Agenten

und die Object Management Group

14.05.2007

Object Management Group

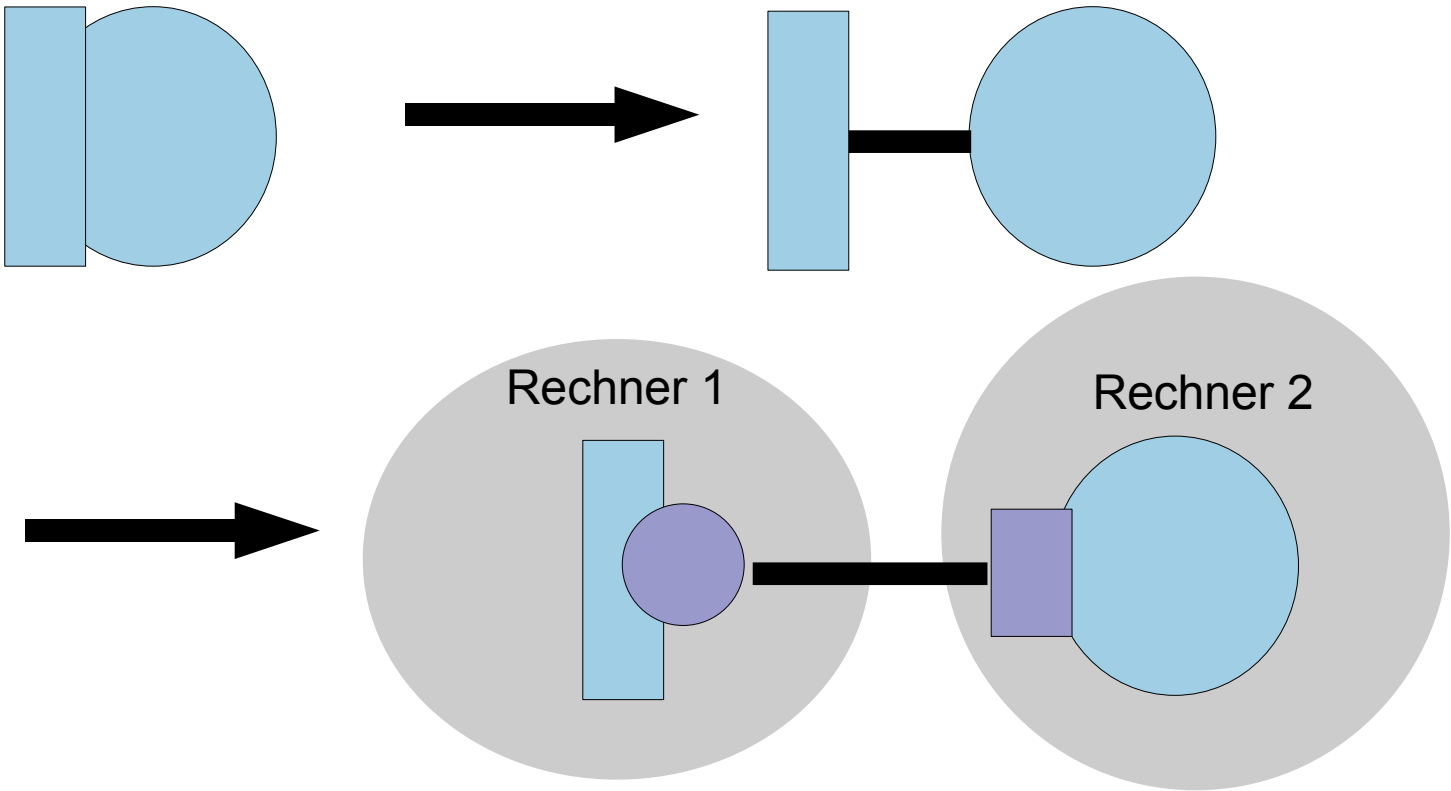
- www.omg.org
  - Nichtkommerzielles Konsortium
  - Zielsetzung: Schaffung von IT-Spezifikationen zur Sicherung der Interoperabilität von Anwendungen
- Beispiele von akzeptierten Standards
  - CORBA, OMG IDL, IIOP
  - Modellierung
    - UML
    - MDA (Model-driven architecture)
    - MOF (Meta-Object Facility)
    - CWM (Common Warehouse Metamodel)
  - XMI (XML Metadata Interchange)

