  Liefert die URL des Agentensystems zurück, das durch as spezifiziert wird.

- *void terminate_agent_system(in agentsystem_name as)*
  Terminiert das Agentensystem mit Namen as.

- *void migrate_agent(in string agent_identity, in agentsystem_name target, in agentsystem_name as)*
  Migriert den Agenten agent_identity vom Agentensystem as ins Agentensystem target.

- *agentname_list list_agent_names(in agentsystem_name as)*
  Liefert eine Liste der Agentennamen des Agentensystems as zurück.

Für eine nähere Beschreibung dieser Methoden siehe Kapitel 4.2.1 *Klasse RegionManagementAgentMobileAgent*.

Kapitel 4

Die Implementierung

Dieses Kapitel stellt kurz die Klassen und die darin enthaltenen Methoden vor, durch die die beiden Applets und die Agenten realisiert werden.

4.1 Der ASManagementAgent

Der Hauptzweck dieses Agenten ist die Bereitstellung einiger Methoden, die es Applets erlauben, mit Diensten und Objekten zu kommunizieren, die möglicherweise nicht auf der gleichen Maschine ausgeführt werden wie der Webserver(agent), von dem das Applet stammt. Im einzelnen handelt es sich dabei um den CORBA Naming Service und den Event Service. Diese Proxy-Funktionalität des Agenten wird benötigt, da die Sicherheitsbestimmungen für Applets es nicht erlauben, auf Objekte und Ressourcen von im obigen Sinne „fremden" Rechnern zuzugreifen. Außerdem übernimmt der ASManagementAgent die Introspektion in Agentenklassen, um die Parameter ihrer Konstruktoren zu bestimmen, da den Applets die Kenntnis über die Lage dieser Klassen fehlt.
Der ASManagementAgent ist ein stationärer Agent, da er für die Verwaltung nur des Agentensystems zustaendig ist, auf dem er gestartet wurde. Er wird zusammen mit jedem Agentensystem gestartet, sofern dies in der Konfigurationsdatei *masa.properties* nicht anders eingestellt wird. Dies gewährleistet, daß den Applets, die im Zuge dieses Systementwicklungsprojekts entstanden sind, auf jedem Agentensystem seine Funktionalität, ohne die sie gar nicht funktionieren wuerden, zur Verfügung steht, und diese Funktionalität auch nicht durch eine Migration entzogen wird.

4.1.1 Klasse ASManagementAgentStationaryAgent

Diese Klasse enthält die eigentliche Implementierung des ASManagementAgenten. Beim ASManagementAgent handelt es sich um einen stationären Agenten, *ASManagementAgentStationaryAgent* erbt daher von der Klasse *StationaryAgent* (näheres siehe [Kemp 98]). Neben den Standardmethoden dieser Klasse (Konstruktor, *cleanUp()* und *checkSerialization()*), die in ihrer Funktionalität nicht verändert wurden, bietet der Agent die nachfolgenden Methoden.

Allgemeine Methoden

void run() Diese Methode wurde von der Klasse Agent geerbt. Sie wird vom Agentensystem aufgerufen, sobald ein Agent gestartet wird. Hier wird mit Hilfe der nachfolgend beschriebenen Methode *get_agent_context()* das entsprechende *NamingContext*-Objekt erzeugt. Außerdem wird ein Java-Thread vom Typ *ThreadEventListener* (siehe unten) gestartet.

Naming Service Proxy

void get_agent_context() Die MASA-Spezifikation sieht vor, daß im CORBA Naming Service sowohl die Agentensysteme, als auch die auf ihnen ausgeführten Agenten eingetragen werden. Diese Einträge befinden sich in *NamingContext*-Objekten. In UNIX-Dateisystem Schreibweise lautet dieser Kontext – relativ zu *_initContext*, dem Basis-*NamingContext* – für Agenten „/Agent“, für Agentensysteme „/AgentSystemService/4/mnm“.
Die Aufgabe der Methode besteht nun darin, ein Objekt vom Typ *org.omg.CosNaming.NamingContext* mit dem entsprechenden obigen *NamingContext*-Objekt aus dem CORBA Naming Service zu initialisieren und es dadurch anderen Methoden des ASManagementAgenten bereitzustellen. Der Bezeichner dieses Objekts ist *agent_context*. Es dient z.B. der Erstellung von Listen der aktiven Agentensysteme.
Der Aufruf dieser Methode erfolgt nur intern in der Methode *run()*, im Konstruktor des Agenten ist dies leider nicht möglich, da zum Zeitpunkt seines Aufrufs *_initContext* noch nicht initialisiert ist.

String[] identify_agentsystems() Dem aufrufenden Applet wird durch diese Methode ein String-Feld zurückgeliefert, das eine Namensliste aller gerade laufenden Agentensysteme enthält. Jedes Feldelement enthält einen Namen. Diese Liste wird aus dem CORBA Naming Service mit Hilfe des *NamingContext*-Objekts *agent_context* ermittelt, das alle Agentensysteme und ihre Agenten auflistet, zusätzlich aber auch – im Gegensatz zu *AS_context* – den Eintrag „global“ enthält, in dem alle Agenten nochmals aufgelistet werden, die mit der Option „exclusive“ gestartet wurden (Näheres dazu siehe Benutzerdokumentation zum RegionManagementAgentApplet Abschnitt 2.2). Diese Methode wird sowohl im AgentSystemApplet (Auswahl des Zielagentensystems beim Transfer eines Agenten), als auch im ASManagementAgentApplet (Ebenfalls Zielagentensystem-Auswahl, Auflistung der Agentensysteme im Applet) eingesetzt.

Klassenintrospektion

String[] get_agent_parameters(String agent_name, String agent_package, String agent_type) Mit Hilfe der übergebenen Parameter und dem Klassenpfad, der durch den Property-Eintrag *de.unimuenchen.informatik.mnm.masa.classCodeBase* festgelegt wird, lokalisiert diese Methode die Java-Klasse eines Agenten mit Namen *agent_name* und stellt anhand der darin gefundenen Konstruktoren des Agenten die Java-Typen der Parameter fest, die dieser bei seinem Aufruf erwartet. Eine Liste dieser Parameter wird dem aufrufenden Applet in Form eines String-Felds zurückgeliefert. Falls ein Agent über mehrere Konstruktoren verfügt, so werden die Parameterlisten der einzelnen Konstruktoren in jeweils ein Feldelement kopiert. Diese Methode findet man im AgentSystemApplet angewandt.

Event Service Proxy

int connect_push_consumer(org.omg.CosEventComm.PushConsumer pushConsumer) Diese Methode dient dazu, eine *PushConsumer*-Objektreferenz, die vom aufrufenden Applet übergeben wird, beim AS-ManagementAgenten zu „registrieren". Dort werden die übergebenen *PushConsumer*-Objektreferenzen in ein dafür vorgesehenes *java.util.Vector* Objekt kopiert, dessen Elemente Instanzen der Klasse *VectorElement* sind (siehe unten). Dem aufrufenden Applet wird eine ID-Nummer zurückgeliefert, mit deren Hilfe später das *PushConsumer*-Objekt zur „Deregistrierung" identifiziert werden kann. Diese Nummer wird zusammen mit der Objektreferenz in oben genanntem Vektor gehalten.
Näheres siehe auch Abschnitt 3.3 *Automatische Benachrichtigung der Applets bei Veränderungen.*

void disconnect_push_consumer(int id) Diese Methode wird von Applets aufgerufen, die vom Benutzer beendet werden. Dabei wird *id* übergeben, bei der es sich um die ID-Nr. handelt, die von connect_push_consumer(...) an das Applet zurückgeliefert wurde. Mit Hilfe dieser Nummer wird die zu diesem Applet gehörende *PushConsumer*-Objektreferenz identifiziert und gezielt beseitigt.

Überdies enthält die Klasse *ASManagementAgentStationaryAgent* zwei „inner classes". Beide dienen dem Umgang mit dem CORBA Event Service. Die Form der „inner classes" wurde gewählt, um Zugriff auf die Daten der Klasse *ASManagementAgentStationaryAgent* zu ermöglichen.

class ThreadEventListener implements Runnable

Diese Klasse definiert einen Java-Thread, der einen „Horchmechanismus" beim CORBA Event Service registriert nach dem sog. *Push-Verfahren.* (Siehe Abschnitt 3.3)

void run() Wird automatisch aufgerufen, sobald der Thread gestartet wird. Dabei wird das Agenten-eigene *PushConsumer*-Objekt (der Klasse *PushConsumerImpl*, siehe unten) in den Event Channel „eingeklinkt" und dann eine Warteschleife aktiviert. Über das *PushConsumer*-Objekt werden ab jetzt CORBA Ereignisse registriert.

class PushConsumerImpl extends _PushConsumerImplBase

Das *PushConsumer*-Objekt, das vom ASManagementAgent zum „Abhorchen“ des Event Channels benutzt wird, ist eine Instanz dieser Klasse. Sie definiert das Verhalten des Objekts bei eingehenden CORBA Ereignissen. Dazu wird die Methode *push(...)* implementiert, die automatisch aufgerufen wird, sobald ein CORBA Ereignis auftritt.

void push(org.omg.CORBA.Any any) Trifft ein CORBA Ereignis sein, so sorgt diese Methode für die Weiterleitung des Ereignisses und seiner Daten, die im Parameter *any* enthalten sind, an die *PushConsumer*-Objektreferenzen, die von den Applets an den Agenten übergeben wurden. Dazu wird für jede Objektreferenz, die zusammen mit ihrer ID-Nummer in der Datenstruktur des Agenten gehalten wird, *push(any)* aufgerufen.

class VectorElement

Diese Klasse definiert einen Datentyp, der zwei Variablen enthält: den *PushConsumer pushConsumer* und die int-Variable *id*. *VectorElement* wird als Elementtyp des Vektors benutzt, der die Objektreferenzen der *PushConsumer* aufnimmt, die von den Applets übergeben werden. Dadurch kann zusammen mit jeder dieser Referenzen auch die ID-Nummer gespeichert werden, die durch *connect_push_consumer(...)* vergeben wird. Das hat den Vorteil, daß die ID nicht jedesmal direkt aus dem *PushConsumer*-Objekt über CORBA-Kommunikationsmechanismen ausgelesen werden muß.

4.2 Der RegionManagementAgent

Der Hauptzweck dieses Agenten ist die Bereitstellung einiger Methoden, die es seinem RegionManagementAgent-Applet erlauben, mit Diensten und Objekten zu kommunizieren, die möglicherweise nicht auf der gleichen Maschine ausgeführt werden wie der Webserver(agent), von dem das Applet stammt. Im einzelnen handelt es sich dabei um den CORBA Naming Service und „fremde“ Agenten(-systeme). Diese Proxy-Funktionalität des Agenten wird benötigt, da die Sicherheitsbestimmungen für Applets es nicht erlauben, auf Objekte und Ressourcen von im obigen Sinne „fremden“ Rechnern zuzugreifen.
Der RegionManagementAgent wurde im Gegensatz zum ASManagementAgent als mobiler Agent konzipiert. Dadurch ist es möglich, diesen Agenten auf andere Agentensysteme zu migrieren und eine MASA-Region mit Hilfe des RegionManagementAgent-Applets von jedem beliebigen Agentensystem aus zu verwalten. Dabei können in der MASA-Region auch meherer Instanzen des Agenten aktiv sein (multipler Agent).

4.2.1 Klasse RegionManagementAgentMobileAgent

Diese Klasse enthält die eigentliche Implementierung des RegionManagement-Agenten. Beim RegionManagementAgent handelt es sich um einen mobilen

Agenten, *RegionManagementAgentMobileAgent* erbt daher von der Klasse *MobileAgent* (näheres dazu siehe [Kemp 98]). Neben den Standardmethoden dieser Klasse (Konstruktor, *cleanUp()* und *checkSerialization()*), die in ihrer Funktionalität nicht verändert wurden, bietet der Agent die nachfolgenden Methoden.

Allgemeine Methoden

void run() Diese Methode wurde von der Klasse Agent geerbt. Sie wird vom Agentensystem aufgerufen, sobald ein Agent gestartet wird. Hier werden mit Hilfe der nachfolgend beschriebenen Methoden *get_agent_context()* und *get_agentsystem_context()* die entsprechenden *NamingContext*-Objekte erzeugt.

Naming Service Proxy

void get_agent_context()* und *void get_agentsystem_context() Die MASA-Spezifikation sieht vor, daß im CORBA Naming Service sowohl die Agentensysteme, als auch die auf ihnen ausgeführten Agenten eingetragen werden. Diese Einträge befinden sich in *NamingContext*-Objekten. In UNIX-Dateisystem Schreibweise lautet dieser Kontext – relativ zu *_initContext*, dem Basis-*NamingContext* – für Agenten „/Agent“, für Agentensysteme „/AgentSystemService/4/mnm“.
Die Aufgabe der beiden Methoden besteht nun darin, jeweils ein Objekt vom Typ *org.omg.CosNaming.NamingContext* mit den entsprechenden obigen *NamingContext*-Objekten aus dem CORBA Naming Service zu initialisieren und sie dadurch anderen Methoden des RegionManagementAgenten bereitzustellen. Die Bezeichner dieser Objekte sind *agent_context* bzw. *AS_context*. Sie dienen z.B. der Erstellung von Listen der aktiven Agentensysteme und deren Agenten. Der Aufruf dieser Methoden erfolgt jeweils nur intern in der Methode *run()*, im Konstruktor des Agenten ist dies leider nicht möglich, da zum Zeitpunkt seines Aufrufs *_initContext* noch nicht initialisiert ist.

String[] list_agent_names(String as) Es wird dem aufrufenden Applet in Form eines String-Feldes eine Liste der Namen der Agenten zurückgeliefert, die im durch den String *as* spezifizierten Agentensystem gestartet wurden. Die Liste wird aus dem CORBA Naming Service mit Hilfe des NamingContext-Objekts *agent_context* ermittelt.

Agenten/Agentensystem Proxy

AgentSystemService get_agentsystemservice(String as) Zweck dieser Methode ist die Rückgabe einer Referenz des *AgentSystemService*-Objektes desjenigen Agentensystems, das durch den String-Parameter *as* spezifiziert wird. Dazu wird das *NamingContext*-Objekt *AS_context* benutzt, das die Referenzen der Agentensysteme enthält. Diese Methode wird nur intern im Agenten von den Methoden *migrate_agent(...)*, *get_AS_URL(...)* und *terminate_agent_system(...)* verwendet.

void migrate_agent(String agent_identity, String target, String as) Diese Methode erlaubt es, den durch *agent_identity* spezifizierten Agenten vom Agentensystem *as* in das Agentensystem „target“ zu verschieben. Jeder mobile Agent bietet dazu die Methode *migrateToAgentSystem(...)*, die auch hier zum Einsatz kommt. Diese Methode wird vom RegionManagementAgentApplet benötigt, um mit Agenten „fremder“ Agentensysteme zu arbeiten.

String get_AS_URL(String as) Mit Hilfe des *AgentSystemService*-Objekts des Agentensystems mit Namen *as* wird die URL des entsprechenden Systems ermittelt und an das aufrufende Applet als String übergeben. Es handelt sich dabei um die URL, unter der der entsprechende Webserveragent die Homepage des Agentensystems bereitstellt.

void terminate_agent_system(String as) Das durch den Übergabeparameter *as* spezifizierte Agentensystem wird von dieser Methode beendet, indem die Methode *terminate_agent_system()* des dazu gehörigen *AgentSystemService*-Objekts aufgerufen wird.

4.3 Die Applets

4.3.1 Klasse CommonAppletHelpers

Diese Klasse enthält Programmcode, der von beiden in dieser Dokumentation beschriebenen Applets benutzt wird, und verhindert dadurch schwer zu wartende Coderedundanz.
Sie enthält zwei Methoden, die beide *public static* deklariert werden und daher aus anderen Klassen direkt aufgerufen werden können.

java.net.URL getAgentURL(String identity, String host) Diese Methode liefert die URL eines Agenten auf dem Agentensystem *host* zurück, der durch *identity* spezifiziert wird. Dazu wird an den Webserveragenten ein HTTP-Request geschickt nach dem Schema *http://host/identity.url*.

String show_migrate_menu(Component parentComponent, String agent_identity, java.util.Vector as_names) Diese Methode erstellt ein Eingabedialogfenster, das dazu dient, das Zielagentensystem aus dem Vektor *as_names* auszuwählen, in das der Agent mit Namen *agent_identity* verschoben werden soll. *as_names* enthält eine Liste der verfügbaren Agentensysteme. Ist darin der Eintrag „global“ enthalten, so wird dieser nicht in die Auswahlliste aufgenommen, da es sich dabei nicht um ein Agentensystem im eigentlichen Sinne handelt. Der Übergabeparameter *parentComponent* ist mit einer Referenz auf das aufrufende Applet belegt. Er wird benötigt, da das erscheinende Auswahldialogfenster ein „Kind“ des Applets ist.

String retrieve_implemented_agents_file(String host) Diese Methode kontaktiert den Webserver-Agenten, um von ihm die Datei *ImplementedAgents.txt* via *HTTP* zu erhalten. Die Applets lesen die Datei nicht direkt aus, da sie einmal aufgrund der Sicherheitsrestriktionen dazu nicht berechtigt sind und zum anderen die Java-Properties des Agentensystems nicht besitzen, in denen u.a. die Position der Datei im Filesystem beschrieben wird.