CORBA

- Common Object Request Broker Architecture
- Aufgabe
  - Interoperabilität zwischen beliebigen Anwendungen auf unterschiedlichen Netzknoten
- Basis: Objektorientierte Programmierung
- Maximale Interoperabilität durch Plattformunabhängigkeit
  - Hardwareunabhängigkeit
  - Anwendungs- und Herstellerunabhängigkeit
  - Betriebssystemunabhängigkeit
  - Programmiersprachenunabhängigkeit

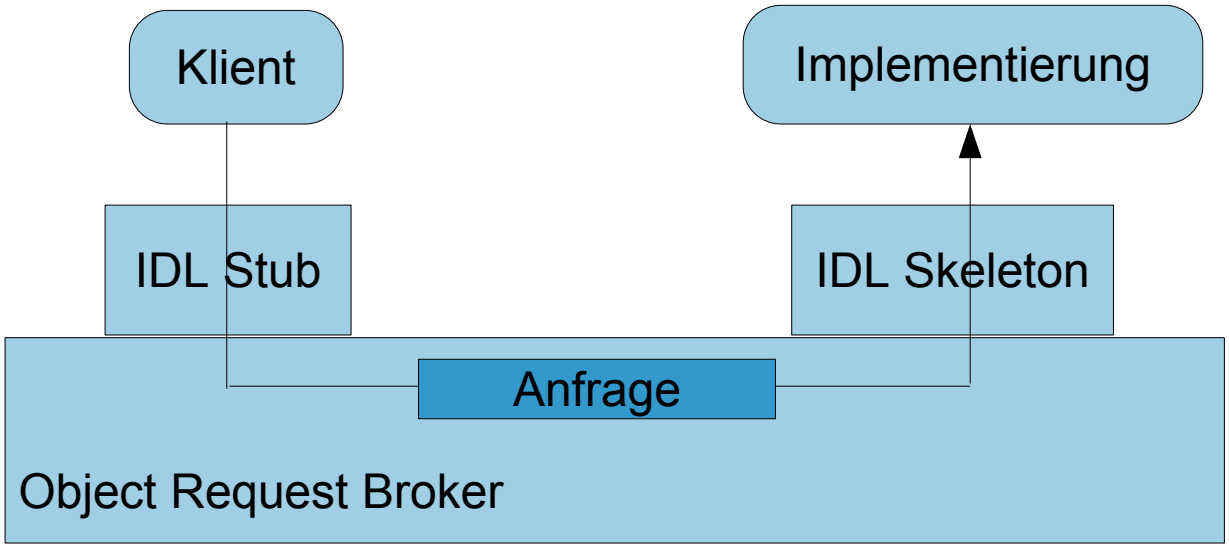
Hauptprinzip von CORBA

- Hauptprinzip