4.3.2 Klasse AgentSystemApplet

Durch die Klasse *AgentSystemApplet* wird das Applet eines Agentensystems definiert. Es ist genau einem Agentensystem zugeordnet und wird gestartet, sobald der Webserver-Agent des betreffenden Agentensystems kontaktiert wird. *AgentSystemApplet* ist Unterklasse von *AgentApplet*. Folgende Methoden werden deklariert:

Allgemeine Methoden

void init() Dies ist eine Standardmethode jeden Applets, die beim Start ausgeführt wird. Sie dient der Initialisierung eines Applets. Zuerst werden CORBA Referenzen des zugehörigen Agentensystems und des ASManagementAgenten gewonnen. Da ein laufender ASManagementAgent für die korrekte Funktion des Applets notwendig ist, wird anschließend überprüft, ob eine CORBA Referenz tatsächlich vorliegt. Falls nicht, wird die weitere Abarbeitung der Applet-Methoden gestoppt und ein Warnhinweis gezeigt.
Ansonsten werden nacheinander für die drei Panels *Manage Agents*, *Create Agent* und *Agent System* die GUI-Elemente erzeugt und verschiedene Methoden aufgerufen, die anfängliche Informationen sammeln. Mit *list_agent_names()* wird eine Liste der momentan geladenen Agenten und ihres Zustands erzeugt, *get_impl_agents()* sammelt Informationen über ladbare Agenten. Aus der daraus erstellten Liste ladbarer Agenten wird standardmäßig der erste mit Index 0 ausgewählt und mit *get_agent_parameters()* eine Aufstellung seiner Konstruktoren gebildet. Die Typen der erwarteten Parameter des ersten Konstruktors werden mit *show_agent_parameters()* in der dafür vorgesehenen Tabelle angezeigt.
Anschließend wird ein *PushConsumer* der Klasse *PushConsumerImpl* mit *create_pushConsumer()* beim ASManagementAgenten registriert, um über CORBA Events benachrichtigt zu werden. Mit *initEventHandler()* werden abschließend den GUI-Elementen entsprechende ActionListener und MouseListener zugeordnet.

void initEventHandler() Diese Methode wird nur einmal aufgerufen, nämlich beim Start des Applets in der Methode *init()*. Sie sorgt dafür, daß jedem Button, der in *init()* eingerichtet wurde, ein entsprechender *ActionListener* zugeordnet wird. Jeder *ActionListener* definiert die Aktion, die ausgeführt wird, wenn ein Button gedrückt wird. Zu diesem Zweck wird die Methode *actionPerformed(...)* jedes *ActionListeners* implementiert. Um zu verhindern, daß Aktionen auf Agenten versehentlich ausgeführt werden, wird am Ende der *action-*

Performed(...)-Methoden der Buttons des *Manage agents* Panels die Methode *clear_selection()* aufgerufen, die dafür sorgt, daß eben diese Buttons deaktiviert werden und die momentane Selektion der Agententabelle gelöscht wird.
Überdies wird in *initEventHandler()* ein *MouseListener* für die Tabelle der geladenen Agenten definiert. Dessen Methode *mousePressed(...)* sorgt dafür, daß im Falle eines Doppelklicks auf einen Agenten der Tabelle ein neues Browserfenster mit dem Applet des Agenten geöffnet wird.

void clear_selection() Durch diese Methode werden alle Buttons, die einen selektierten Agenten in der Agententabelle benötigen, deaktiviert und die momentane Selektion der Tabelle gelöscht. Ein Aufruf dieser Methode nach jeder Agenten-manipulierenden Aktion verhindert, daß für denselben Agenten versehentlich eine weitere Aktion ausgeführt wird.

Naming Service bezogene Methoden

void list_agent_names() Diese Methode erstellt mit Hilfe der CORBA Referenz auf das Agentensystem eine Liste aller geladenen Agenten. Ferner wird für jeden Agenten mit Hilfe der Methode *AgentSystem.get_agent_status(...)* sein Status bestimmt („running“/„suspended“). Diese Informationen werden im TableModel der Agententabelle gespeichert, für welche abschließend ein Refresh ausgeführt wird, um die neuen Informationen anzuzeigen.

Event Service bezogene Methoden

void create_pushConsumer() Auch diese Methode wird nur einmal pro Applet in dessen *init()* Methode aufgerufen. Sie erzeugt eine Instanz der Klasse *PushConsumerImpl* (siehe unten) und übergibt dem ASManagementAgenten über dessen Methode *connect_push_consumer(...)* eine Referenz auf dieses Objekt. Die zurückgelieferte ID-Nummer wird in der *id*-Variable des *PushConsumer*-Objekts abgelegt.

void destroy() Auch *destroy()* ist eine Standardmethode der Klasse Applet. Sie wird ausgeführt, wenn das Applet beendet oder neu geladen wird, und dient dazu, Ressourcen wieder freizugeben. In diesem Fall wird der *PushConsumer*, der in *init()* per *create_pushConsumer()* erstellt wurde, wieder beim ASManagementAgenten abgemeldet, indem dessen Methode *disconnect_push_consumer(...)* mit der ID-Nummer des *PushConsumers* aufgerufen wird.

Methoden für den Agentenstart

void get_impl_agents() Diese Methode benutzt *CommonAppletHelpers.retrieve_implemented_agents_file(...)*, um vom Webserver-Agenten die Datei *ImplementedAgents.txt* zu erhalten. Die Informationen in dieser Datei (Agentennamen, Java-Package und Agententyp) werden in drei Felder kopiert, so daß sie anderen Methoden des Applets, insbesondere dem ActionListener des *create* *agent* Buttons, zur Verfügung stehen. Das Applet liest die Datei nicht direkt aus, da es einmal aufgrund der Sicherheitsrestriktionen dazu nicht berechtigt ist und zum anderen die Java-Properties des Agentensystems nicht besitzt, in denen u.a. die Position der Datei im Filesystem beschrieben wird.

void get_ agent_ parameters() Sobald der Benutzer aus der Liste der implementierten Agenten des Panels *Create agent* einen Agenten auswählt, der gestartet werden soll, wird diese Methode aufgerufen. Hier wird über den ASManagementAgenten eine Auflistung aller Parametertypen der verschiedenen Konstruktoren des gewählten Agenten gewonnen. Ein Feld von Vektoren wird erstellt, jedes Feldelement enthält die Parametertyp-Liste eines Konstruktors. Außerdem wird ein String-Feld *constructor_ list[]* erzeugt, das eine Auswahl der verschiedenen Konstruktoren des selektierten Agenten anhand ihrer Parameterzahl enthält. Dieses Feld wird von der nachfolgend aufgerufenen Methode *repaint_ ch_ constructors()* benötigt, um die Combobox, die dem Benutzer eine Auswahl der möglichen Konstruktoren des selektierten Agenten bietet, zu aktualisieren.
Der Grund dafür, daß die Parameterliste über den ASManagementAgenten erzeugt wird, ist die Tatsache, daß diese Information durch Introspektion in die Klasse des entsprechenden Agenten über das *Reflection API* von Java erzeugt wird. Dies ist dem Applet aufgrund seiner Sicherheitsbeschränkungen nicht möglich.

void repaint_ ch_ constructors() Diese Methode definiert die Combobox, die dem Benutzer eine Auswahl der möglichen Konstruktoren des selektierten Agenten bietet, anhand des String Feldes *constructor_ list[]*, das in der Methode *get_ agent_ parameters()* erzeugt wird. Anschließend wird der erste Konstruktor der Liste vorselektiert und für diesen die Methode *show_ agent_ parameters()* aufgerufen.

void show_ agent_ parameters(int param_ index) Mit Hilfe des Übergabeparameters *param_ index* wird aus dem Feld mit Parametertyp-Listen, das in *get_ agent_ parameters()* erzeugt wurde, die Liste für den richtigen Konstruktor ausgewählt. Die Elemente dieser Liste werden in der dafür vorgesehenen zweispaltigen Tabelle in der linken Spalte eingetragen. Rechts neben jedem eingetragenen Element wird in der zweiten Spalte der Wert des Parameters mit „none“ vorbelegt.

Außerdem werden folgende „inner classes“ definiert:

class ParameterTableModel extends AbstractTableModel

Diese Klasse definiert das TableModel für die Tabelle der Parameter, die aktualisiert wird, sobald der Benutzer einen Konstruktor aus der Konstruktorliste des Applets für eine Agentenklasse auswählt. Als Datenstruktur, die die Parametertypen und die vom Benutzer eingegebenen Werte aufnimmt, kommt ein *java.util.Vector*-Objekt zum Einsatz. Neben den Standardmethoden eines TableModels (*int getColumnCount()*, *int getRowCount()*, *String getColumnName (int col)*, *Object getValueAt(int row, int col)*, *void setValueAt(Object value, int row, int col)*) wurden zusätzlich definiert:

void addValue(String[] value) Fügt den Parameter *value* an das Ende des Vectors an.

void rmValue(int row) Löscht das Element des Vektors, dessen Position durch *row* spezifiziert wird.

boolean isElement(String an) Prüft, ob der angegebene String *an* in der ersten Tabellenspalte auftaucht, welche die Parametertypen enthält. Das Ergebnis wird als *boolean*-Wert zurückgeliefert.

boolean isCellEditable(int row, int col) Gibt *false* zurück, falls *col* kleiner als 1 ist, verhindert also ein Editieren der ersten Spalte, welche die Parametertypen auflistet.

class AppletTableModel extends AbstractTableModel

Diese Klasse definiert das TableModel der Tabelle, welche die Liste der geladenen Agenten mit ihrem Status aufnimmt. Wieder findet ein *java.util.Vector*-Objekt Verwendung, um die Daten aufzunehmen. Auch hier werden zusätzliche Methoden neben den Standardmethoden definiert:

void addValue(String value) Fügt den Parameter *value* an das Ende des Vektors an.

void rmValue(int row) Löscht das Element des Vektors, dessen Position durch *row* spezifiziert wird.

void changeStatus(int row, String statusValue) Diese Methode dient dazu, den Statuswert eines Agenten in Zeile *row* der Tabelle auf *statusvalue* zu setzen.

int whereis(String an) Liefert die Nummer der Zeile des Vectors, in der der Agent mit dem Namen *an* enthalten ist.

class ListSelectionHandler implements ListSelectionListener

Durch diese Klasse wird ein *ListSelectionHandler* für die Tabelle mit geladenen Agenten definiert. Zu diesem Zweck wird eine Methode implementiert:

void valueChanged(ListSelectionEvent e) Diese Methode wird aktiviert, sobald ein Element der überwachten Tabelle ausgewählt wurde (z.B. mit der Maus). Falls die Wahl ungültig ist, der Auswahlindex also kleiner als 0 ist, werden sämtliche Buttons deaktiviert, die einen korrekt ausgewählten Agenten voraussetzen. Außerdem wird die String-Variable *agent_identity* auf *null* gesetzt.
Ist die Wahl jedoch gültig (Auswahlindex $>$ 0), so werden alle Buttons des Agenten-Managements aktiviert und *agent_identity* der Name des aktuell ausgewählten Agenten zugewiesen. Eine Ausnahme gilt für den Button, der die Migration eines Agenten auslöst. Bevor dieser aktiviert werden kann, wird anhand der Informationen, die aus der Datei *ImplementedAgents.txt* gewonnen wurden (siehe Abschnitt 3.1 *Erstellung der Übersicht ladbarer Agenten*) überprüft, ob es sich bei dem ausgewählten Agenten um einen mobilen Agenten handelt.

class PushConsumerImpl extends _PushConsumerImplBase

Die Klasse definiert den spezifischen *PushConsumer* dieses Applets. Eine Instanz dieser Klasse wird beim Starten des Applets an den ASManagementAgenten übergeben. Dieser ruft die *push(...)* Methode des Objekts auf und übergibt dabei die Daten des CORBA Events, der aufgetreten ist. Folgende Methoden sind realisiert:

void push(org.omg.CORBA.Any any) Die Event-Daten in *any* werden ausgewertet. Dabei wird festgestellt, um welche Art von Ereignis es sich handelt. Reagiert wird auf:

- AgentUp
- AgentDown

4.3.3 Klasse RegionManagementAgentApplet

Diese Klasse definiert das Applet des RegionManagementAgenten. Sie ist, wie die Klassen aller Agentenapplets, eine Unterklasse von *AgentApplet.* Folgende Methoden sind definiert:

Allgemeine Methoden

void init() Dies ist eine Standardmethode jeden Applets, die beim Start ausgeführt wird. Sie dient der Initialisierung eines Applets. In diesem Fall wird zuerst eine Verbindung zum Agentensystem, zum ASManagementAgenten und zum RegionManagementAgenten aufgebaut, indem eine CORBA Referenz derselben erzeugt wird. Da ein laufender ASManagementAgent für die korrekte Funktion des Applets notwendig ist, wird anschließend überprüft, ob eine CORBA Referenz tatsächlich vorliegt. Falls nicht, wird die weitere Abarbeitung der Applet-Methoden gestoppt und ein Warnhinweis gezeigt.
Danach werden alle Komponenten der Bedienungsschnittstelle eingerichtet (Fenster, Buttons, Bäume, Tabellen etc.) und verschiedene Methoden aufgerufen, um

anfängliche Daten über laufende Agentensysteme und deren Agenten zu erhalten. Dazu gehören die Liste der aktiven Agentensysteme und eine Aufstellung der Agenten, die auf dem ausgewählten Agentensystem geladen sind.
Schließlich wird die Methode *create_pushConsumer()* aufgerufen, die dafür sorgt, daß ein *PushConsumer*-Objekt der Klasse *PushConsumerImpl* (siehe unten) beim ASManagementAgenten registriert wird.

void migrate_agent() Diese Methode wird aufgerufen, wenn der entsprechende Menüpunkt des Kontextmenüs eines Agenten der Agentenliste ausgewählt wird. Es wird zuerst mit Hilfe der Methode *show_migrate_menu(...)* der Klasse *CommonAppletHelpers* ein Dialogfenster zur Auswahl des Zielagentensystems erzeugt. Dann wird mittels der Methode *migrate_agent(...)* des RegionManagementAgenten der Agent auf das neue Agentensystem transferiert. Der Umweg über den RegionManagementAgenten muß beschritten werden, weil auch Agenten „fremder“ Agentensysteme betroffen sein können.

Naming Service bezogene Methoden

void create_tree() Diese Methode erzeugt den Baum des Applets, der die aktiven Agentensysteme auflistet. Sie wird nur einmal beim Starten des Applets durch *init()* aufgerufen. Die Liste der geladenen Agentensysteme wird über die Methode *identify_agentsystems()* des ASManagementAgenten ermittelt. Für den Baum wird ein *TreeSelectionListener* definiert, der dafür sorgt, daß die Liste der Agenten aktualisiert wird, wenn ein neues Agentensystem im Baum selektiert wird. Ein *MouseListener* bewirkt, daß bei einem Doppelklick auf ein Element des Baums für das betreffende Agentensystem *show_agentsystems_webpage(...)* aufgerufen wird, bei einem Klick mit der rechten Maustaste dagegen das Agentensystem-Kontextmenü angezeigt wird.

void update_tree() Diese Methode aktualisiert den Agentensystem-Baum. Sie wird aufgerufen, wenn der *update manually* Button gedrückt wurde. Über *identify_agentsystems()* des ASManagementAgenten wird erneut die Liste der aktiven Agentensyteme ermittelt und mit dieser Information der Baum neu aufgebaut.

void list_agent_names() Sobald der Benutzer ein Agentensystem im Baum des Applets auswählt, sorgt dessen *TreeSelectionListener* dafür, daß diese Methode aufgerufen wird. Sie stellt mittels der Methode *list_agent_names(...)* des RegionManagementAgenten die Liste der Agenten für das ausgewählte Agentensystem neu auf.

Event Service bezogene Methoden

void create_pushConsumer() Wie auch schon beim AgentSystemApplet wird diese Methode in *init()* aufgerufen. Sie erzeugt eine Instanz der Klasse

PushConsumerImpl (siehe unten) und übergibt eine Referenz davon an den ASManagementAgenten über dessen Methode *connect_push_consumer(...)*. Die zurückgelieferte ID-Nummer wird im *PushConsumer*-Objekt abgelegt.

void destroy() Dabei handelt es sich ebenfalls um eine Standardmethode eines Applets. Sie wird ausgeführt, wenn das Applet beendet oder neu geladen wird. Sie dient dazu, Ressourcen wieder freizugeben. In diesem Fall wird der *PushConsumer*, der in *init()* per *create_pushConsumer()* erstellt wurde, wieder abgemeldet, indem *disconnect_push_consumer(...)* des ASManagementAgenten mit der ID-Nummer des *PushConsumers* aufgerufen wird.

Methoden zum Anzeigen von Webseiten der Agenten/-systeme

void show_agentsystems_webpage(String as) Diese Methode öffnet ein neues Browserfenster, in dem die Webpage des Agentensystems angezeigt wird, das durch den Übergabeparameter *as* spezifiziert wird. Die URL des Agentensystems wird über die Methode *get_AS_URL(...)* des RegionManagementAgenten ermittelt, da vom Applet aus kein direkter Zugriff auf alle Agentensysteme möglich ist.

void show_agents_webpage() Auch diese Methode öffnet ein neues Browserfenster, zeigt darin allerdings die Webpage des Agenten, in dessen Kontextmenü der Menüpunkt *show webpage* gewählt wurde. Die URL des Agenten wird über die Methode *getAgentURL(...)* der Klasse *CommonAppletHelpers* in Erfahrung gebracht.

Methoden zum Erzeugen der Popupmenüs

JPopupMenu create_as_popup(final String as) Definiert das Kontextmenü, das angezeigt wird, wenn die rechte Maustaste auf einem Agentensystem im Baum gedrückt wird. Zwei Menüpunkte werden definiert, der eine ruft *show_agentsystems_webpage(...)* auf, der andere *terminate_agent_system(...)* des RegionManagementAgents. Beide male wird der Name des ausgewählten Agentensystems in Form des Parameters *as* übergeben.
Das erzeugte Menü wird der aufrufenden Methode übergeben.

JPopupMenu create_popup() Erzeugt das Kontextmenü, das bei einem Klick mit der rechten Maustaste auf einen Agenten der Agentenliste angezeigt wird. Zwei Menüpunkte werden definiert, der eine ruft *migrate_agent()* auf, der andere *show_agents_webpage()*. Rückgabewert ist das erzeugte Menü.

Außerdem sind folgende „inner classes" definiert:

class PushConsumerImpl extends _PushConsumerImplBase

Die Klasse definiert den spezifischen *PushConsumer* dieses Applets. Eine Instanz dieser Klasse wird beim Starten des Applets an den ASManagementAgenten übergeben. Dieser ruft die *push(...)* Methode des Objekts auf und übergibt dabei die Daten des CORBA Events, der aufgetreten ist.

Methoden:

void push(org.omg.CORBA.Any any) Die Event-Daten in *any* werden ausgewertet. Dabei wird festgestellt, um welche Art von Ereignis es sich handelt. Reagiert wird auf:

- agentSystemUp
- agentSystemDown
- AgentUp
- AgentDown

class AppletTableModel extends AbstractTableModel

Diese Klasse definiert das Tabellenmodell für die Tabelle der Agenten, die im ausgewählten Agentensystem enthalten sind. Die Namen der Agenten werden in einem *java.util.Vector*-Objekt gehalten, das dynamisch verändert werden kann. Neben den Standardmethoden eines TableModels (*int getColumnCount()*, *int getRowCount()*, *String getColumnName (int col)*, *Object getValueAt(int row, int col)*, *void setValueAt(Object value, int row, int col)*) wurden zusätzlich definiert:

void addValue(String value) Fügt den Parameter *value* an das Ende des Vectors an.

void rmValue(int row) Löscht das Element des Vektors, dessen Position durch *row* spezifiziert wird.

int whereis(String an) Liefert die Nummer der Zeile des Vectors, in der der Agent mit dem Namen *an* enthalten ist.