  - Trennung von Schnittstelle und Implementierung



CORBA

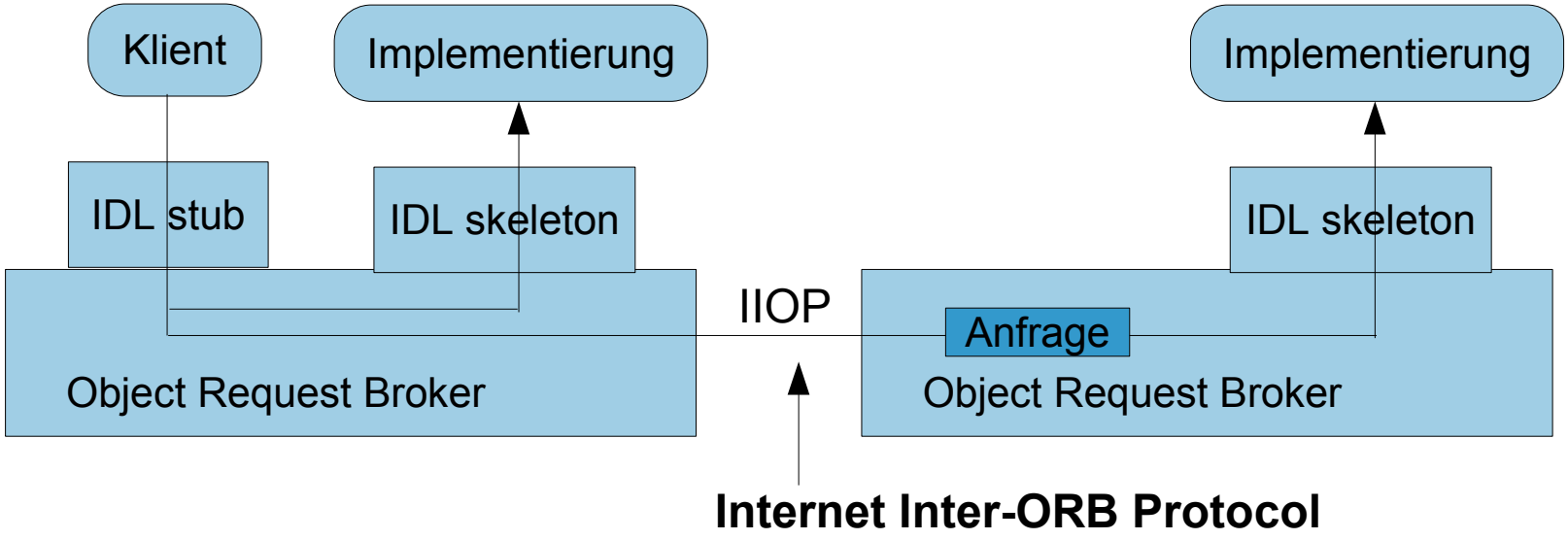
- Standarddiagramm
  - schematische Darstellung



- ORB sorgt für Verteilungstransparenz und leitet Anfragen weiter
  - Realisierung z.B. als zentraler Server oder als Bibliotheksmodul (gebunden an jeden einzelnen Teilnehmer)

CORBA

- Verteilung auf verschiedene ORBs



- Lokationstransparenz
  - Klient merkt nicht, dass das Objekt auf einem anderen Rechner läuft
  - Möglichkeiten zur Lastverteilung durch den ORB