Kapitel 5

Ausblick

5.1 Schwierigkeiten bei der Entwicklung: BOA vs. POA

Einige Probleme bereitete die Implementierung der automatischen Benachrichtigung der Applets bei CORBA-Events durch den ASManagementAgenten. Wie in Kapitel 3.3 bereits beschrieben, erfolgt die Kommunikation zwischen Applets und dem ASManagementAgenten über die CORBA Kommunikationsmechanismen. Applets definieren eigene *PushConsumer*-Objekte, die sie beim Start dem zugehörigen ASManagementAgenten übergeben, welcher anschließend eingehende Events an diese Objekte (und damit an die Applets) weitergibt. Die *PushConsumer*-Instanzen der Applets „leben" jedoch weiterhin bei den Applets selbst. Damit der ASManagementAgent die *push(...)*-Methoden der Applet-*PushConsumer* überhaupt aufrufen kann, muß das Applet daher als CORBA-Objekt-Server agieren, d.h. den *PushConsumer* über den *Object Adapter* instanziieren und beim ORB (*Object Request Broker*) registrieren. Erst dann kann eine Methode dieses Objekts außerhalb des Applets, in diesem Fall vom ASManagementAgenten, benutzt werden.
Leider treten dabei mit dem verwendeten ORB *VisiBroker 3.0* Probleme auf, wenn dazu der CORBA 2.0 konforme *Basic Object Adapter* (BOA) verwendet werden soll. Aufgrund der Unterspezifikation des BOAs in der Version 2.0 von CORBA ergänzen viele ORB-Hersteller ihre Produkte um proprietäre Lösungen, um diese Spezifikationslücken zu schließen. Im Falle von *VisiBroker 3.0* wird zwingend eine installierte und konfigurierte Version des *Gatekeeper*, einer proprietären IIOP Firewall von Inprise/Visigenic, vorausgesetzt, bevor ein Applet mit Hilfe des BOAs ein Objekt verfügbar machen kann. Dies ist jedoch unerwünscht.
Glücklicherweise unterstützt *VisiBroker 3.0* jedoch bereits den (ausführlicher spezifizierten) *Portable Object Adapter* (POA) der CORBA 3.0 Spezifikation, welcher den Einsatz des *Gatekeepers* unnötig macht. Die Registrierung der *PushConsumer* beim ORB durch die Applets erfolgt daher durch eine Instanz des POA.

5.2 Mögliche Weiterentwicklungen: Konfigurationsmanagement mittels Property Service

Im Rahmen eines FOPRAS wird zum Zeitpunkt der Entstehung dieser Dokumentation ein Property Service für MASA entworfen, der unter anderem ein zentrales Archiv aller implementierten Agenten zur Verfügung stellen wird. Dadurch kann auf die Datei „ImplementedAgents.txt“ verzichtet werden.

Anhang A

Listing der IDL-Schnittstelle des ASManagementAgenten

```
// the next two lines are C preprocessor directives
// which are allowed in CORBA IDL
#ifndef _ASManagementAgent_idl_
#define _ASManagementAgent_idl_

#include "CosEventComm.idl"

// this module will be converted to
// a java package 'ASManagementAgent'
module ASManagementAgent {

  interface ASManagementAgent
  {
        //************************
        //      data types
        //************************

        //parameterlist is a string containing the parameters of one constructor
        //separated by spaces
        typedef string parameterlist;

        //return value for get_agent_parameters(...). It is an array of parameterlists
        typedef sequence< parameterlist > parametersets;

        //agentsystemname is a string containing the name of an agent system
        typedef string agentsystemname;

        //agentsystem_list is a sequence of agentsystemnames containing the names of the
        //agent systems found in the appropriate naming context of the CORBA Naming Service.
        typedef sequence< agentsystemname > agentsystem_list;

        //packagename is a string containing the name of a Java package
        typedef string packagename;

        //agenttype is a string containing the MASA type of an agent. This can be
        //"StationaryAgent" or "MobileAgent"
        typedef string agenttype;

        //pushConsumer_idnr is an integer containing the id number of a pushConsumer
        //that has been registered by an applet with the ASManagementAgent. It's value
        //is assigned by ASManagementAgent.connect_push_consumer(...)
        typedef long pushConsumer_idnr;


        //************************
        //  methods
        //************************

        //get list of active agent systems
        agentsystem_list identify_agentsystems();

        //get parameter types for agent "agent_name"
        parametersets get_agent_parameters(in string agent_name, in packagename agent_package, in agenttype agent_type);

        //connect pushConsumer of applet to the ASManagementAgent
        pushConsumer_idnr connect_push_consumer(in CosEventComm::PushConsumer pushConsumer);

        //disconnect pushConsumer  of applet from ASManagementAgent
        void disconnect_push_consumer(in pushConsumer_idnr id);
  };
};
#endif
```

Anhang B

Listing des ASManagementAgenten

```
package __MASA_PACKAGE_thisagent;

import __MASA_PACKAGE_agentSystem__(*);
import __MASA_PACKAGE_agent__(*);
import __MASA_PACKAGE_CfMAF__(*);
import __MASA_PACKAGE__(tools.NameWrapper);
import __MASA_PACKAGE__(tools.Debug);
import __MASA_PACKAGE__(tools.GlobalConstants);

import java.io.*;
import java.util.*;

import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosNaming.*;


/**
 * This is the class defining the ASManagementAgent.
 * <br>The ASManagementAgent is a stationary agent which primary task it is to provide methods
 * that allow applets to communicate with objects and CORBA services. In particular these are
 * other agent systems, the CORBA Name Service and the CORBA Event Service, which are often not located
 * on the same machines as the applets. The ASManagementAgent therefore acts as a proxy for these applets.
 *
 *      @see StationaryAgent
 *      @author Stefan Gerber
 */

public class ASManagementAgentStationaryAgent extends StationaryAgent
    implements ASManagementAgentOperations {

    private org.omg.CosNaming.NamingContext agent_context; // CORBA naming context for agents
    private ThreadEventListener eventlistener = null;      //thread listening for CORBA events
    private Vector pushConsumerVector = new Vector(0,1);   //vector containing the pushConsumers that the applets register with the ASManagementAgent
    private int id_incr = 0;                               //used to determine the IDs for the applet-pushConsumers
    private Thread el_thread = null;  //thread that will run the ThreadEventListener



    public ASManagementAgentStationaryAgent() {
        super();
        //get_agent_context();  //doesn't work here, because _initContext = null at this moment
    }

    public void cleanUp() throws Throwable {
        System.err.println("ASManagementAgentMobileAgent.cleanUp()");
        el_thread.stop();  //stop the eventlistener thread
    }

    public void checkSerialization()
        throws CouldNotMigrate {

    }

    /**
     * Calls methods to fetch the naming contexts of agents/agent systems and starts the EventListener thread.
     * <br>This method is called, when the agent is started.
     */

    public void run() {
        get_agent_context();
        //Start the Thread that listens for CORBA events
        eventlistener = new ThreadEventListener(_initContext);
        el_thread = new Thread(eventlistener);
        el_thread.start();
    }


    /**
     * Fetches the naming context containing the agents of the different agent systems.
     * <br>The naming context is saved into agent_context. This method is only for internal use.
     */

    void get_agent_context() {
        //get context of agents
        try {
            NameComponent comp1 = new NameComponent("Agent", "");
            NameComponent name[] = {comp1};
            agent_context = NamingContextHelper.narrow(_initContext.resolve(name));
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("\nfailed :" + e.toString());
        }

    }


    /**
     * Registers a "client´s" PushConsumer.
     * <br>Clients (e.g. applets), which want to use the ASManagementAgent as CORBA event service proxy,
     * call this method once to register their PushConsumers. An id number is returned to each client. This
     * number is used by clients to deregister their PushConsumers with disconnect_push_consumer().
     *
     * @param pushConsumer PushConsumer that´s to be registered
     * @return ID number of type int
     */

    public int connect_push_consumer(org.omg.CosEventComm.PushConsumer pushConsumer) {
        //add the new pushConsumer and it's id to the global vector
        VectorElement ve = new VectorElement(pushConsumer, id_incr);
        pushConsumerVector.addElement(ve);

        System.err.println("\nconnect_push_consumer: pushConsumer added");
        System.err.println("connect_push_consumer: Now "+pushConsumerVector.size()+" PushConsumers registered");
        return id_incr++;  //return id for this new pushConsumer and increment id_incr afterwards
    }


    /**
```

```
 * Deregisters a "client´s" PushConsumer.
 * <br>This method is invoked by clients (e.g. applets) that want to disconnect their PushConsumers from
 * the ASManagementAgent. The PushConsumer with the ID number "id" is removed from the global vector
 * containing all registered PushConsumers. If the client is an applet then its destroy()-method is
 * a good place to invoke this method from.
 *
 * @param id ID of PushConsumer to be deregistered
 */

public void disconnect_push_consumer(int id) {
    int size = pushConsumerVector.size();  // get size of vector
    for (int i=0; i<size; i++) {
        VectorElement ve = (VectorElement) pushConsumerVector.elementAt(i); //get element at index i
        if (ve.id == id) {
            //if id has been found remove the vector element
            pushConsumerVector.removeElementAt(i);
            System.err.println("\n\nASManagementAgent: pushConsumer (id="+id+") removed");
            return;
        }
    }
    System.err.println("\n\nASManagementAgent: pushConsumer to remove has not been found");
}


/**
 * Returns a String[] containing a list of the names of the currently active agent systems.
 * <br>This list is determined using the NamingContext object agent_context, which also
 * contains the name space "global". "global" lists all agents that have been started with
 * the option "exclusive". For further information consult the MASA manual.
 *
 * @return String[] containing the names of active agent systems
 */

public String[] identify_agentsystems() {
    //get the naming context from the CORBA name service
    //get_agent_context();

    //get names of Agent Systems
    BindingListHolder bl_holder = new BindingListHolder();
    BindingIteratorHolder bi_holder = new BindingIteratorHolder();
    agent_context.list(1000000, bl_holder, bi_holder);
    Binding[] AS_bindings = bl_holder.value;
    String as_names[] = new String[AS_bindings.length];

    //copy the names separated by spaces into the String as_names
    for (int i=0; i<AS_bindings.length; i++) {
        as_names[i] = (String) AS_bindings[i].binding_name[AS_bindings[i].binding_name.length-1].id;
    }
    return as_names;
}


/**
 * Returns a String array containing the parameter data types of all constructors of an agent.
 * <br>agent_name, agent_package and agent_type specify the class of the agent. The method finds
 * out all constructors of the agent and copies the set of parameters for each constructor into
 * a String array which is then returned.
 *
 * @param agent_name name of the agent
 * @param agent_package package containing the class of the agent
 * @param agent_type type of the agent (MobileAgent or StationaryAgent)
 * @return String array containing the parameter sets
 */

public String[] get_agent_parameters(String agent_name, String agent_package, String agent_type) {
    String parametersets[] = new String[1]; //array to contain the parameter sets
    try {
        //get class of selected agent
        String code_base = System.getProperty( GlobalConstants.masa_property_prefix+"agentCodeBase");

        //get class of specified agent
        //Class agent_class = java.rmi.server.RMIClassLoader.loadClass(new java.net.URL(code_base), agent_package + "." + agent_name + agent_type);
            Class agent_class = AgentSystemClassLoader.loadClass( new java.net.URL( code_base), agent_package + "." + agent_name + agent_type);

        //get constructors of the agent
        java.lang.reflect.Constructor constructors[] = agent_class.getConstructors();

        //get types of parameters for each constructor
        int nr_of_parametersets = constructors.length;
        parametersets = new String[nr_of_parametersets]; //nr_of_parametersets is always at least 1
        for (int i=0; i<nr_of_parametersets; i++) {
            Class parameter_types[] = constructors[i].getParameterTypes();
            //create parameter set for one constructor (constructor[i])
            parametersets[i] = " "; //prevent null-Strings in parametersets[], otherwise marshalling-exceptions are generated
                for (int j=0; j<parameter_types.length; j++) {
                    parametersets[i] += parameter_types[j].getName() + " ";
                }
        }
    } catch (Exception e1) {
        if (e1 instanceof java.lang.ClassNotFoundException)
            parametersets[0] = "failed";
        else {
            parametersets[0] = "failed";
            e1.printStackTrace();
        }
    }
    return parametersets;
}


/**
 * Defines a thread that creates a PushConsumer and connects it to the CORBA event service.
 * <br>The method used is the so called "Push method".
 */

class ThreadEventListener implements Runnable {
    org.omg.CosNaming.NamingContext initContext = null;

    ThreadEventListener(org.omg.CosNaming.NamingContext context) {
        initContext = context;
    }

    public void run() {
        try {
            // get event channel
            String channelName = System.getProperty( GlobalConstants.masa_property_prefix+"agent.channel", "AgentInitChannel");
            org.omg.CosNaming.NameComponent ch = new org.omg.CosNaming.NameComponent(channelName,"");
            org.omg.CosNaming.NameComponent[] arr_ch = new org.omg.CosNaming.NameComponent[1];
            arr_ch[0] = ch;
            org.omg.CosEventChannelAdmin.EventChannel eventChannel =
                org.omg.CosEventChannelAdmin.EventChannelHelper.narrow(initContext.resolve(arr_ch));
#ifdef VISIBROKER30
            // Initialize the ORB.
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
            // Initialize the BOA.
            org.omg.CORBA.BOA boa = orb.BOA_init();
#endif
#ifdef ORBACUS311
            // Initialize the ORB.
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( new String[0], new java.util.Properties());
            // Initialize the BOA.
        org.omg.CORBA.BOA boa= ((com.ooc.CORBA.ORB)orb).BOA_init( new String[0], new java.util.Properties());
#endif

            // create PushConsumer
            PushConsumerImpl pushConsumer = new PushConsumerImpl();
```

```
#ifdef VISIBROKER30
                boa.obj_is_ready(pushConsumer);  //Export the newly created object
#endif
#ifdef ORBACUS311
                boa.obj_is_ready(pushConsumer, null);  //Export the newly created object
#endif

                //get ProxyPushSupplier from the CORBA event service
                org.omg.CosEventChannelAdmin.ProxyPushSupplier pushSupplier = eventChannel.for_consumers().obtain_push_supplier();
                //connect the PushConsumer to the ProxyPushSupplier
                pushSupplier.connect_push_consumer(pushConsumer);

                //wait for events
#ifdef VISIBROKER30
                boa.impl_is_ready();
#endif
#ifdef ORBACUS311
                boa.impl_is_ready( null);
#endif

            } catch (Exception el) {
                el.printStackTrace();
            }
        }
    }


    /**
     * Defines the behaviour of the ASManagementAgent´s PushConsumer.
     * <br>Apart from the PushConsumers that clients can register with the ASManagementAgent, this agent
     * has also its own PushConsumer. This one is connected directly to the CORBA event service using the
     * "Push model". Whenever a CORBA event occurs the push() method of the PushConsumer is executed.  In
     * this push() method the push() methods of all the PushConsumers that clients connected to the
     * ASManagementAgent are called one after another. This way the CORBA event is forwarded to the clients
     * that cannot directly contact the CORBA event service.
     */

    class PushConsumerImpl extends _PushConsumerImplBase {

        public PushConsumerImpl()
        {
            System.err.println("\n\nASManagementAgent: PushConsumer erstellt");
        }

        public void push(org.omg.CORBA.Any any)
        {
            int nopc = pushConsumerVector.size();  //get the number of pushConsumers
            org.omg.CosEventComm.PushConsumer tmpPushConsumer = null;
            VectorElement ve = null;
            for (int i=0; i<nopc; i++) {
                try {
                    ve = (VectorElement) pushConsumerVector.elementAt(i);
                    System.err.println("\nASManagementAgent: executing push() for pushConsumer #"+ve.id);
                    tmpPushConsumer = ve.pushConsumer;
                    tmpPushConsumer.push(any);
                } catch (Exception e) {
                    System.err.println("\n\nASManagementagent: push() failed!");
                    if(e instanceof org.omg.CORBA.COMM_FAILURE) { //no communication possible
                        System.err.println("\n\nASManagementAgent: removing pushConsumer #"+ve.id);
                        pushConsumerVector.removeElement(ve);
                    }
                    e.printStackTrace();
                }
            }
        }

        public void disconnect_push_consumer() {
            try {
#ifdef VISIBROKER30
                // Initialize the ORB.
                org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
                // Initialize the BOA.
                org.omg.CORBA.BOA boa = orb.BOA_init();
#endif
#ifdef ORBACUS311
                // Initialize the ORB.
                org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( new String[0], new java.util.Properties());
                // Initialize the BOA.
            org.omg.CORBA.BOA boa= ((com.ooc.CORBA.ORB)orb).BOA_init( new String[0], new java.util.Properties());
#endif

                System.err.println("\n\nDisconnecting push_consumer...\n");
            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }


    /**
     * Datatype for use in pushConsumerVector.
     * <br>This data type is used as a wrapper for the PushConsumers that clients connect to the
     * ASManagementAgent. It contains the PushConsumer itself and the ID number that is assigned
     * to it by connect_push_consumer().
     */

    class VectorElement {
        public org.omg.CosEventComm.PushConsumer pushConsumer = null;
        public int id = 0;

        //Constructor
        public VectorElement(org.omg.CosEventComm.PushConsumer _pushConsumer, int _id) {
            pushConsumer = _pushConsumer;
            id = _id;
        }
    }
}
```

Anhang C

Listing der IDL-Schnittstelle des RegionManagementAgenten

```
// the next two lines are C preprocessor directives
// which are allowed in CORBA IDL
#ifndef _RegionManagementAgent_idl_
#define _RegionManagementAgent_idl_

// this will be a mobile agent
// so I need Migration.idl
#include "Migration.idl"
#include "CosEventComm.idl"

// this module will be converted to
// a java package 'RegionManagementAgent'
module RegionManagementAgent {

  interface RegionManagementAgent : agent::Migration
  {
        //************************
        //    data types
        //************************

        //agentsystem_url is a string containing the url of an agent system,
        //e.g. "http://pcheger14.nm.informatik.uni-muenchen.de:4324"
        typedef string agentsystem_url;

        //agentsystem_name is a string containing the name a an agent system.
        typedef string agentsystem_name;

        //agentname is a string containing the name of an agent
        typedef string agentname;

        //agentname_list is a sequence of agentnames containing a list of names of the agents
        //found in the naming context of an agent system.
        typedef sequence< agentname > agentname_list;

        //************************
        //   methods
        //************************

        //get url of agent system "as"
        agentsystem_url get_AS_URL(in agentsystem_name as);

        //terminate agent system "as"
        void terminate_agent_system(in agentsystem_name as);

        //migrate agent "agent_identity" from agent system "as" to "target"
        void migrate_agent(in string agent_identity, in agentsystem_name target, in agentsystem_name as);

        //list all agents in agent system "as"
        agentname_list list_agent_names(in agentsystem_name as);
  };
};
#endif
```

Anhang D

Listing des RegionManagementAgenten

```
package __MASA_PACKAGE_thisagent;

import __MASA_PACKAGE_agentSystem__(*);
import __MASA_PACKAGE_agent__(*);
import __MASA_PACKAGE_CfMAF__(*);
import __MASA_PACKAGE__(tools.NameWrapper);
import __MASA_PACKAGE__(tools.Debug);
import __MASA_PACKAGE__(tools.GlobalConstants);

import java.io.*;
import java.util.*;

import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosNaming.*;


/**
 * This is the class defining the RegionManagementAgent.
 * <br>The RegionManagementAgent is a mobile agent which primary task it is to provide methods
 * that allow applets to communicate with objects and CORBA services. In particular these are
 * other agent systems, the CORBA Name Service and the CORBA Event Service, which are often not located
 * on the same machines as the applets. The RegionManagementAgent therefore acts as a proxy for these applets.
 *
 *      @see MobileAgent
 *      @author Stefan Gerber
 */

public class RegionManagementAgentMobileAgent extends MobileAgent
    implements RegionManagementAgentOperations {

    transient private org.omg.CosNaming.NamingContext agent_context; // CORBA naming context for agents
    transient private org.omg.CosNaming.NamingContext AS_context;    //CORBA naming context for agent systems


    //constructor
    public RegionManagementAgentMobileAgent() {
        super();
        //get_agent_context();  //doesn't work here, because _initContext = null at this moment
    }

    public void cleanUp() throws Throwable {
        System.err.println("RegionManagementAgentMobileAgent.cleanUp()");
    }

    public void checkSerialization()
        throws CouldNotMigrate {
    }


    /**
     * Calls methods to fetch the naming contexts of agents/agent systems.
     * <br>This method is called, when the agent is started, also after migration.
     */

    public void run() {
        get_agent_context();
        get_agentsystem_context();
    }


    /**
     * Fetches the naming context containing the agents of the different agent systems.
     * <br>The naming context is saved into agent_context. This method is only for internal use.
     */

    void get_agent_context()
    {
        //get context of agents
        try {
            NameComponent comp1 = new NameComponent("Agent", "");
            NameComponent name[] = {comp1};
            agent_context = NamingContextHelper.narrow(_initContext.resolve(name));
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("\nfailed :" + e.toString());
        }

    }


    /**
     * Fetches the naming context containing the different agent systems.
     * <br>The naming context is saved into AS_context. This method is only for internal use.
     */

    void get_agentsystem_context()
    {
        //Get context of the agent systems
        try {
            NameComponent comp1 = new NameComponent("AgentSystemService", "");
            NameComponent comp2 = new NameComponent("4", "");
            NameComponent comp3 = new NameComponent("mnm", "");
            NameComponent name[] = {comp1, comp2, comp3};
            AS_context = NamingContextHelper.narrow(_initContext.resolve(name));
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("failed :" + e.toString());
        }
    }

    /**
     * Returns the AgentSystemService object of an agent system.
     * <br>This method is internally used by migrate_agent() and get_AS_URL()
     *
     * @param as name of agent system
     * @return AgentSystemService object of agent system specified by "as"
     **/
```

```
AgentSystemService get_agentsystemservice(String as) {
    //get_agentsystem_context();

    AgentSystemService ass = null;

    try {
        //resolve AgentSystemService-Object of as
        NameComponent comp1 = new NameComponent(as, "");
        NameComponent name[] = {comp1};
        ass = AgentSystemServiceHelper.narrow(AS_context.resolve(name));
    } catch (Exception e) {
        e.printStackTrace();
    }

    return ass;
}


/**
 * Returns a list of all agents in an agent system.
 * <br>This list is determined using the NamingContext object agent_context.
 *
 * @param as name of the agent system
 * @return String[] containing the list of agent names
 */

public String[] list_agent_names(String as) {
    //get the naming context from the CORBA name service
    //get_agent_context();

    //get name context of agent system "as"
    org.omg.CosNaming.NamingContext agentsystem_context = null;
    try {
        NameComponent comp1 = new NameComponent(as, "");
        NameComponent name[] = {comp1};
        agentsystem_context = NamingContextHelper.narrow(agent_context.resolve(name));
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println("\n\nfailed :" + e.toString());
    }

    BindingListHolder bl_holder = new BindingListHolder();
    BindingIteratorHolder bi_holder = new BindingIteratorHolder();
    agentsystem_context.list(1000000, bl_holder, bi_holder);
    Binding[] agent_bindings = bl_holder.value;

    System.err.println("list_agent_names: agent_bindings.length = "+agent_bindings.length);
    String agent_names[] = new String[agent_bindings.length];
    for (int i=0; i<agent_bindings.length; i++) {
        agent_names[i] = (String) agent_bindings[i].binding_name[agent_bindings[i].binding_name.length-1].id;
    }
    return agent_names;
}


/**
 * Transfers an agent to another agent system.
 * <br>The agent specified by "agent_identity"  is transferred to the
 * agent system with the name provided by the parameter "target". The
 * parameter "as" is the name of the source agent system.
 *
 * @param agent_identity name of agent
 * @param target target agent system
 * @param as source agent system
 */

public void migrate_agent(String agent_identity, String target, String as) {
    AgentSystemService ass = null;
        try {
            ass = get_agentsystemservice(as);

            NameWrapper nw= new NameWrapper(agent_identity);

            // connect to agent to transfer
            org.omg.CORBA.Object obj = ass.connectToAgent(nw._name);
            Migration agent = MigrationHelper.narrow(obj);
            // connect to destination agent system
            agent.migrateToAgentSystem("AgentSystemService/4/mnm/"+ target);
            System.err.println("\n\ntransferAgent: agent "+ agent_identity + " transferred to "+ target);
        }
        catch(Exception e1){
            if (e1 instanceof org.omg.CORBA.BAD_PARAM) { }
            else {
                e1.printStackTrace();
                System.err.println("transferAgent failed: "+e1.toString()+"\n"+e1.getMessage()+"\n");
            }
        }
    }


    /**
     * Returns a String that contains the URL of an agent system.
     *
     * @param as name of the agent system
     * @return String containing the URL of the agent system
     */

    public String get_AS_URL(String as) {
        AgentSystemService ass = get_agentsystemservice(as);
        return ass.getURL();
    }



  /**
   * Terminates an agent system.
   * <br>This method simply gets a reference to the AgentSystemService instance of agent system "as"
   * and calls its terminate_agent_system() method.
   *
   * @param as name of the agent system
   */

  public void terminate_agent_system(String as) {
    try {
      //connect to agent system specified by "as"
      AgentSystemService ass = get_agentsystemservice(as);
      //terminate agent system
      ass.terminate_agent_system();
    } catch (Exception e) {
      e.printStackTrace();
      System.err.println("\nRegionManagementAgent: terminate agent system "+as+" failed!");
      if (e instanceof TerminateFailed)
        System.err.println(e.toString()+"\n"+e.getMessage());
    }
  }

}
```

Anhang E

Listing von CommonAppletHelpers


```
package __MASA_PACKAGE_agentSystem;

import __MASA_PACKAGE_agent__(AgentApplet);
import __MASA_PACKAGE__(tools.NameWrapper);
import __MASA_PACKAGE_CfMAF__(AgentStatus);


import java.awt.*;
import java.applet.*;
import java.util.*;


import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.text.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.JOptionPane;

import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosNaming.*;



/**
 * This class contains some methods which are used by both AgentSystemApplet and ASManagementAgentApplet.
 * <br>All methods are declared public static, so they can be called from every other class without even
 * creating an instance of CommonAppletHelpers.
 *
 * @author Stefan Gerber
 * @author extended by Harald Roelle
 * @see AgentSystemApplet
 * @see de.unimuenchen.informatik.mnm.masa.agent.asmanagementagent.ASManagementAgentApplet
 */

public class CommonAppletHelpers {

    /**
     * Returns the URL of an agent specified by "identity" on the agent system "host".
     * <br>The URL is resolved using the Webserveragent by sending a request for "identity.url" to it.
     *
     * @param identity name of agent
     * @param host URL of agent system
     * @return URL of the agent
     */

public static java.net.URL getAgentURL(String identity, String host)
    throws java.net.MalformedURLException, java.io.IOException {
    System.err.println("\n\nCommonAppletHelpers.getAgentURL: host = "+host);
    // create URL of the webserver-agent and url request:
    // e.g.: http://sunhegering2:4300/FOO.url
    // if identity is FOO.
    String urlname= new String(host+"/"+identity+".url");

    // send the request
    java.net.URL urlIOR= new java.net.URL(urlname);
    java.net.URLConnection con= urlIOR.openConnection();

    // getting the answer
    java.io.BufferedReader in = new java.io.BufferedReader(new java.io.InputStreamReader(con.getInputStream()));
    String inputLine;
    String urlstring=new String();

    while ((inputLine = in.readLine()) != null)
        urlstring=urlstring.concat(inputLine);
    in.close();
    System.err.println("CommonAppletHelpers.getAgentURL: urlstring = "+urlstring);
    return (new java.net.URL(urlstring));
}


/**
 * Shows a ComboBox from which the user can select the target agent system for an agent transfer.
 * <br>The possible alternatives are defined by the Vector as_names. "parentComponent" is needed
 * for JOptionPane.showInputDialog().
 *
 * @param parentComponent parent applet
 * @param agent_identity name of agent to be transferred
 * @param as_names vector containing the names of possible target agent systems
 * @return String containing the name of the target agent system
 */

public static String show_migrate_menu(Component parentComponent, String agent_identity, java.util.Vector as_names) {

    //specify the size str_as_names will have
    //Both "global" and "voyager" are no agent systems a MASA agent can be transferred to.
    int size = 0;
    if (as_names.contains("voyager")) {
        size = as_names.size() - 2; //-2 because of "global" and "voyager"
    }
    else {
        size = as_names.size() - 1; //-1 because of "global"
    }

    String[] str_as_names = new String[size];
    //copy names of all agent systems except "global" and "voyager"
    int j =0;
    String n;
    for (int i = 0; i < as_names.size(); i++) {
        n = (String) as_names.elementAt(i);
        if (!((n.equals("global")) || (n.equals("voyager")))) {
            str_as_names[j++] = n;
        }
    }

    String target = (String) JOptionPane.showInputDialog(parentComponent,
                                                     "Migrating agent \""+ agent_identity +
                                                     "\"\n" + "Select the target agent system",
```

```
                                                            "Migrate agent dialog",
                                                            JOptionPane.QUESTION_MESSAGE,
                                                            null,
                                                            str_as_names,
                                                            str_as_names[0]);

        return(target);
    }


    /**
     * Retrieve a list of implemented agents from the Webserver agent
     * <br>The file "ImplementedAgents.txt" contains a list of agents that are ready to load. It
     * provides the name, the package and the type (mobile/stationary) of the agents. It´s location
     * is defined by the MASA properties of the agent system, therefore it is read by
     * the webserver agent and provided via HTTP request.
     *
     * @see de.unimuenchen.informatik.mnm.masa.agent.webserver.WebserverStationaryAgent
     */

    public static String retrieve_implemented_agents_file(String host) {
        //prepare the request for the list of implemented agents
        String urlname = new String(host+"/"+"ImplementedAgents");

        java.net.URL urlIOR = null;
        java.net.URLConnection con = null;
        String string_file = new String();  //will contain the answer of the webserver
        try {
            // send the request
            urlIOR= new java.net.URL(urlname);
            con= urlIOR.openConnection();

            // getting the answer
            java.io.BufferedReader in = new java.io.BufferedReader(new java.io.InputStreamReader(con.getInputStream()));
            String inputLine;

            //put the answer into string_file
            while ((inputLine = in.readLine()) != null)
                string_file = string_file.concat(inputLine)+ " ";
            in.close();
        } catch (Exception e1) {
                System.err.println( "get_impl_agents urlname="+urlname);
            e1.printStackTrace();
        }
        return string_file;
    }

        public static org.omg.CORBA.ORB initialize_orb_for_push_consumer()
        {
#ifdef VISIBROKER30
                org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
#endif
#ifdef ORBACUS311
                java.util.Properties props = new java.util.Properties();
                props.put( "ooc.orb.conc_model",              "threaded");
                props.put( "ooc.boa.conc_model",              "thread_per_request");
                props.put( "org.omg.CORBA.ORBClass",          "com.ooc.CORBA.ORB");
                props.put( "org.omg.CORBA.ORBSingletonClass", "com.ooc.CORBA.ORBSingleton");
                org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( new String[0], props);
                // with Orbacus we still need the BOA for the event channel, sorry
                org.omg.CORBA.BOA boa = ((com.ooc.CORBA.ORB)orb).BOA_init( new String[0], props);
                ((com.ooc.CORBA.BOA)boa).init_servers();
#endif

                return orb;
        }

}
```