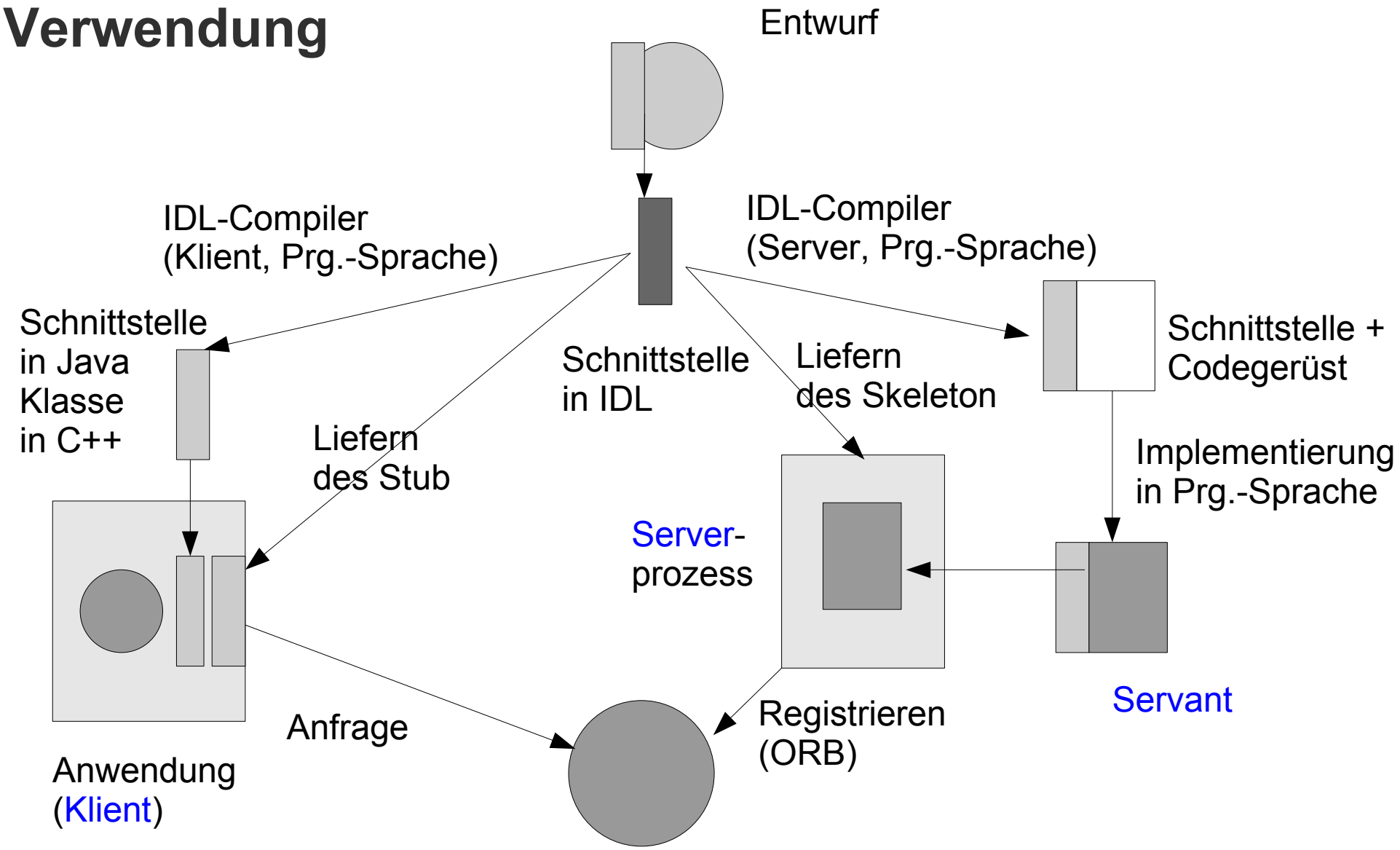
CORBA

- Schnittstelle
  - beschrieben durch OMG Interface Definition Language (OMG IDL)
  - keine Festlegung auf spezifische Programmiersprache
  - Ähnlichkeiten mit DCE IDL, aber im Kontext von C++

```
module BANK {
  interface account {
    readonly attribute float balance;
    void makeLodgement (in float f);
    void makeWithdrawal (in float f);

    struct description {
      string name;
      float balance;
    };
    any describe ();
  };
 ...
};
```