Anhang F

Listing des AgentSystemApplets

```
package __MASA_PACKAGE_agent__(AgentSystem);

import __MASA_PACKAGE_agentSystem__(*);
import __MASA_PACKAGE_agent__(ASManagementAgent.*);
import __MASA_PACKAGE_agent__(AgentApplet);
import __MASA_PACKAGE_agent__(Migration);
import __MASA_PACKAGE_agent__(MigrationHelper);
import __MASA_PACKAGE__(tools.NameWrapper);
import __MASA_PACKAGE_CfMAF__(*);

import java.awt.*;
import java.applet.*;
import java.util.*;
import java.io.*;

import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.text.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.event.*;
import com.sun.java.swing.JOptionPane;

import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosNaming.*;


/**
 *      The applet of the agent system. This applet is loaded,
 *      when the webserver agent is contacted via http the first
 *      time.
 *
 *      <br>(See Mobile Agent Facility (MAF)
 *       of Open Management Group (OMG))
 *
 *      @author Stefan Gerber
 *      @see de.unimuenchen.informatik.mnm.masa.agent.webserver.WebserverStationaryAgent
 */

public class AgentSystemApplet extends AgentApplet
{
    /*
     * CORBA reference to the agent system
     */
    private AgentSystemService _ass;

    private ASManagementAgent _asm;

    private JButton _bt_create_agent;
    private JButton _bt_transferAgent;
    private JButton _bt_resume_agent;
    private JButton _bt_suspend_agent;
    private JButton _bt_terminate_agent;
    private JButton _bt_get_agent_system_info;
    private JButton _bt_terminate_agent_system;
    private JButton _bt_readAgentFromFile;
    private JButton _bt_writeAgentToFile;
    private JButton _bt_showwebpage;
    private JButton _bt_update_list;
    private JButton _bt_update_impl_agents;

    private JTextArea _ta_output_agent;
    private JTextArea _ta_output_agent_system;
    private JTextArea _ta_output_create_agent;

    private JTextField _tf_file;

    private JComboBox _ch_create_exclusive;
    private JComboBox _ch_impl_agents;
    private JComboBox _ch_constructors = null;

    private JLabel _label_exclusive;
    private JLabel _label_identity;
    private JLabel _label_file;
    private JLabel _label_agentTable;

    private AppletTableModel myModel;
    private JTable agentTable;
    private JScrollPane tableScrollPane;
    private ListSelectionModel listSelectionModel;
    private String agent_identity = null;
    private java.awt.Component parentComponent;

    private String impl_agent_list[];     //list of implemented agents generated by get_impl_agents
    private String impl_package_list[];   //other data concerning implemented agents list
    private String impl_type_list[];      // ""    ""    ""
    private ActionListener actionlistener_ch_impl_agents;

    private JLabel constructors;
    private String[] constructor_list;

    private ParameterTableModel paramTableModel;
    private JTable paramTable;
    private JScrollPane paramScrollPane;

    private String agent_system_name;
    private Vector[] param_matrix;
    private PushConsumerImpl pushConsumer = null;



    /**
  *      Initialize this applet and connect to agent system.
  *      <br>Sets and displays all JFC components and calls various methods to get initial information
  *      about the agent system and it´s agents (e.g. list of running agents, list of implemented agents
  *      and their parameters)
  *      <br>Connects to the agent system and to the ASManagementAgent
  *
```

```
*      @see AgentSystemService
*      @see ASManagementAgent
*/

 public void init()
 {

     // Connect to appropriate agent system
     try {
         org.omg.CORBA.Object obj = initCORBA("AgentSystemService");
         _ass = AgentSystemServiceHelper.narrow(obj);
     }
     catch(java.net.MalformedURLException e) {//error
         _ta_output_agent_system.append(e.toString()+"\n");
     }
     catch(java.io.IOException ioe) {//error
         _ta_output_agent_system.append(ioe.toString()+"\n");
     }


     // Connect to ASManagement Agent
     try {
         org.omg.CORBA.Object objekt = initCORBA("ASManagementAgent");
         _asm = ASManagementAgentHelper.narrow(objekt);
     }
     catch(java.net.MalformedURLException e) {
         System.err.println(e.toString()+"\n");
         e.printStackTrace();
     }
     catch(java.io.IOException ioe) {//error
         System.err.println(ioe.toString()+"\n");
         ioe.printStackTrace();
     }

     // test if ASManagementAgent has been found
     // if not, then display warning message
     if (_asm == null) {
         JOptionPane.showMessageDialog(null, "ASManagementAgent could not be found.\nPlease load it and restart the applet.");
     }
     else {
         //Create tabbed Panel
         JTabbedPane tabbedPane = new JTabbedPane();

         // Set layout to GridBag
         GridBagLayout gridBag = new GridBagLayout();
         getContentPane().setLayout(gridBag);
         GridBagConstraints c_root = new GridBagConstraints();


         /**********************************************************
          *******  Panel for "Manage Agents"    *********************
          **********************************************************/

         //Create Layout Manager for "Manage Agents" Part
         GridBagLayout manage_agents_gridBag = new GridBagLayout();
         GridBagConstraints c1 = new GridBagConstraints();

         // Create Panel for "Manage agents" part
         JPanel manage_agents_panel = new JPanel(manage_agents_gridBag);

         //Create JFC components and arrange them
         //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

         c1.fill = GridBagConstraints.HORIZONTAL;
         c1.gridwidth = GridBagConstraints.RELATIVE;
         c1.gridx = 0;
         c1.weightx = 1;
         c1.weighty = 2.5;
         //c1.insets = new Insets(8,0,0,0);
         //c1.insets.top = 8;
         c1.insets = new Insets(8,0,0,0);
         c1.gridwidth = GridBagConstraints.REMAINDER;

         _label_agentTable = new JLabel("Agents on this Agent System");
         manage_agents_gridBag.setConstraints(_label_agentTable, c1);
         manage_agents_panel.add(_label_agentTable);

         c1.insets.top = 0;
         c1.weighty = 2.5;

         myModel = new AppletTableModel();
         agentTable = new JTable(myModel);
         myModel.addTableModelListener(agentTable);
         agentTable.setPreferredScrollableViewportSize(new Dimension(250,60));
         tableScrollPane = new JScrollPane(agentTable, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
         tableScrollPane.setBorder(BorderFactory.createTitledBorder("Agents on this Agent System"));
         tableScrollPane.setMinimumSize(new Dimension(tabbedPane.getWidth(), 130));
         tableScrollPane.setPreferredSize(new Dimension(tabbedPane.getWidth(), 130));

         // Set selection model for agentTable
         listSelectionModel = agentTable.getSelectionModel();
         listSelectionModel.addListSelectionListener(new ListSelectionHandler());
         listSelectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

         //get list of running agents and their status and display in agentTable
         list_agent_names();

         manage_agents_gridBag.setConstraints(tableScrollPane, c1);
         manage_agents_panel.add(tableScrollPane);

         //create various buttons for managing the agents
         c1.insets = new Insets(8,25,0,25);
         c1.gridx = 0;
         c1.weighty = 0;
         c1.gridwidth = GridBagConstraints.RELATIVE;

         _bt_suspend_agent = new JButton("suspend agent");
         _bt_suspend_agent.setMnemonic('s');
         _bt_suspend_agent.setToolTipText("suspend the selected agent");
         _bt_suspend_agent.setActionCommand("button");
         _bt_suspend_agent.setBackground(new Color(12632256));
         manage_agents_gridBag.setConstraints(_bt_suspend_agent, c1);
         manage_agents_panel.add(_bt_suspend_agent);

         c1.gridx = GridBagConstraints.RELATIVE;
         c1.gridwidth = GridBagConstraints.REMAINDER;

         _bt_resume_agent = new JButton("resume agent");
         _bt_resume_agent.setMnemonic('u');
         _bt_resume_agent.setToolTipText("resume the selected agent");
         _bt_resume_agent.setActionCommand("button");
         _bt_resume_agent.setBackground(new Color(12632256));
         manage_agents_gridBag.setConstraints(_bt_resume_agent, c1);
         manage_agents_panel.add(_bt_resume_agent);

         c1.gridx = 0;
         c1.gridwidth = GridBagConstraints.RELATIVE;
         c1.insets.top = 0;

         _bt_terminate_agent = new JButton("terminate agent");
         _bt_terminate_agent.setMnemonic('t');
         _bt_terminate_agent.setToolTipText("terminate the selected agent");
         _bt_terminate_agent.setActionCommand("button");
         _bt_terminate_agent.setBackground(new Color(12632256));
         manage_agents_gridBag.setConstraints(_bt_terminate_agent, c1);
         manage_agents_panel.add(_bt_terminate_agent);
```

```
c1.gridx = GridBagConstraints.RELATIVE;
c1.gridwidth = GridBagConstraints.REMAINDER;

_bt_showwebpage = new JButton("show agent's webpage");
_bt_showwebpage.setActionCommand("button");
_bt_showwebpage.setMnemonic('w');
_bt_showwebpage.setToolTipText("show the webpage of the selected agent");
_bt_showwebpage.setBackground(new Color(12632256));
manage_agents_gridBag.setConstraints(_bt_showwebpage, c1);
manage_agents_panel.add(_bt_showwebpage);

c1.gridx = 0;
c1.gridwidth = GridBagConstraints.RELATIVE;

_bt_transferAgent = new JButton("transfer Agent");
_bt_transferAgent.setMnemonic('f');
_bt_transferAgent.setToolTipText("migrate the selected agent to a different agent system");
_bt_transferAgent.setActionCommand("button");
_bt_transferAgent.setBackground(new Color(12632256));
manage_agents_gridBag.setConstraints(_bt_transferAgent, c1);
manage_agents_panel.add(_bt_transferAgent);

c1.gridx = GridBagConstraints.RELATIVE;
c1.gridwidth = GridBagConstraints.REMAINDER;

//Create "update button"
_bt_update_list = new JButton("update manually");
_bt_update_list.setMnemonic('u');
_bt_update_list.setToolTipText("update the table of loaded agents manually");
_bt_update_list.setActionCommand("button");
_bt_update_list.setBackground(new Color(12632256));
_bt_update_list.setMargin(new Insets(2,2,2,2));
manage_agents_gridBag.setConstraints(_bt_update_list, c1);
manage_agents_panel.add(_bt_update_list);


//insert a separator
c1.insets = new Insets(26,0,0,0);

JSeparator sep2 = new JSeparator();
manage_agents_gridBag.setConstraints(sep2, c1);
manage_agents_panel.add(sep2);

//create the textfield
//the user specifies the input/output file with it
c1.insets.top = 26;

_label_file = new JLabel("File");
manage_agents_gridBag.setConstraints(_label_file, c1);
manage_agents_panel.add(_label_file);

c1.insets.top = 0;

_tf_file = new JTextField();
manage_agents_gridBag.setConstraints(_tf_file, c1);
manage_agents_panel.add(_tf_file);

//create buttons for reading/writing agents to files
c1.insets = new Insets(8,25,0,25);
c1.gridx = 0;
c1.gridwidth = GridBagConstraints.RELATIVE;

_bt_readAgentFromFile = new JButton("read agent from file");
_bt_readAgentFromFile.setMnemonic('a');
_bt_readAgentFromFile.setToolTipText("read an agent from the specified file");
_bt_readAgentFromFile.setActionCommand("button");
_bt_readAgentFromFile.setBackground(new Color(12632256));
manage_agents_gridBag.setConstraints(_bt_readAgentFromFile, c1);
manage_agents_panel.add(_bt_readAgentFromFile);

c1.gridwidth = GridBagConstraints.REMAINDER;
c1.gridx = GridBagConstraints.RELATIVE;

_bt_writeAgentToFile = new JButton("write agent to file");
_bt_writeAgentToFile.setMnemonic('i');
_bt_writeAgentToFile.setToolTipText("write selected agent to the specified file");
_bt_writeAgentToFile.setActionCommand("button");
_bt_writeAgentToFile.setBackground(new Color(12632256));
manage_agents_gridBag.setConstraints(_bt_writeAgentToFile, c1);
manage_agents_panel.add(_bt_writeAgentToFile);

//create textarea for outputs of various methods in this panel
c1.gridx = 0;
c1.insets = new Insets(16,0,0,0);
c1.gridheight = GridBagConstraints.REMAINDER;
c1.weighty = c1.weightx = 2.0;
c1.fill = GridBagConstraints.BOTH;

_ta_output_agent = new com.sun.java.swing.JTextArea();
_ta_output_agent.setBounds(72,600,624,168);
_ta_output_agent.setEditable(false);
_ta_output_agent.setMargin(new Insets(5,5,5,5));

Color color = new Color(188,210,238);
_ta_output_agent.setBackground(color);
JScrollPane scrollPane_agent = new JScrollPane(_ta_output_agent, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
scrollPane_agent.setBorder(BorderFactory.createTitledBorder("Output:"));
scrollPane_agent.setMinimumSize(new Dimension(tabbedPane.getWidth(), 110));
scrollPane_agent.setPreferredSize(new Dimension(tabbedPane.getWidth(), 110));

manage_agents_gridBag.setConstraints(scrollPane_agent, c1);
manage_agents_panel.add(scrollPane_agent);

//add manage_agents_panel to the tabbed panel
tabbedPane.addTab("Manage agents", null, manage_agents_panel, "manage the agents of this agent system");
tabbedPane.setSelectedIndex(0);


/***********************************************************
 *******  Panel for "Create Agent"     ********************
 ***********************************************************/

//Create Layout Manager for "Create Agent" Part
GridBagLayout create_agent_gridBag = new GridBagLayout();
GridBagConstraints c2 = new GridBagConstraints();

// Create Panel for "Create agent" part
JPanel create_agent_panel = new JPanel(create_agent_gridBag);

//Create JFC components and arrange them
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

c2.fill = GridBagConstraints.HORIZONTAL;
c2.gridwidth = GridBagConstraints.REMAINDER;
c2.gridx = 0;
c2.weightx = 1;
c2.insets = new Insets(8,0,0,0);

_label_identity = new JLabel("Choose agent");
_label_identity.setBackground(new Color(8454016));
create_agent_gridBag.setConstraints(_label_identity, c2);
create_agent_panel.add(_label_identity);

c2.insets.top = 0;

// get list of implemented agents from webserver and put it in impl_agent_list
get_impl_agents();
```

```
//create combobox which displays the implemented agents
_ch_impl_agents = new JComboBox(impl_agent_list);
_ch_impl_agents.setSelectedIndex(0);
_ch_impl_agents.setEditable(false);
create_agent_gridBag.setConstraints(_ch_impl_agents, c2);
create_agent_panel.add(_ch_impl_agents);


//define an actionlistener for _ch_impl_agents
actionlistener_ch_impl_agents = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            get_agent_parameters();
        }
    };

//if the user selects an agent from the list it´s parameters are obtained immediately
_ch_impl_agents.addActionListener(actionlistener_ch_impl_agents);


//create combobox which determines if the new agent will be exclusive
c2.insets.top = 16;
c2.gridwidth = GridBagConstraints.REMAINDER;
c2.gridx = GridBagConstraints.RELATIVE;;

_label_exclusive = new JLabel("Exclusive ");
_label_exclusive.setBackground(new Color(8454016));
create_agent_gridBag.setConstraints(_label_exclusive, c2);
create_agent_panel.add(_label_exclusive);

c2.insets = new Insets(0,0,0,300);

_ch_create_exclusive = new JComboBox();
_ch_create_exclusive.addItem("false");
_ch_create_exclusive.addItem("true");
_ch_create_exclusive.setSelectedIndex(0);
create_agent_gridBag.setConstraints(_ch_create_exclusive, c2);
create_agent_panel.add(_ch_create_exclusive);

//create combobox which selects between the different constructors for the new agent
c2.insets = new Insets(16,0,0,0);
c2.gridx = 0;

constructors = new JLabel("Create Agent with...");
constructors.setBackground(new Color(8454016));
create_agent_gridBag.setConstraints(constructors, c2);
create_agent_panel.add(constructors);

c2.insets = new Insets(0,0,0,300);

//get list of constructors and put it into constructor_list
get_agent_parameters();

_ch_constructors = new JComboBox(constructor_list);
_ch_constructors.setSelectedIndex(0);
_ch_constructors.setEditable(false);
create_agent_gridBag.setConstraints(_ch_constructors, c2);
create_agent_panel.add(_ch_constructors);


//if a constructor is chosen then the table shows the different types of parameters
_ch_constructors.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        show_agent_parameters(_ch_constructors.getSelectedIndex());
    }
}
                                 );
```

```
//create table showing the different parameters of a constructor
//the user also types in the values for these parameters using this table
c2.insets = new Insets(16,0,0,0);

paramTableModel = new ParameterTableModel();
paramTable = new JTable(paramTableModel);
paramTableModel.addTableModelListener(paramTable);
paramTable.setPreferredScrollableViewportSize(new Dimension(250,60));
paramScrollPane = new JScrollPane(paramTable, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
paramScrollPane.setBorder(BorderFactory.createTitledBorder("Parameters for this constructor"));
paramScrollPane.setMinimumSize(new Dimension(tabbedPane.getWidth(), 100));
paramScrollPane.setPreferredSize(new Dimension(tabbedPane.getWidth(), 100));

create_agent_gridBag.setConstraints(paramScrollPane, c2);
create_agent_panel.add(paramScrollPane);

//show list of parameters for currently selected implemented agent
show_agent_parameters(0);

//create buttons
c2.gridx = 0;
c2.gridwidth = GridBagConstraints.RELATIVE;
c2.insets = new Insets(20,25,0,25);

_bt_create_agent = new JButton("create agent");
_bt_create_agent.setMnemonic('c');
_bt_create_agent.setToolTipText("load the selected Agent with the specified options");
_bt_create_agent.setActionCommand("button");
_bt_create_agent.setBackground(new Color(12632256));
create_agent_gridBag.setConstraints(_bt_create_agent, c2);
create_agent_panel.add(_bt_create_agent);

c2.gridwidth = GridBagConstraints.REMAINDER;
c2.gridx = GridBagConstraints.RELATIVE;

_bt_update_impl_agents = new JButton("update list");
_bt_update_impl_agents.setMnemonic('u');
_bt_update_impl_agents.setToolTipText("update the list of loadable agents");
_bt_update_impl_agents.setActionCommand("button");
_bt_update_impl_agents.setBackground(new Color(12632256));
create_agent_gridBag.setConstraints(_bt_update_impl_agents, c2);
create_agent_panel.add(_bt_update_impl_agents);

//create textarea for output of the methods of this panel
c2.gridx = 0;
c2.gridheight = 6;
c2.weighty = c2.weightx = 2.5;
c2.fill = GridBagConstraints.BOTH;
c2.insets = new Insets(20,0,0,0);

_ta_output_create_agent = new com.sun.java.swing.JTextArea();
_ta_output_create_agent.setEditable(false);
_ta_output_create_agent.setMargin(new Insets(5,5,5,5));


_ta_output_create_agent.setBackground(color);
JScrollPane scrollPane_create_agent = new JScrollPane(_ta_output_create_agent, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
scrollPane_create_agent.setBorder(BorderFactory.createTitledBorder("Output:"));

create_agent_gridBag.setConstraints(scrollPane_create_agent, c2);
create_agent_panel.add(scrollPane_create_agent);

//add panel to tabbed pane
tabbedPane.addTab("Create agent", null, create_agent_panel, "load a new agent into this agent system");


/**********************************************************
 *******  Panel for "Manage agent system"  ****************
```

```
 ***********************************************************/

//Create Layout Manager for "Manage agent system" Part
GridBagLayout agent_system_gridBag = new GridBagLayout();

//Create Panel for "Manage agent system" Part
JPanel agent_system_panel = new JPanel(agent_system_gridBag);
GridBagConstraints c3 = new GridBagConstraints();


//create buttons
c3.gridx = 0;
c3.gridwidth = GridBagConstraints.REMAINDER;
c3.fill = GridBagConstraints.HORIZONTAL;
c3.weightx = 1;
c3.insets = new Insets(20,150,0,150);

_bt_get_agent_system_info = new JButton("get agent system info");
_bt_get_agent_system_info.setMnemonic('i');
_bt_get_agent_system_info.setToolTipText("show information about this agent system");
_bt_get_agent_system_info.setActionCommand("button");
_bt_get_agent_system_info.setBackground(new Color(12632256));
agent_system_gridBag.setConstraints(_bt_get_agent_system_info, c3);
agent_system_panel.add(_bt_get_agent_system_info);

c3.insets.top = 0;

_bt_terminate_agent_system = new JButton("terminate agent system");
_bt_terminate_agent_system.setMnemonic('t');
_bt_terminate_agent_system.setToolTipText("shut down this agent system");
_bt_terminate_agent_system.setActionCommand("button");
_bt_terminate_agent_system.setBackground(new Color(12632256));
agent_system_gridBag.setConstraints(_bt_terminate_agent_system, c3);
agent_system_panel.add(_bt_terminate_agent_system);

//create textarea for output of the methods of this panel
c3.gridx = 0;
c3.gridwidth = GridBagConstraints.REMAINDER;
c3.insets = new Insets(20,0,0,0);
c3.gridheight = 4;
c3.weighty = 2;
c3.fill = GridBagConstraints.BOTH;

_ta_output_agent_system = new com.sun.java.swing.JTextArea();
_ta_output_agent_system.setBounds(72,600,624,168);
_ta_output_agent_system.setEditable(false);
_ta_output_agent_system.setMargin(new Insets(5,5,5,5));
_ta_output_agent_system.setBackground(color);
JScrollPane scrollPane_agent_system = new JScrollPane(_ta_output_agent_system, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
scrollPane_agent_system.setBorder(BorderFactory.createTitledBorder("Output:"));

agent_system_gridBag.setConstraints(scrollPane_agent_system, c3);
agent_system_panel.add(scrollPane_agent_system);

//add panel to tabbed pane
tabbedPane.addTab("Agent System", null, agent_system_panel, "manage the agent system");



//add tabbed Pane to Applet panel
c_root.fill = GridBagConstraints.BOTH;
c_root.gridx = 0;
c_root.gridwidth = GridBagConstraints.REMAINDER;
c_root.gridheight = GridBagConstraints.REMAINDER;
c_root.weightx = c_root.weighty = 1.0;
gridBag.setConstraints(tabbedPane, c_root);

getContentPane().add(tabbedPane);
```

```
        //set parentComponent variable to point to this applet
        //needed to display transfer-agent-dialogue from within ASManagementAgent
        parentComponent = this;

        //get the name of the agent system this applet belongs to
        String as_url = _ass.getURL();
        agent_system_name = as_url.substring(as_url.lastIndexOf("/")+1, as_url.lastIndexOf(":"));

        //create pushConsumer and connect it to the ASManagementAgent
        //needed to react to events from CORBA event service
        create_pushConsumer();

        //define the action listeners for the various buttons created above
        this.initEventHandler();
    }
}


/**
*  Implement all actions that take place when a certain button is pushed.
*  <br>This method defines the action listeners for the various buttons of this applet and
*  the mouse listener for the table of loaded agents
*/

public void initEventHandler() {

    /*
     *  action listener for create_agent button
     */

    //clear any selection in the table of loaded agents for safety reasons
    agentTable.clearSelection();

    _bt_create_agent.addActionListener(new java.awt.event.ActionListener() {

        public void actionPerformed(java.awt.event.ActionEvent e) {
            try {
                // test if the agent is already running in this agent system
                // if "yes" then show a warning message
                if ( myModel.isElement((String) _ch_impl_agents.getSelectedItem()) ) {
                    int answer = JOptionPane.showConfirmDialog(null, "An agent with this name is already\nrunning on this agent system!\nDo you really want to start another instance?", "Warning!", JOptionPane.YES_NO_OPTION);
                    //if user cancels then return without any further action, else proceed
                    if (answer == JOptionPane.NO_OPTION) {
                        return;
                    }
                }

                //create the parameters for the create_agent() call
                java.io.ByteArrayOutputStream agentBytes = new java.io.ByteArrayOutputStream();
                java.io.ObjectOutputStream outputAgent = new java.io.ObjectOutputStream(agentBytes);

                //get the index of the item currently selected in the combobox with impl. agents
                int selected_index = _ch_impl_agents.getSelectedIndex();

                //prepare outputAgent with data about the agent to be loaded
                outputAgent.writeObject(impl_package_list[selected_index]);
                outputAgent.writeObject(impl_type_list[selected_index]);
#ifdef VISIBROKER30
                outputAgent.writeObject("_tie_");
                outputAgent.writeObject("Operations");
#endif
#ifdef ORBACUS311
                outputAgent.writeObject("_");
                outputAgent.writeObject("ImplBase_tie");
#endif
                outputAgent.writeObject(new Boolean((String) _ch_create_exclusive.getSelectedItem()));
                outputAgent.writeObject("");
```

```
                    java.io.ByteArrayOutputStream constructorArgs = new java.io.ByteArrayOutputStream();
                    java.io.ObjectOutputStream constructorObject = new java.io.ObjectOutputStream(constructorArgs);

                    //get number of rows of parameter table i.e. the number of constructor parameters
                    int nor = paramTableModel.getRowCount();

                    //fetch the parameter data types and their values provided by the user via paramTable
                    String paramtype = null;
                    for (int i=0; i<nor; i++) {
                        //get parameter type
                        paramtype = (String) paramTableModel.getValueAt(i, 0);

                        //create appropriate object and copy it into the ByteArray
                        if (paramtype.equals("java.lang.String")) {
                            constructorObject.writeObject((String) paramTableModel.getValueAt(i, 1));
                        }
                        else if (paramtype.equals("java.lang.Integer")) {
                            constructorObject.writeObject(java.lang.Integer.valueOf((String) paramTableModel.getValueAt(i, 1)));
                        }
                        else if (paramtype.equals("java.lang.Float")) {
                            constructorObject.writeObject(java.lang.Float.valueOf((String) paramTableModel.getValueAt(i, 1)));
                        }
                        else if (paramtype.equals("java.lang.Double")) {
                            constructorObject.writeObject(java.lang.Double.valueOf((String) paramTableModel.getValueAt(i, 1)));
                        }
                        //(roelle): IMHO this was missing
                        else if (paramtype.equals("java.lang.Boolean")) {
                            constructorObject.writeObject(java.lang.Boolean.valueOf((String) paramTableModel.getValueAt(i, 1)));
                        }

                    }

                    // agent name is equal class name, so no information about classes needed
                    ClassName[] className = new ClassName[0];

                    //fetch the name of the agent to be created
                    NameWrapper nw = new NameWrapper((String) _ch_impl_agents.getSelectedItem());

                    //create the profile of the agent and it´s properties
                    AgentProfile ap = new AgentProfile();
                    org.omg.CORBA.Any props[] = new org.omg.CORBA.Any[1];
                    org.omg.CORBA.Any any = getORB().create_any();
#ifdef VISIBROKER30
                    any.insert_wstring("Property");
#endif
#ifdef ORBACUS311
                    any.insert_string("Property");
#endif
                    props[0]=any;
                    ap.language_id=1;
                    ap.serialization=1;
                    ap.properties= props;
                    ap.agent_system_description= "Agent System of MNM-Team";

                    // create the agent with the information collected and prepared above
                    Name res = _ass.create_agent(nw._name,
                                                       ap,
                                                       agentBytes.toByteArray(),
                                                       new String("default"),
                                                       constructorArgs.toByteArray(),
                                                       className,
                                                       "",
                                                       (MAFAgentSystem)_ass);

                    //print a status message to the output window
                    _ta_output_create_agent.append("create_agent(): agent "+_ch_impl_agents.getSelectedItem()+
                                                   " created.\n");
                }
                catch(Exception e0){ //error
                    e0.printStackTrace();
                    _ta_output_create_agent.append("create_agent(): create agent "+_ch_impl_agents.getSelectedItem()+
                                                   " failed: "+ e0.toString()+"\n"+ (e0.getMessage() == null ? "" : e0.getMessage()+"\n"));
                }
            }
        }
                                           );

        /*
         *  action listener for transferAgent button
         */
        _bt_transferAgent.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                try {
                    // get list of all active agent systems from the ASManagementAgent
                    String agentsystems[] = _asm.identify_agentsystems();

                    //create a Vector containing the names of the active agent systems
                    java.util.Vector as_names = new java.util.Vector(0,1);
                    int n_o_t = agentsystems.length;
                    for (int i=0; i<n_o_t; i++) {
                        as_names.addElement(agentsystems[i]);
                    }

                    //Show dialogue to get target agent system
                    String target = CommonAppletHelpers.show_migrate_menu(parentComponent, agent_identity, as_names);

                    if (target == null) { //if user pressed the cancel-button
                        //clear selection in agentTable and disable buttons for safety reasons
                        clear_selection();
                        return;
                    }

                    // connect to agent to transfer
                    NameWrapper nw = new NameWrapper(agent_identity);
                    org.omg.CORBA.Object obj = _ass.connectToAgent(nw._name);
                    Migration agent = MigrationHelper.narrow(obj);

                    // transfer agent to destination agent system
                    agent.migrateToAgentSystem("AgentSystemService/4/mnm/"+target);
                    _ta_output_agent.append("transferAgent: agent "+ agent_identity +
                                            " transferred to "+ target + "\n");
                }
                catch(Exception e1){
                    if (e1 instanceof org.omg.CORBA.BAD_PARAM) { }
                    else {
                        e1.printStackTrace();
                        _ta_output_agent.append("transferAgent failed: "+e1.toString()+"\n"+e1.getMessage()+"\n");
                    }

                }
                //clear selection in agentTable and disable buttons for safety reasons
                clear_selection();
            }
        }
                                           );
        //disable button initially
        _bt_transferAgent.setEnabled(false);


        /*
         *  action listener for resume_agent button
         */
        _bt_resume_agent.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                try {
```

```
            //resume agent
            NameWrapper nw = new NameWrapper(agent_identity);
            _ass.resume_agent(nw._name);

            //update the status of the agent in the agentTable
            myModel.changeStatus(agentTable.getSelectedRow(), "Running");

            //display status message in output window
            _ta_output_agent.append("resume_agent(): "+agent_identity+" resumed.\n\n");
        }
        catch(Exception e2){
            _ta_output_agent.append("resume_agent for "+agent_identity+" failed!\n"
                                    +e2.toString()+"\n\n");
        }
        //clear selection in agentTable and disable buttons for safety reasons
        clear_selection();
    }
}
                                   );
//disable button initially
_bt_resume_agent.setEnabled(false);



/*
 *  action listener for suspend_agent button
 */
_bt_suspend_agent.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        try {
            //warn the user if the webserver agent is to be suspended
            if (agent_identity.equals("Webserver") ||
                agent_identity.equals("ASManagementAgent")) {

                int answer = JOptionPane.showConfirmDialog(null, "Do you really want to suspend this agent?\nThis applet would no longer work correctly!", "Warning!", JOptionPane.YES_NO_OPTION);
                if (answer == JOptionPane.NO_OPTION) {
                    clear_selection();
                    return;
                }

            }

            //suspend agent
            NameWrapper nw= new NameWrapper(agent_identity);
            _ass.suspend_agent(nw._name);

            //update the status of the agent in the agentTable
            myModel.changeStatus(agentTable.getSelectedRow(), "Suspended");

            //display status message in output window
            _ta_output_agent.append("suspend_agent(): "+agent_identity+" suspended.\n\n");
        }
        catch(Exception e3){
            e3.printStackTrace();
            _ta_output_agent.append("suspend_agent for "+agent_identity+" failed!\n"
                                    +e3.toString()+"\n\n");
        }
        //clear selection in agentTable and disable buttons for safety reasons
        clear_selection();
    }
}
                                    );
//disable button initially
_bt_suspend_agent.setEnabled(false);



/*
 *  action listener for terminate_agent button
 */
_bt_terminate_agent.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        try {
            //warn the user if the webserver is to be terminated
            if (agent_identity.equals("Webserver") ||
                agent_identity.equals("ASManagementAgent")) {
                int answer = JOptionPane.showConfirmDialog(null, "Do you really want to terminate this agent?\nThis applet would no longer work correctly!", "Warning!", JOptionPane.YES_NO_OPTION);
                if (answer == JOptionPane.NO_OPTION) {
                    clear_selection();
                    return;
                }
            }

            //terminate agent
            NameWrapper nw= new NameWrapper(agent_identity);
            _ass.terminate_agent(nw._name);

            //display status message in output window
            _ta_output_agent.append("terminate_agent(): "+agent_identity+" terminated.\n\n");
        }
        catch(Exception e4){
            _ta_output_agent.append("terminate_agent for "+agent_identity+" failed!\n"
                                    +e4.toString()+"\n\n");
        }
        //clear selection in agentTable and disable buttons for safety reasons
        clear_selection();
    }
}
                                      );
//disable button initially
_bt_terminate_agent.setEnabled(false);


/*
 *  action listener for get_agent_system_info button
 */
_bt_get_agent_system_info.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        try {
            //get information about the agent system this applet belongs to via _ass
            AgentSystemInfo asi = _ass.get_agent_system_info();

            //display the information in the output window
            _ta_output_agent_system.append("get_agent_system_info():\n"+
                                           "Name:\n"+
                                           "  Agent System Type: "+String.valueOf((int)asi.agent_system_name.agent_system_type)+"\n"+
                                           "  Authority: "+new String(asi.agent_system_name.authority)+"\n"+
                                           "  Identity: "+new String(asi.agent_system_name.identity)+"\n"+
                                           "Description: "+ asi.agent_system_description+"\n\n");
        }
        catch(Exception e6){
            e6.printStackTrace();
            _ta_output_agent_system.append("get_agent_system_info failed: "+e6.toString()+"\n"+e6.getMessage()+"\n");
        }
    }
}
                                            );


/*
 *  action listener for terminate_agent_system button
 */
_bt_terminate_agent_system.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        try {
            //give the user a last chance to cancel
```

```
            int answer = JOptionPane.showConfirmDialog(null, "Do you really want to terminate this agent system?", "Warning", JOptionPane.YES_NO_OPTION);
            if (answer == JOptionPane.NO_OPTION) {
                return;
            }
            //terminate agent system via _ass
            _ass.terminate_agent_system();
        }
        catch(TerminateFailed e7){
            e7.printStackTrace();
            _ta_output_agent_system.append("terminate_agent_system(): failed!\n"
                                           +e7.toString()+"\n"+e7.getMessage()+"\n"
                                           +"This applet no longer works!\n");

        }
        catch(Throwable cf){
            // thrown when agent system is disconnected from boa
            _ta_output_agent_system.append("terminate_agent_system(): all agents and this agent system is terminated!\n"+
                                           "This applet no longer works!\n");
        }
    }
}
                                             );



/*
 *  action listener for readAgentFromFile button
 */
_bt_readAgentFromFile.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        //get name and path of the file to be used
        String file = _tf_file.getText();
        try {
            //read and create the agent from the data provided by the file
            Name name= _ass.readAgentFromFile(file);

            _ta_output_agent.append("Read agent "+new String(name.identity)+" from file "+file+"\n");
        }
        catch(Exception e11){
            e11.printStackTrace();
            _ta_output_agent.append("Read agent failed: "+e11.toString()+"\n"+e11.getMessage()+"\n");
        }
        //clear selection in agentTable and disable buttons for safety reasons
        clear_selection();
        list_agent_names();  // Update the agent list
    }
}
                                        );



/*
 *  action listener for writeAgentToFile button
 */
_bt_writeAgentToFile.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        //get name and path of the file to be used
        String file = _tf_file.getText();
        try {
            //write the agent to the file
            NameWrapper nw = new NameWrapper(agent_identity);
            _ass.writeAgentToFile(nw._name,file);
            _ta_output_agent.append("Agent "+agent_identity+" to file "+file+" written.\n");
        }
        catch(Exception e12){
            e12.printStackTrace();
            _ta_output_agent.append("Write agent failed: "+e12.toString()+"\n"+e12.getMessage()+"\n");
        }
        //clear selection in agentTable and disable buttons for safety reasons
        clear_selection();
        list_agent_names();  // Update the agent list
    }
}
                                       );

//disable button initially
_bt_writeAgentToFile.setEnabled(false);



/*
 *  action listener for showwebpage button
 */
_bt_showwebpage.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        try {
            //create a URL object for the selected agent
            java.net.URL agent_url = CommonAppletHelpers.getAgentURL(agent_identity, _ass.getURL());
            //open another browser window and show agent's webpage
            getAppletContext().showDocument(agent_url, agent_identity);
        }
        catch (Exception ei) {
            ei.printStackTrace();
        }
        //clear selection in agentTable and disable buttons for safety reasons
        clear_selection();
    }
}
                                  );

//disable button initially
_bt_showwebpage.setEnabled(false);



/*
 *  action listener for update_list button
 */

_bt_update_list.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        list_agent_names();
        clear_selection();
    }
}
                                  );



/*
 *  action listener for update_impl_agents button
 */
_bt_update_impl_agents.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        //prevent from updating too early
        _ch_impl_agents.removeActionListener(actionlistener_ch_impl_agents);
        //reload ImplementedAgents.txt
        get_impl_agents();
        //update JCombobox
        _ch_impl_agents.removeAllItems();
        for (int i=0; i<impl_agent_list.length; i++) {
            _ch_impl_agents.addItem(impl_agent_list[i]);
        }
        //reenable ActionListener
        _ch_impl_agents.addActionListener(actionlistener_ch_impl_agents);
        //preselect the first choice
        _ch_impl_agents.setSelectedIndex(0);
        _ch_impl_agents.setEditable(false);
    }
}
```

```
                                            );


    /*
     *  mouse listener for agentTable
     */
    MouseListener ml = new MouseAdapter() {
        public void mousePressed(MouseEvent e)
            {
                if (e.getClickCount() == 2) {
                    //if double click
                    try {
                        //get URL of selected agent
                        java.net.URL agent_url = CommonAppletHelpers.getAgentURL(agent_identity, _ass.getURL());
                        //open another browser window and show agent's webpage
                        getAppletContext().showDocument(agent_url, agent_identity);
                    }
                    catch (Exception ei) {
                        ei.printStackTrace();
                    }
                }
            }
    };
    agentTable.addMouseListener(ml);

}



/**
 * Retrieves a list of agents on this agent system.
 * <br>The name of each loaded agent and it´s status (running/suspended) are