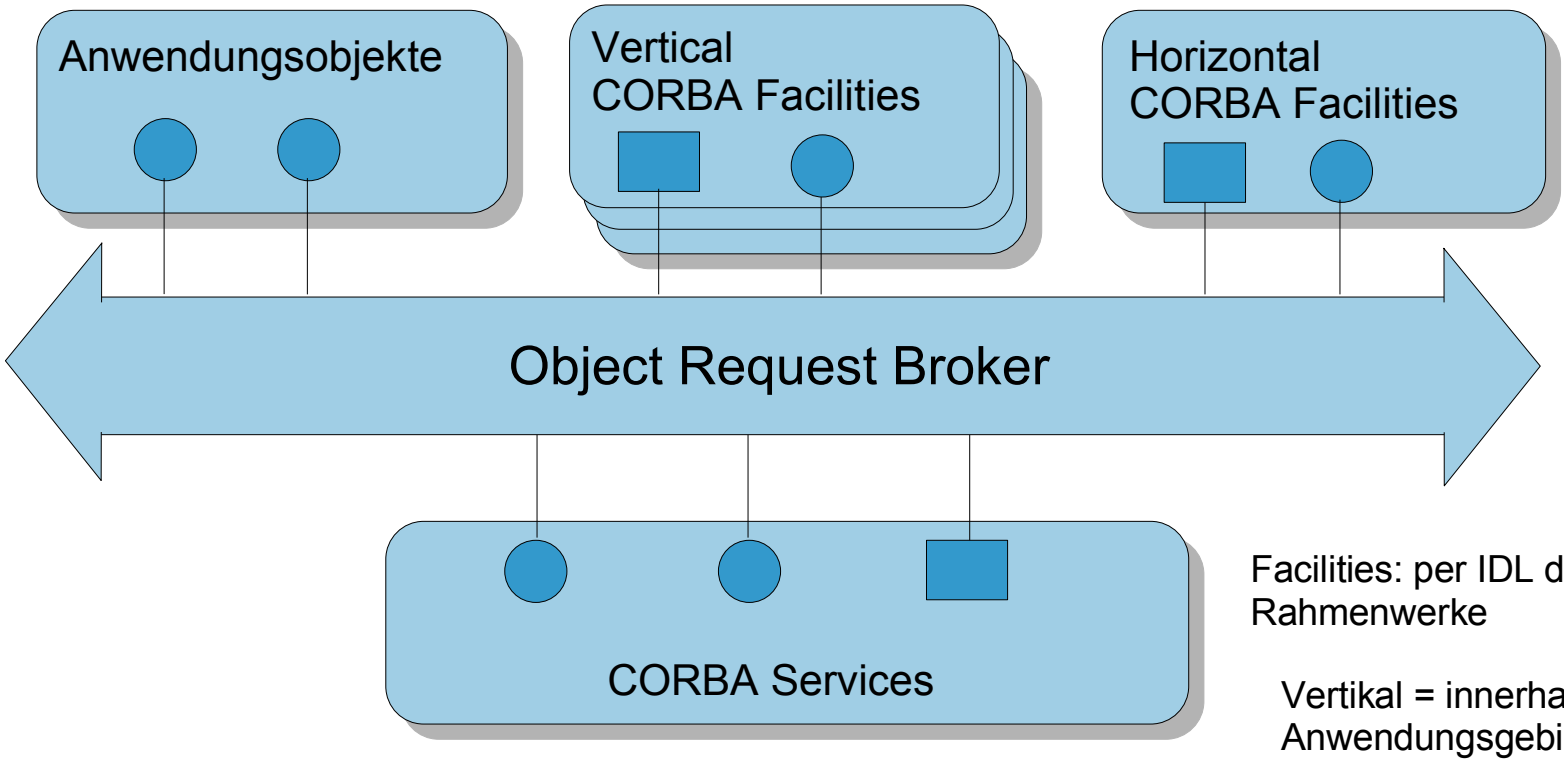
Verwendung


Entwurf
IDL-Compiler
(Klient, Prg.-Sprache)
IDL-Compiler
(Server, Prg.-Sprache)
Schnittstelle
in Java
Klasse
in C++
Schnittstelle
in IDL
Liefern
des Skeleton
Schnittstelle +
Codegerüst
Liefern
des Stub
Implementierung
in Prg.-Sprache
Server-
prozess
Servant
Anfrage
Registrieren
(ORB)
Anwendung
(Klient)

CORBA

- Unterscheidung zwischen Dienstanbieter und Diensterbringer
- CORBA-Objekt
  - „virtuelles Objekt“
  - kann von einem ORB lokalisiert werden und als Zielpunkt von Klientenanfragen verwendet werden
  - Vergleichbar mit abstraktem Objekt in C++ oder Interface in Java
- Servant (Diensterbringer)
  - Reale, implementierte Programmentität, die die Funktion eines CORBA-Objekts realisiert und auf einem Server läuft
  - In objektorientierten Sprachen: Objekte aus Klassen
  - In anderen (prozeduralen) Sprachen: Sammlung von Prozeduren
  - kann in beliebiger Programmiersprache implementiert sein