 * copied to the TableModel of agentTable.
 */

public void list_agent_names() {
    String[][] names_n_status = new String[0][0];
    try {
        //contact agent system and retrieve a list of the names of all loaded agents
        Name name[] = _ass.list_all_agents();

        //create a 2D array that will contain the names and the status the the loaded agents
        names_n_status = new String[name.length][2];
        NameWrapper status_name;
        AgentStatus state;

        //create new data vector for the AppletTableModel
        myModel.AppletTableModel();

        //provide the name and the status of each loaded agent to the AppletTableModel myModel
        for (int i = 0; i < name.length; i++) {
            names_n_status[i][0] = new String(name[i].identity);
            status_name = new NameWrapper(names_n_status[i][0]);
            state = _ass.get_agent_status(status_name._name);
            switch(state.value()){
            case AgentStatus._CfMAFRunning:   names_n_status[i][1]="Running"; break;
            case AgentStatus._CfMAFSuspended: names_n_status[i][1]="Suspended"; break;
            case AgentStatus._CfMAFTerminated: names_n_status[i][1]="Terminated"; break;
            default: names_n_status[i][1] = "none"; break;
            }
            myModel.addValue(names_n_status[i]);
        }
        //update the displayed agentTable
        getContentPane().validate();
        tableScrollPane.repaint();
        //agentTable_panel.repaint();
    }
    catch(Exception e){
        e.printStackTrace();
    }
}


/**
 * Defines what´s to be done when applet is terminated.
 * <br>Disconnects pushConsumer that has been registered with the ASManagementAgent
 */

public void destroy() {
    try {
        _asm.disconnect_push_consumer(pushConsumer.id);
    } catch (Exception e) {
        System.err.println("\n\nAgentSystemApplet: Could not disconnect pushConsumer");
        e.printStackTrace();
    }
}


/**
 * Create a PushConsumer and register it with the ASManagementAgent.
 * <br>The pushConsumer is then provided with CORBA events and reacts appropriately
 */

public void create_pushConsumer() {
    try {
        // create PushConsumer
        pushConsumer = new PushConsumerImpl();

            org.omg.CORBA.ORB orb = CommonAppletHelpers.initialize_orb_for_push_consumer();

        //use the Portable Object Adapter (POA) to register the object with the ORB
        orb.connect(pushConsumer);

        //register pushConsumer with the ASManagementAgent and retrieve the ID
        //It is later used to disconnect the pushConsumer from the asm again
        pushConsumer.id = _asm.connect_push_consumer(pushConsumer);

    } catch (Exception e1) {
        e1.printStackTrace();
    }
}



/**
 * Retrieve a list of implemented agents and prepare for later use.
 * <br>The file "ImplementedAgents.txt" contains a list of agents that are ready to load. It
 * provides the name, the package and the type (mobile/stationary) of the agents. It is
 * obtained using CommonAppletHelpers.retrieve_implemented_agents_file(...).
 *
 * @see CommonAppletHelpers
 */

public void get_impl_agents() {

    //retrieve the file ImplementedAgents.txt and store it into string_file
    String string_file = CommonAppletHelpers.retrieve_implemented_agents_file(_ass.getURL());

    //copy the data into the appropriate arrays
    //they will be used when loading agents into the agent system with create_agent
    java.util.StringTokenizer agent_tokenizer = new java.util.StringTokenizer(string_file);
    int noe = (agent_tokenizer.countTokens())/3;
    impl_agent_list = new String[noe];
    impl_package_list = new String[noe];
    impl_type_list = new String[noe];

    for (int i=0; i<noe; i++) {
```

```
            impl_agent_list[i] = agent_tokenizer.nextToken();
            impl_package_list[i] = agent_tokenizer.nextToken();
            impl_type_list[i] = agent_tokenizer.nextToken();
        }
    }



    /**
     * Clears the selection in the agentTable and disables the buttons.
     * <br>To prevent accidential execution of agent-manipulating methods the selection in the agentTable
     * and the buttons that need a preselected agent are disabled after each action. The String variable
     * agent_identity is also cleared.
     */

    public void clear_selection() {
        agentTable.clearSelection();
        agent_identity = null;
        _bt_transferAgent.setEnabled(false);
        _bt_resume_agent.setEnabled(false);
        _bt_suspend_agent.setEnabled(false);
        _bt_terminate_agent.setEnabled(false);
        _bt_writeAgentToFile.setEnabled(false);
        _bt_showwebpage.setEnabled(false);
    }



    /**
     * Retrieves the parameters for all constructors defined for the selected agent.
     * <br>An agent can have more than one constructor, each with a different set of parameters. These
     * parameters are collected into an array of Vectors (param_matrix), each Vector representing
     * the parameter set of one constructor.<br>
     * The applet presents to the user a combobox (_ch_constructors) containing a list of constructor
     * choices defined by the number of parameters the constructor expects.
     * The parameters are retrieved using the ASManagementAgent.
     *
     * @see ASManagementAgent
     */

    public void get_agent_parameters() {
        //get name, package and type (mobile/stationary) of the selected agent that is to be loaded
        String agent_name = (String) _ch_impl_agents.getSelectedItem();
        int agent_index = _ch_impl_agents.getSelectedIndex();
        String agent_package = impl_package_list[agent_index];
        String agent_type = impl_type_list[agent_index];

        //get parameters of all possible constructors for this agent
        //the parameter sets of the different constructors are located in the array elements
        //of the returned String array
        String parametersets[] = _asm.get_agent_parameters(agent_name, agent_package, agent_type);

        //check if ASManagementAgent could find the agent's class
        //if not then display a warning message
        if (parametersets[0].equals("failed")) {
            JOptionPane.showMessageDialog(null, "The class of the agent \""+agent_name+"\" could not be found.\nPlease check if the agent is properly installed.");
        }

        //parameter_set_tokenizer is used to retrieve the different sets of parameters
        //java.util.StringTokenizer parameter_set_tokenizer = new java.util.StringTokenizer(parameters, "#");
        //parameter_tokenizer is used to retrieve the parameters within one set
        java.util.StringTokenizer parameter_tokenizer = null;
        //get number of parameter sets (= number of different constructors)
        int nop = parametersets.length;//parameter_set_tokenizer.countTokens();

        if (nop == 0) {  //no constructor defined
            param_matrix = new Vector[1];
            param_matrix[0] = new Vector(0,1);
            constructor_list = new String[1];
            constructor_list[0] = "0 arguments";
        }
        else {
            param_matrix = new Vector[nop];  //each element of the array represents one parameter set
            constructor_list = new String[nop];  //list which will be shown in _ch_constructors ComboBox
        }

        //create the "matrix" of parameters and set up the combobox with constructor choices
        for (int i=0; i<nop; i++) {
            //parameter_tokenizer = new java.util.StringTokenizer(parameter_set_tokenizer.nextToken());  //get parameters of one constructor
            parameter_tokenizer = new java.util.StringTokenizer(parametersets[i]); //get parameters of one constructor
            int nop2 = parameter_tokenizer.countTokens();
            param_matrix[i] = new Vector(0,1);
            constructor_list[i] = nop2 + " arguments";
            for (int j=0; j<nop2; j++) {
                //copy parameters of one constructor into appropriate vector
                param_matrix[i].addElement(parameter_tokenizer.nextToken());
            }
        }
        //update the displayed combobox with constructor choices
        repaint_ch_constructors();
    }


    /**
     * Redefines the ComboBox showing the choice between possible constructors of the selected agent.
     * <br>Called when a new agent is selected from the implemented-agents-combobox (_ch_impl_agents).
    */

    public void repaint_ch_constructors() {
        if (_ch_constructors != null) {  //if it has already been called...

            //update the contents of _ch_constructors
            _ch_constructors.removeAllItems();
            for (int i=0; i<constructor_list.length; i++) {
                _ch_constructors.addItem(constructor_list[i]);
            }
            //preselect the first choice
            _ch_constructors.setSelectedIndex(0);
            _ch_constructors.setEditable(false);

            //repaint JComboBox
            getContentPane().validate();
            _ch_constructors.repaint();

            //list parameters for the first constructor
            show_agent_parameters(0);
        }
    }


    /**
     * Shows the parameter set of the selected constructor in the table.
     * <br>The value of each parameter is predefined with the String constant "none"
     *
     * @param param_index the index of the selected constructor
     */

    public void show_agent_parameters(int param_index) {
        if (param_index != (-1)) {  //(-1) after _ch_constructors.removeAllElements()
            paramTableModel.ParameterTableModel();  //create new data vector

            int nop = param_matrix[param_index].size();

            for (int i=0; i<nop; i++) {
                String tmp[] = new String[2];
                tmp[0] = (String) param_matrix[param_index].elementAt(i);  //get parameter #i

                //check if parameter type is a valid one
```

```
            if (!(tmp[0].equals("java.lang.String")) &&
                !(tmp[0].equals("java.lang.Integer")) &&
                !(tmp[0].equals("java.lang.Float")) &&
                !(tmp[0].equals("java.lang.Double")) &&
                    //(roelle): IMHO this was missing
                !(tmp[0].equals("java.lang.Boolean")) &&
                !(tmp[0].equals("failed"))
                ) {
                JOptionPane.showMessageDialog(null, "\""+tmp[0]+"\" is not a valid parameter data type!");
            }

            tmp[1] = "none";
            paramTableModel.addValue(tmp);
        }
        //repaint table
        getContentPane().validate();
        paramScrollPane.repaint();
    }
}



/**
 * TableModel managing a list of arguments for an agent´s constructor.
 * <br>This TableModel manages a parameter set of one constructor. Each time a different constructor
 * is selected the Vector "data" is completely new defined. Two columns are defined. The first one
 * contains the data type of a parameter, the second one it´s value which must be provided
 * by the user.
 *
 * @see AbstractTableModel
 * @author Stefan Gerber
 */

class ParameterTableModel extends AbstractTableModel {
    String[] columnNames = {"Expected Parameters", "Values"};
    Vector data;


    public void ParameterTableModel()
    {
        data = new Vector(0,1);
    }

    public int getColumnCount()
    {
        return columnNames.length;
    }

    public int getRowCount()
    {
        return data.size();
    }

    public String getColumnName (int col)
    {
        return columnNames[col];
    }

    public Object getValueAt(int row, int col)
    {
        return ((String[]) data.elementAt(row))[col];
    }

    public void setValueAt(Object value, int row, int col)
    {
        //first get old value
        String newvalue[] = {(String) ((String[]) data.elementAt(row))[0],
                             (String) ((String[]) data.elementAt(row))[1]};
        newvalue[col] = (String) value;
        data.setElementAt(newvalue, row);

        fireTableCellUpdated(row, col);
    }

    public void addValue(String[] value)
    {
        data.addElement(value);
        fireTableRowsInserted(data.size()-1, data.size()-1);
    }

    public void rmValue(int row)
    {
        data.removeElementAt(row);
        fireTableRowsDeleted(row, row);
    }

    public boolean isElement(String an) {
        for (int i=0; i<data.size(); i++) {
            if (an.equals((String) (((String[]) data.elementAt(i))[0]))) return true;
        }
        return false;
    }

    public boolean isCellEditable(int row, int col) {
        //Note that the data/cell address is constant,
        //no matter where the cell appears onscreen.
        if (col < 1) {
            return false;
        } else {
            return true;
        }
    }
}



/**
 * TableModel managing the list of loaded agents in the agent system and their current status.
 * <br>Two columns are defined. The first one contains the names of the currently loaded agents,
 * the second one represents their status (running/suspended).
 *
 * @see AbstractTableModel
 * @author Stefan Gerber
 */

class AppletTableModel extends AbstractTableModel {
    String[] columnNames = {"Agent", "Status"};
    Vector data;


    public void AppletTableModel()
    {
        data = new Vector(0,1);
    }

    public int getColumnCount()
    {
        return columnNames.length;
    }

    public int getRowCount()
    {
        return data.size();
    }

    public String getColumnName (int col)
    {
```

```
        return columnNames[col];
    }

    public Object getValueAt(int row, int col)
    {
        return ((String[]) data.elementAt(row))[col];
    }

    public void setValueAt(Object value, int row, int col)
    {
        ((String[]) data.elementAt(row))[col] = (String) value;
        fireTableCellUpdated(row, col);
    }

    public void addValue(String[] value)
    {
        data.addElement(value);
        fireTableRowsInserted(data.size()-1, data.size()-1);
    }

    public void rmValue(int row)
    {
        data.removeElementAt(row);
        fireTableRowsDeleted(row, row);
    }

    public boolean isElement(String an) {
        for (int i=0; i<data.size(); i++) {
            if (an.equals((String) (((String[]) data.elementAt(i))[0]))) return true;
        }
        return false;
    }

    public void changeStatus(int row, String statusValue) {
        setValueAt(statusValue, row, 1);
    }

    public int whereis(String an) {
        for (int i=0; i<data.size(); i++) {
            if (an.equals((String) (((String[]) data.elementAt(i))[0]))) return i;
        }
        return (-1);
    }

}



/**
 * SelectionHandler for list of loaded agents (agentTable).
 * <br>Each time the user selects an agent from the table of loaded agents, the String variable
 * agent_identity is set and the buttons for the agent management are enabled.
 *
 * @see ListSelectionListener
 * @author Stefan Gerber
 */

class ListSelectionHandler implements ListSelectionListener
{
    public void valueChanged(ListSelectionEvent e)
    {
        ListSelectionModel ls_model = (ListSelectionModel) e.getSource();
        int index = ls_model.getMinSelectionIndex();
        if (index < 0) {
            _bt_transferAgent.setEnabled(false);
            _bt_resume_agent.setEnabled(false);
            _bt_suspend_agent.setEnabled(false);
            _bt_terminate_agent.setEnabled(false);
            _bt_writeAgentToFile.setEnabled(false);
            _bt_showwebpage.setEnabled(false);
            agent_identity = null;
        }
        else {
            /* check if selected agent is mobile before enabling the transfer_agent-button */
            int file_index = -1;
            String selected_agent = (String) agentTable.getValueAt(index, 0);
            //determine index where selected agent is in list of implemented agents
            //this index is stored in file_index
            for (int i=0; i<impl_agent_list.length; i++) {
                if (selected_agent.equals(impl_agent_list[i])) {
                    file_index = i;
                    break;
                }
            }
            //use file_index to determine if agent is mobile
            //enable button if agent is mobile or if no information about the agent was available
            //i.e. the agent was not in ImplementedAgents.txt
            if (file_index == (-1)) {
                _bt_transferAgent.setEnabled(true);
            }
            else if (impl_type_list[file_index].equals("MobileAgent")) {
                _bt_transferAgent.setEnabled(true);
            }
            else {
                _bt_transferAgent.setEnabled(false);
            }

            //check if selected agent is running or resumed
            if (((String) agentTable.getValueAt(index, 1)).equals("Running")) {
                _bt_resume_agent.setEnabled(false);
                _bt_suspend_agent.setEnabled(true);
            } else {
                _bt_resume_agent.setEnabled(true);
                _bt_suspend_agent.setEnabled(false);
            }
            _bt_terminate_agent.setEnabled(true);
            _bt_writeAgentToFile.setEnabled(true);
            _bt_showwebpage.setEnabled(true);
            agent_identity = (String) agentTable.getValueAt(index, 0);
        }
    }
}


/**
 * Implements the PushConsumer that is connected to the ASManagementAgent.
 * <br>The applet connects an instance of this class to the ASManagementAgent on startup. The agent calls
 * the PushConsumer´s push() method when a CORBA event is registered and supplies the event data. The
 * push() method defined in this class finds out which CORBA event occurred and reacts appropriately.
 *
 * @see ASManagementAgent
 * @author Stefan Gerber
 */

class PushConsumerImpl extends _PushConsumerImplBase {
    String[] event_data;
    public int id = 0;  //id of this pushConsumer, needed for disconnection from ASManagementAgent

    public PushConsumerImpl()
    {
        System.err.println("\n\nAgentSystemApplet: PushConsumer created");
    }

    public void push(org.omg.CORBA.Any any)
    {
        System.out.println("\n\nAgentSystemApplet: running push");

        //extract CORBA event data out of any
```

```
            event_data = new String[5];
            CosNotification.StructuredEvent se = CosNotification.StructuredEventHelper.extract(any);
            event_data[0] = se.header.fixed_header.event_type.domain_name;
            event_data[1] = se.header.fixed_header.event_type.type_name; //AgentUp/AgentDown/agentSystemUp/agentSystemDown
            event_data[2] = se.filterable_data[0].value.extract_string();  //agent_type
            event_data[3] = se.filterable_data[1].value.extract_string();  //agent_name/agent_system_ame
            event_data[4] = se.filterable_data[2].value.extract_string();  //event_channel_name

            if (agent_system_name.equals(event_data[0])) {  // is this AS concerned?
                if (event_data[1].equals("AgentUp")) {  //agent was started on this agent system
                    String status = null;
                    try {
                        //find out status of the new agent
                        AgentStatus state = _ass.get_agent_status((new NameWrapper(event_data[3]))._name);
                        switch(state.value()){
                        case AgentStatus._CfMAFRunning: status = "Running"; break;
                        case AgentStatus._CfMAFSuspended: status = "Suspended"; break;
                        case AgentStatus._CfMAFTerminated: status = "Terminated"; break;
                        default: status = "none"; break;
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }

                    //add name and status to agentTable
                    String[] value = {event_data[3], status};
                    myModel.addValue(value);

                    //repaint agentTable
                    getContentPane().validate();
                    tableScrollPane.repaint();
                    //agentTable_panel.repaint();

                    //clear selection in agentTable and disable buttons for safety reasons
                    clear_selection();
                }
                else if (event_data[1].equals("AgentDown")) {  //agent was terminated on this agent system
                    //remove entry of the agent from agentTable
                    myModel.rmValue(myModel.whereis(event_data[3]));

                    //repaint agentTable
                    getContentPane().validate();
                    tableScrollPane.repaint();
                    //agentTable_panel.repaint();

                    //clear selection in agentTable and disable buttons for safety reasons
                    clear_selection();
                }
            }
        }


        public void disconnect_push_consumer() {
            try {
                // Initialize the ORB.
                //org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
                // Initialize the BOA.
                //org.omg.CORBA.BOA boa = orb.BOA_init();

                System.err.println("\n\nDisconnecting push_consumer...\n");
                //boa.exit_impl_ready();
            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

}
```