Object Management Architecture



Facilities: per IDL definierte Rahmenwerke

Vertikal = innerhalb eines Anwendungsgebiets auf verschiedenen Ebenen

Horizontal = auf gleicher Ebene über Anwendungsgebiete hinweg

CORBA-Dienste

- CORBA-Dienste
  - Collection service
  - Concurrency service
  - Enhanced view of time service
  - Event service
  - Externalization service
  - Naming service
  - Licensing service
  - Notification service
  - Persistent state service
  - Property service
  - Query service
  - Relationship service
  - Security service
  - Time service
  - Trading Object service
  - Transaction service

CORBA und Softwareagenten

- 1997: Proposal an die OMG von FhG FOKUS (damals GMD), IBM u.a.
- Idee der Schaffung einer Spezifikation für die Interoperabilität von Mobile-Agenten-Systemen
- **Mobile Agent Facility** (MAF)
  - ursprünglicher Titel des Vorschlags, dann MASIF (Mobile Agent System Interoperability Facility)
  - dann offiziell MAF V1.0 (2000)

> Die Unterschiede zwischen den Mobile-Agenten-Systemen verhindern die Interoperabilität und die rasche Verbreitung der Agententechnologie [...] Um sowohl die Interoperabilität als auch die Vielzahl der Systeme zu fördern, müssen einige Aspekte der Technologie mobiler Agenten standardisiert werden.

MAF-Spezifikation, Einleitung

Agenten und CORBA

- Agenten müssen keine CORBA-Objekte sein
  - Agenten können CORBA-Objekte sein
  - Agenten dürfen CORBA nutzen
- Die Infrastruktur muss über CORBA angesprochen werden können
  - entsprechende Schnittstellen implementieren
  - beim ORB anmelden
  - ggf. auch beim Namensdienst (CosNaming)

Konzepte der MAF-Spezifikation

Ziele der Spezifikation

- Grundlegendes Modell
  - Interoperabilität
  - Grundkonzepte: Was sind Agenten? Welche Strukturen unterscheidet man? Welche Aufgaben übernimmt eine Infrastruktur?
  - Agenteninteraktion
  - Grundfunktionen eines Agentensystems
- Welche Dienste stehen aus CORBA zur Verfügung?
- Wie müssen Schnittstellen gestaltet werden?

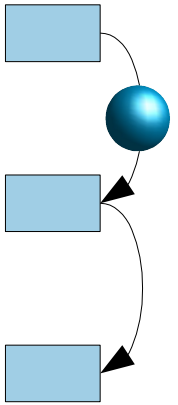
Interoperabilität

- Was soll standardisiert werden?
  - Agentenmanagement
  - Agententransfer
  - Agentennamen, Agentensystemnamen
  - Typen von Agentensystemen
  - Lokationssyntax

Voraussetzung ist eine gewisse „Ähnlichkeit“ der Agentensysteme; sonst ist nur minimale Interoperabilität gegeben

MAF-Spezifikation

Interoperabilität – auf später verschoben



- Sicherheitsprobleme
  - Single-Hop: Agent führt nur eine Migration aus
  - Multi-Hop-Agenten: Inhärente Komplexität
  - Die meisten Ansätze bewältigen nur Single-Hop
- Sprachenwirrwarr
  - Tcl, Java
  - Konvertierung von Agentenrepräsentationen zu schwierig
  - Ggf. Brücken bauen, wenn die jeweiligen internen Repräsentationen ähnlich genug sind (z.B. wenn die Serialisierungen von Code und Zustand konvertierbar sind)

... abwarten, bis die Industrie reifer ist oder De-facto-Standards errichtet wurden ...