Anhang G

Listing des RegionManagementAgent-Applets

```
package __MASA_PACKAGE_thisagent;

//import __MASA_PACKAGE_thisagent__(*);
//import __MASA_PACKAGE_agentSystem__(AgentSystem.*);

import __MASA_PACKAGE_agent__(ASManagementAgent.*);

import __MASA_PACKAGE_agent__(*);
import __MASA_PACKAGE_agentSystem__(*);
import __MASA_PACKAGE__(tools.NameWrapper);
import __MASA_PACKAGE_CfMAF__(AgentStatus);

import org.omg.CosNaming.*;

import java.awt.*;
import java.applet.*;
import java.util.*;

import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.text.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.tree.*;
import com.sun.java.swing.event.*;

import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdmin.*;


/**
 * This is the applet of the RegionManagementAgent.
 * <br>Its main use is to display a split window the left half of which showing the currently active
 * agent systems, the right half listing the agents loaded into these agent systems. One can also
 * use the applet to call up the webpages/applets of the different agents and agent systems. Agents can be
 * tranferred from one agent system to another.
 *
 * @author Stefan Gerber
 */

public class RegionManagementAgentApplet extends AgentApplet
{
    /*
     * CORBA reference to the agent system
     */
    private AgentSystemService _ass;

    private ASManagementAgent _asm;

    private RegionManagementAgent _rma;

    private JSplitPane splitpane;
    private JScrollPane tree_panel;
    private JScrollPane leaf_panel;
    private DefaultMutableTreeNode root;
    private JTree tree;
    private DefaultTreeModel treemodel;

    private AppletTableModel myModel;
    private JTable agentTable;
    private JPopupMenu popup;
    private int row;  //is set by the ListSelectionListener of agentTable

    private Vector as_names;  //needed primarily for invoking CommonAppletHelpers.show_migrate_menu()
    private DefaultMutableTreeNode selected_node;  //is set by the TreeSelectionListener of tree
    private JPopupMenu as_popup;
    private JButton updateTreeButton;
    private JButton updateListButton;
    private JPanel left_panel;
    private JPanel right_panel;

    private PushConsumerImpl pushConsumer;


    /**
     * Initialize this applet and connect to ASManagementAgent and RegionManagementAgent.
     * <br>Sets and displays all JFC components and calls various methods to get initial information
     * about the agent systems and their agents (e.g. list of active agent systems, list of agents loaded
     * in the currently selected agent system)
     *
     * @see ASManagementAgent
     */

    public void init() {

        // Set layout to GridBag
        GridBagLayout gridBag = new GridBagLayout();
        getContentPane().setLayout(gridBag);
        GridBagConstraints c_root = new GridBagConstraints();

        // Connect to appropriate agent system
        try {
            org.omg.CORBA.Object obj = initCORBA("AgentSystemService");
            _ass = AgentSystemServiceHelper.narrow(obj);
        }
        catch(java.net.MalformedURLException e) {
            System.err.println(e.toString()+"\n");
        }
        catch(java.io.IOException ioe) {
            System.err.println(ioe.toString()+"\n");
        }


        //get CORBA reference of the own agent system's ASManagementAgent
```

```
try {
    org.omg.CORBA.Object objekt = initCORBA("ASManagementAgent");
    _asm = ASManagementAgentHelper.narrow(objekt);
}
catch(java.net.MalformedURLException e) {
    System.err.println(e.toString()+"\n");
}
catch(java.io.IOException ioe) {
    System.err.println(ioe.toString()+"\n");
}


//get CORBA reference of the own RegionManagementAgent
try {
    org.omg.CORBA.Object objekt = initCORBA("RegionManagementAgent");
    _rma = RegionManagementAgentHelper.narrow(objekt);
}
catch(java.net.MalformedURLException e) {
    System.err.println(e.toString()+"\n");
}
catch(java.io.IOException ioe) {
    System.err.println(ioe.toString()+"\n");
}



// test if ASManagementAgent has been found
// if not, then display warning message
if (_asm == null) {
    JOptionPane.showMessageDialog(null, "ASManagementAgent could not be found.\nPlease load it and restart the applet.");
}
else {

    /***********************************************************
     *******        Panel for the tree          ****************
     ***********************************************************/

    //Create tree of currently active agent systems
    create_tree();

    //panel to the left which contains the tree
    tree_panel = new JScrollPane(tree, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    tree_panel.setBorder(BorderFactory.createTitledBorder("Agent Systems & Co"));

    //create Button that updates the agentsystems-tree
    updateTreeButton = new JButton("update manually");
    updateTreeButton.setMnemonic('u');
    updateTreeButton.setToolTipText("update tree of agent systems manually");
    updateTreeButton.setActionCommand("button");
    updateTreeButton.setBackground(new Color(12632256));
    updateTreeButton.setMargin(new Insets(2,2,2,2));

    updateTreeButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            update_tree();
        }
    }
                                      );

    // Set layout of left_panel to GridBag
    GridBagLayout left_gridBag = new GridBagLayout();
    GridBagConstraints panel_constraints = new GridBagConstraints();

    //create the panel that will contain the agentsystems-tree and the button(s)
    left_panel = new JPanel(left_gridBag);



    /***********************************************************
     *******      Panel for the agent table     ****************
     ***********************************************************/

    //Creating the table showing the agents of the selected system
    myModel = new AppletTableModel();
    myModel.AppletTableModel();  //create a new vector
    agentTable = new JTable(myModel);
    agentTable.setSelectionMode((new DefaultListSelectionModel()).SINGLE_SELECTION);
    myModel.addTableModelListener(agentTable);


    //add PopupMenu and MouseListener to agentTable
    popup = create_popup();
    //the global variabel row is set to the number of the row in agentTable selected by the user
    MouseListener ml_table = new MouseAdapter() {
        public void mousePressed(MouseEvent e) {
            String selected_cell = (String) selected_node.getUserObject();
            if (!(selected_cell.equals("global")) &&
                !(selected_cell.equals("voyager"))) { //if global or voyager are not selected

                if (e.isPopupTrigger()) {
                        //right mouse button clicked
                    row = agentTable.rowAtPoint(e.getPoint());  //get row on which Mouse is located on

                        //check if agent is mobile
                    String impl_agents = CommonAppletHelpers.retrieve_implemented_agents_file(_ass.getURL());
                    java.util.StringTokenizer agent_tokenizer = new java.util.StringTokenizer(impl_agents);
                    String agent_type = new String();
                    while (agent_tokenizer.hasMoreTokens()) {
                        if ((agent_tokenizer.nextToken()).equals((String) myModel.getValueAt(row,0))) {
                        //if selected agent has been found in the list of implemented agents
                            agent_type = agent_tokenizer.nextToken(); //this is the agent´s package
                            agent_type = agent_tokenizer.nextToken(); //this is the agent´s type
                            break;
                        }
                    }
                        //then find out if selected agent is mobile
                    if (agent_type.equals("StationaryAgent")) { //agent cannot be migrated
                        //get menuitem "migrate" and disable it
                        JMenuItem mi = (JMenuItem) popup.getComponentAtIndex(0);
                        mi.setEnabled(false);
                    } else {
                        //get menuitem "migrate" and enable it
                        JMenuItem mi = (JMenuItem) popup.getComponentAtIndex(0);
                        mi.setEnabled(true);
                    }

                    popup.show(e.getComponent(), e.getX(), e.getY());
                }


                //if agent is double-clicked then show its webpage/applet
                if (e.getClickCount() == 2) {
                        //Double Click
                    show_agents_webpage();
                }
            }
        }

    };
    agentTable.addMouseListener(ml_table);

    //create panel which contains agentTable
    leaf_panel = new JScrollPane(agentTable, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    leaf_panel.setBorder(BorderFactory.createTitledBorder("Agents in selected agent aystem"));

    //create Button that updates the list of agents
    updateListButton = new JButton("update manually");
    updateListButton.setMnemonic('p');
```

```
        updateListButton.setToolTipText("update list of agents in selected agent system manually");
        updateListButton.setActionCommand("button");
        updateListButton.setBackground(new Color(12632256));
        updateListButton.setMargin(new Insets(2,2,2,2));

        updateListButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                list_agent_names();
            }
        }
                                           );

        // Set layout of right_panel to GridBag
        GridBagLayout right_gridBag = new GridBagLayout();

        //create the panel that will contain the list of agents and the button(s)
        right_panel = new JPanel(right_gridBag);



        /************************************************************
         ******* add the components to their panels ****************
         ***********************************************************/

        panel_constraints.fill = GridBagConstraints.BOTH;
        panel_constraints.gridwidth = GridBagConstraints.REMAINDER;
        panel_constraints.gridx = 0;
        panel_constraints.weightx = 1;
        panel_constraints.weighty = 1;

        left_gridBag.setConstraints(tree_panel, panel_constraints);
        left_panel.add(tree_panel);
        right_gridBag.setConstraints(leaf_panel, panel_constraints);
        right_panel.add(leaf_panel);

        panel_constraints.fill = GridBagConstraints.HORIZONTAL;
        panel_constraints.gridwidth = GridBagConstraints.REMAINDER;
        panel_constraints.gridx = 0;
        panel_constraints.weightx = 1;
        panel_constraints.weighty = 0;

        left_gridBag.setConstraints(updateTreeButton, panel_constraints);
        left_panel.add(updateTreeButton);
        right_gridBag.setConstraints(updateListButton, panel_constraints);
        right_panel.add(updateListButton);


        //create the splitpane to contain right_panel and left_panel
        splitpane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, left_panel, right_panel);
        //splitpane.setDividerLocation(0.4);  //ignored by JVM, reported Bug
        splitpane.setPreferredSize(new Dimension(280, 520));  //workaround
        splitpane.setBorder(BorderFactory.createLineBorder(new Color(255,255,255).lightGray, 5));

        c_root.fill = GridBagConstraints.BOTH;
        c_root.gridx = 0;
        c_root.gridwidth = GridBagConstraints.REMAINDER;
        c_root.gridheight = GridBagConstraints.REMAINDER;
        c_root.weightx = c_root.weighty = 1.0;
        gridBag.setConstraints(splitpane, c_root);

        //add splitpane to Applet
        getContentPane().add(splitpane);

        //create pushConsumer and connect it to the ASManagementAgent
        //needed to react to events from CORBA event service
        create_pushConsumer();
    }
}
```

```
    /**
     * Defines what´s to be done when applet is terminated.
     * <br>Disconnects pushConsumer that has been registered with the ASManagementAgent
     */

    public void destroy() {
        try {
            _asm.disconnect_push_consumer(pushConsumer.id);
        } catch (Exception e) {
            System.err.println("\n\nRegionManagementAgentApplet: Could not disconnect pushConsumer");
            e.printStackTrace();
        }
    }


    /**
     * Opens a new browser window and displays the webpage of the agent system specified by "as".
     * <br>The URL is achieved using the RegionManagementAgent method "get_AS_URL()", because the applet
     * cannot communicate with an agent system not located on the same machine.
     *
     * @param as name of the agent system
     */

public void show_agentsystems_webpage(String as) {
    try {
        //generate URL of selected Agent system's webserver
        java.net.URL url_target = new java.net.URL(_rma.get_AS_URL(as));

        //Open new browser window and show agent system's webpage (titled "as")
        getAppletContext().showDocument(url_target, as);

    } catch (java.net.MalformedURLException mfe) {
        mfe.printStackTrace();
    }
}


/**
     * Opens a new browser window and displays the webpage of the selected agent.
     * <br>The URL is achieved using CommonAppletHelpers.getAgentURL(). This method is called
     * when the user selects the appropriate menu item from the context menu of an agent.
     */

public void show_agents_webpage() {
    //get name of selected agent. "row" has been set when right mouse button was clicked on the agent
    String agent_identity = (String) myModel.getValueAt(row,0);
    try {
        //get the URL of the selected agent.
        java.net.URL agent_url = CommonAppletHelpers.getAgentURL(agent_identity, _rma.get_AS_URL((String) selected_node.getUserObject()));
        getAppletContext().showDocument(agent_url, agent_identity);  //open another browser window and show agent's webpage
    }
    catch (Exception eaw) {
        eaw.printStackTrace();
    }
}


/**
     * Creates and returns the popup menu for the tree of active agent systems.
     * <br>The menu is shown, when the user right-clicks on an agent system in the tree.
     *
     * @param as name of agent system
     * @return JPopupMenu for the agent system
     */

public JPopupMenu create_as_popup(final String as) {
    JPopupMenu popup = new JPopupMenu();
```

```
    //define menu item that will show the selected agent system´s webpage
    JMenuItem mi = new JMenuItem("Show webpage");
    mi.setActionCommand("show_webpage");
    mi.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            show_agentsystems_webpage(as);
        }
    }
                        );
    popup.add(mi);


    JMenuItem mi2 = new JMenuItem("Terminate agent system");
    mi2.setActionCommand("terminate_agent_system");
    mi2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            try {
                //give the user a last chance to cancel
                int answer = JOptionPane.showConfirmDialog(null, "Do you really want to terminate this agent system?", "Warning!", JOptionPane.YES_NO_OPTION);
                if (answer == JOptionPane.NO_OPTION) {
                    return;
                }

                _rma.terminate_agent_system(as);

            } catch (Exception exc) {
                exc.printStackTrace();
                System.err.println("\nRegionManagementAgentApplet: terminate_agent_system failed!");
            }
        }
    }
                         );
    popup.add(mi2);

    return popup;
}


    /**
     * Creates and returns the popup menu for the list of agents in the selected agent system.
     * <br>The menu is shown, when the user right-clicks on an agent in the table.
     *
     * @return JPopupMenu for the agents
     */

public JPopupMenu create_popup() {
    //define menu item that initiates transfer of an agent
    JPopupMenu popup = new JPopupMenu();
    JMenuItem mi = new JMenuItem("Migrate agent");
    mi.setActionCommand("migrate");
    mi.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            migrate_agent();
        }
    }
                        );
    popup.add(mi);

    //define menu item that will show the selected agent´s webpage
    JMenuItem mi2 = new JMenuItem("Show webpage");
    mi2.setActionCommand("Show_Webpage");
    mi2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            show_agents_webpage();
        }
    }
                         );
    popup.add(mi2);
```

```
    return popup;
}



/**
     * Transfers the agent selected in agentTable to another agent system.
     * <br>This method is called when the user selects "Migrate Agent" from the context menu of an agent
     * in the agentTable. The user is first presented with a dialogue where he can select the target
     * agent system. After he has chosen the target the agent is transferred to it. This is done using
     * the RegionManagementAgent because the applet cannot communicate with an agent system not located on
     * the same machine.
     *
     * @see CommonAppletHelpers
     */

public void migrate_agent() {
    //get the name of the selected agent
    String agent_identity = (String) tableModel.getValueAt(row,0);

    //check if RegionManagementAgent is to be migrated and warn user
    if(agent_identity.equals("RegionManagementAgent")) {
        int answer = JOptionPane.showConfirmDialog(null, "Do you really want to migrate the RegionManagementAgent?\nThis applet would no longer work correctly\nif it belongs to this agent!", "Warning!", JOptionPane.YES_NO_OPTION);
        if (answer == JOptionPane.NO_OPTION) {
            return;
        }
    }

    //get target agent system.
    String target = CommonAppletHelpers.show_migrate_menu(this, agent_identity, as_names);


    //initialize agentsystemservice object
    String as = (String) selected_node.getUserObject(); //get name of agent system represented by the node
    _rma.migrate_agent(agent_identity, target, as);
}


    /**
     * Sets up the tree of agent systems.
     * <br>This method is called only once in the init() method of this applet. It identifies all currently
     * active agent systems (via ASManagementAgent) and sets up the tree and its TreeModel with this
     * information. In addition to this it generates a TreeSelectionListener and a MouseListener for
     * the tree.
     */

public void create_tree() {
    //set up tree components
    root = new DefaultMutableTreeNode("Agent System Context");
    treemodel = new DefaultTreeModel(root);
    tree = new JTree(treemodel);
    tree.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);

    //add a tree selection listener
    //if a node is selected then the global variable "selected_node" is set to it and the list of
    //agents in the agent system represented by this node is generated.
    tree.addTreeSelectionListener(new TreeSelectionListener() {
        public void valueChanged(TreeSelectionEvent e) {
            selected_node = (DefaultMutableTreeNode) (e.getPath().getLastPathComponent());
            list_agent_names();
        }
    }

                                  );

    //create MouseListener for the tree
    MouseListener ml_tree = new MouseAdapter() {
        public void mousePressed(MouseEvent e)
            {
```

```
                TreePath tp = tree.getPathForLocation(e.getX(), e.getY());
                tree.setSelectionPath(tp);
                DefaultMutableTreeNode node = (DefaultMutableTreeNode) (tp.getLastPathComponent());
                String as = (String) node.getUserObject();  //get name of agent system

                if (!(as.equals("global")) &&
                    !(as.equals("voyager")) &&
                    !(as.equals("Agent System Context"))) {

                    if (e.getClickCount() == 2) {
                        //if double clicked then show the webpage for the agent system
                        show_agentsystems_webpage(as);
                    }

                    if (e.isPopupTrigger()) {
                        //if right clicked then show popup menu
                        as_popup = create_as_popup(as);  //create Popup Menu
                        as_popup.show(e.getComponent(), e.getX(), e.getY());
                    }
                }
            }

    };
    tree.addMouseListener(ml_tree);

    //get list of active agent systems from ASManagementAgent and add it to the tree
    String agentsystems[] = _asm.identify_agentsystems();
    as_names = new java.util.Vector(0,1); //needed for CommonAppletHelpers.show_migrate_menu()
    int n_o_t = agentsystems.length;

    for (int i=0; i<n_o_t; i++) {
        as_names.addElement(agentsystems[i]);
        root.add(new DefaultMutableTreeNode(as_names.elementAt(i)));
    }
    //repaint the tree
    treemodel.reload();
}


    /**
     * Refreshes the tree showing the active agent systems.
     * <br>Invoked by the update-agent-system-list button. It fetches the list of active agent systems
     * from the ASManagementAgent (because the applet cannot communicate with the CORBA name service
     * directly) and recreates the tree with it.
     */

public void update_tree() {

    root = new DefaultMutableTreeNode("Agent System Context");  //create a new root

    //get list of active agent systems from ASManagementAgent and add it to the tree
    String agentsystems[] = _asm.identify_agentsystems();
    as_names = new java.util.Vector(0,1); //needed for CommonAppletHelpers.show_migrate_menu()
    int n_o_t = agentsystems.length;

    for (int i=0; i<n_o_t; i++) {
        as_names.addElement(agentsystems[i]);
        root.add(new DefaultMutableTreeNode(as_names.elementAt(i)));
    }

    treemodel.setRoot(root);
    selected_node = (DefaultMutableTreeNode) treemodel.getChild(root, 0);
    //repaint the tree
    treemodel.reload();
}


    /**
     * Retrieves a list of agents on the selected agent system.
     * <br>When the user selects an agent system in the tree this method is invoked by the tree´s
     * TreeSelectionListener. It uses the RegionManagementAgent to get the list of agents in this agent system
     * (because the applet cannot communicate with the CORBA name service directly), replaces the old
     * information in the agentTable with it and repaints the table.
     */

public void list_agent_names() {
    myModel.AppletTableModel();   //create new data vector
    String agents[] = _rma.list_agent_names((String) selected_node.getUserObject());

    int n_o_t = agents.length;
    for (int i=0; i<n_o_t; i++) {
        myModel.addValue(agents[i]);
    }

    getContentPane().validate();
    leaf_panel.repaint();
    agentTable.repaint();
}


/**
 * Create a PushConsumer and register it with the ASManagementAgent.
 * <br>The pushConsumer is then provided with CORBA events and reacts appropriately
 */

public void create_pushConsumer() {
    try {
        // create PushConsumer
        pushConsumer = new PushConsumerImpl();

        org.omg.CORBA.ORB orb =
            CommonAppletHelpers.initialize_orb_for_push_consumer();

        //use the Portable Object Adapter (POA) to register the object with the ORB
        orb.connect(pushConsumer);

        //register pushConsumer with the ASManagementAgent and retrieve the ID
        //It is later used to disconnect the pushConsumer from the asm again
        pushConsumer.id = _asm.connect_push_consumer(pushConsumer);

    } catch (Exception e1) {
        e1.printStackTrace();
    }
}



/**
     * Implements the PushConsumer that is connected to the ASManagementAgent.
     * <br>The applet connects an instance of this class to the ASManagementAgent on startup. The agent calls
     * the PushConsumer´s push() method when a CORBA event is registered and supplies the event data. The
     * push() method defined in this class finds out which CORBA event occurred and reacts appropriately.
     *
     * @see ASManagementAgent
     * @author Stefan Gerber
     */

class PushConsumerImpl extends _PushConsumerImplBase {
    String[] event_data;
    public int id = 0;  //id of this pushConsumer, needed for disconnection from ASManagementAgent

    public PushConsumerImpl()
    {
        System.err.println("\n\nRegionManagementAgentApplet: PushConsumer created");
    }

    public void push(org.omg.CORBA.Any any)
```

```
    {
        event_data = new String[5];
        CosNotification.StructuredEvent se = CosNotification.StructuredEventHelper.extract(any);
        event_data[0] = se.header.fixed_header.event_type.domain_name;
        event_data[1] = se.header.fixed_header.event_type.type_name; //AgentUp/AgentDown/agentSystemUp/agentSystemDown
        event_data[2] = se.filterable_data[0].value.extract_string();  //agent_type
        event_data[3] = se.filterable_data[1].value.extract_string();  //agent_name/agent_system_ame
        event_data[4] = se.filterable_data[2].value.extract_string();  //event_channel_name
        if (event_data[1].equals("agentSystemUp")) {  //new agent system was started
            //add the name of the agent system to the as_names vector and to the tree
            as_names.addElement(event_data[3]);
            root.add(new DefaultMutableTreeNode(as_names.lastElement()));
            //refresh the tree
            treemodel.reload();
        } else if (event_data[1].equals("agentSystemDown")) {  //agent system was shut down
            //select another node if selected_node is agentsystem to be removed
            DefaultMutableTreeNode new_node = new DefaultMutableTreeNode();
            if (((String) selected_node.getUserObject()).equals(event_data[3])) {
                new_node = (DefaultMutableTreeNode) root.getChildBefore(selected_node);
                if (new_node == null) {
                    //selected_node is already first child
                    new_node = (DefaultMutableTreeNode) root.getChildAfter(selected_node);
                }
                tree.setSelectionPath(new TreePath(new_node.getPath()));
            }

            //remove the agent system from as_names Vector
            as_names.removeElement(event_data[3]);

            //remove agent system from the tree
            DefaultMutableTreeNode as_node;
            for(int i = 0; i < root.getChildCount(); i++) {  //scanning the tree for the node to remove
                as_node = (DefaultMutableTreeNode) root.getChildAt(i);
                if (((String) as_node.getUserObject()).equals(event_data[3])) { //node found
                    root.remove(as_node);
                    break;
                }
            }
            //refresh tree
            treemodel.reload();
        } else if (event_data[1].equals("AgentUp")) {  //new agent is up
            //if agent system currently shown in agentTable equals the agent system where the new agent
            //has been started up then add new agent to agentTable
            if (((String) selected_node.getUserObject()).equals(event_data[0])) {
                myModel.addValue(event_data[3]);
            }
        } else if (event_data[1].equals("AgentDown")) {  //agent is down
            //if agent system currently shown in agentTable equals the agent system where the agent
            //has been terminated then remove agent from agentTable
            if (((String) selected_node.getUserObject()).equals(event_data[0])) {
                myModel.rmValue(myModel.whereis(event_data[3]));
            }
        }
    }

public void disconnect_push_consumer() {
    try {
        // Initialize the ORB.
        //org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
        // Initialize the BOA.
        //org.omg.CORBA.BOA boa = orb.BOA_init();

        System.err.println("\n\nDisconnecting push_consumer...\n");
        //boa.exit_impl_ready();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

/**
 * TableModel managing the list of loaded agents in the selected agent system.
 * <br>One column is defined, which contains the names of the currently loaded agents in the
 * agent system selected in the tree.
 *
 * @see AbstractTableModel
 * @author Stefan Gerber
 */

class AppletTableModel extends AbstractTableModel {
    String[] columnNames = {"Agents"};
    Vector data;

    public void AppletTableModel()
    {
        data = new Vector(0,1);
    }

    public int getColumnCount()
{
    return columnNames.length;
}

public int getRowCount()
{
    return data.size();
}

public String getColumnName (int col)
{
    return columnNames[col];
}

public Object getValueAt(int row, int col)
{
    return (String) data.elementAt(row);
}

public void setValueAt(Object value, int row, int col)
{
    data.setElementAt((String) value, row);
    fireTableCellUpdated(row, col);
}

public void addValue(String value)
{
    data.addElement(value);
    fireTableRowsInserted(data.size()-1, data.size()-1);
}

public void rmValue(int row)
{
    data.removeElementAt(row);
    fireTableRowsDeleted(row, row);
}

//returns the row number where agent "an" is located
public int whereis(String an) {
    for (int i=0; i<data.size(); i++) {
        if (an.equals((String) data.elementAt(i))) return i;
    }
    return (-1);
}
}
}
```

Anhang H

Verwendete Entwicklungswerkzeuge

- Sun/Javasoft JDK 1.1.7
- Sun/Javasoft Swing/JFC 1.1 Beta 2
- Sun/Javasoft Hotjava 1.1
- Inprise/Visigenic VisiBroker 3.0 for Java

Literaturverzeichnis

[Bran 99] BRANDT, R.: *Interoperabilität verschiedener CORBA-ORBs in einem mobilen Agentensystem.* TUM, 1999.

[CORBA 2.2] *The Common Object Request Broker: Architecture and Specification.* OMG Specification Revision 2.2, Object Management Group, Februar 1998.

[Flan 96] FLANAGAN, D.: *Java in a Nutshell.* O´Reilly & Associates, Inc., 1996.

[Inp 97] INPRISE/VISIGENIC SOFTWARE, INC.: *Visigenic VisiBroker for Java Reference Manual Version 3.0*, 1997.

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

[Naug 96] NAUGHTON, P.: *Java Handbook.* Osborne McGraw-Hill, 1996.

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

[Sun 98a] SUN MICROSYSTEMS, INC.: *Java Development Kit 1.2 Documentation*, 1998.

[Sun 98b] SUN MICROSYSTEMS, INC.: *The Java Tutorial*, 1998.