MAF-Spezifikation

Interoperabilität – keine Standardisierung

- Interoperabilität bei verschiedenen Programmiersprachen
  - in CORBA generell möglich, da nur Daten verschickt werden
  - zu schwierig für mobile Agenten als „aktive Objekte“, die an verschiedenen Knoten zur Ausführung kommen sollen
  - MAF nimmt daher an, dass die Systeme in derselben Programmiersprache geschrieben sind, aber eventuell von verschiedenen Autoren
- Lokaler Agentenbetrieb
  - Interpretierung, Serialisierung, Ausführung
  - Man verzichtet auf Vorschriften, wie ein Agentensystem aufgebaut werden soll
  - damit Akzeptanz erhöhen

Interoperabilität – keine Standardisierung

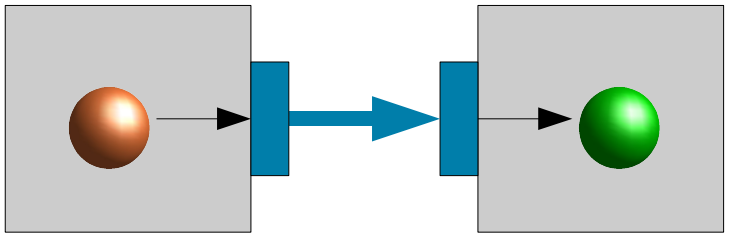
- Kommunikation

> Agentenkommunikation wird in der MAF-Spezifikation nicht betrachtet. Sie wird im Rahmen von CORBA ausführlich als Objektkommunikation behandelt.

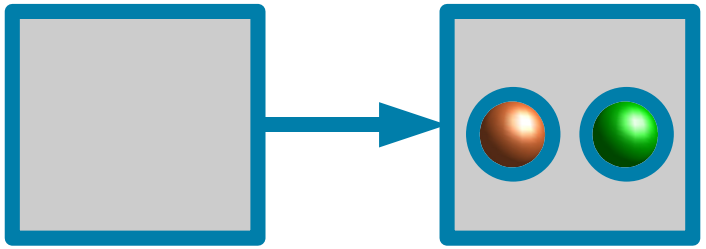
MAF-Spezifikation

Ziele der Standardisierung

- Interoperabilität
  - durch Kommunikation
  - durch Migration



Kommunikation
über standardisierte
Schnittstelle



Agentensysteme mit
standardisierten Schnittstellen
erlauben Migration

Standardisierung des Managements

- Szenario
  - Administrator verwaltet verschiedene Agentensysteme über Standardprozeduren
- Beispiele
  - Erzeugen eines Agenten über dessen Klassenname
  - Anhalten des Agententhreads
  - Fortsetzen der Ausführung
  - Terminierung eines Agenten

Standardisierung des Konzepts eines Agenten

- Agent

Ein Agent ist ein Computer-programm, das im Namen einer Person oder Organisation selbständig handelt. Gegenwärtig sind Agenten meist in einer Interpretersprache aus Portabilitätsgründen geschrieben. Jeder Agent hat seinen eigenen Ausführungsfluss, sodass Aufgaben in eigener Initiative erledigt werden können.

MAF-Spezifikation

- Stationärer Agent
  - Agent, welcher sich nicht fortbewegt
  - „Wenn ein solcher Agent Informationen benötigt, bedient er sich typischerweise spezieller Mechanismen der Kommuni-kation wie RPC“
- Mobiler Agent
  - Agent, welcher seine Ausführungsumgebung ändert
  - „Ein mobiler Agent hat größere Fähigkeiten und Erfordernisse, als die derzeit erhältlichen Systeme verteilter Objekte anbieten.“

Standardisierung von Namen

- Namen von Agenten und deren Systemen müssen standardisiert werden
  - Das Management von Agenten erfordert insbesondere die Benennung des fraglichen Agenten
- Weitere Vorteile einer Namensstandardisierung
  - Agentensysteme können bestimmen, ob sie die Agenten unterstützen
  - Agenten können sich gegenseitig identifizieren
- Lokationssyntax sollte standardisiert werden
  - Agenten können spezifische Informationen des Zielsystems erfahren
  - Namensautorität sollte definiert werden

Standardisierte Namen

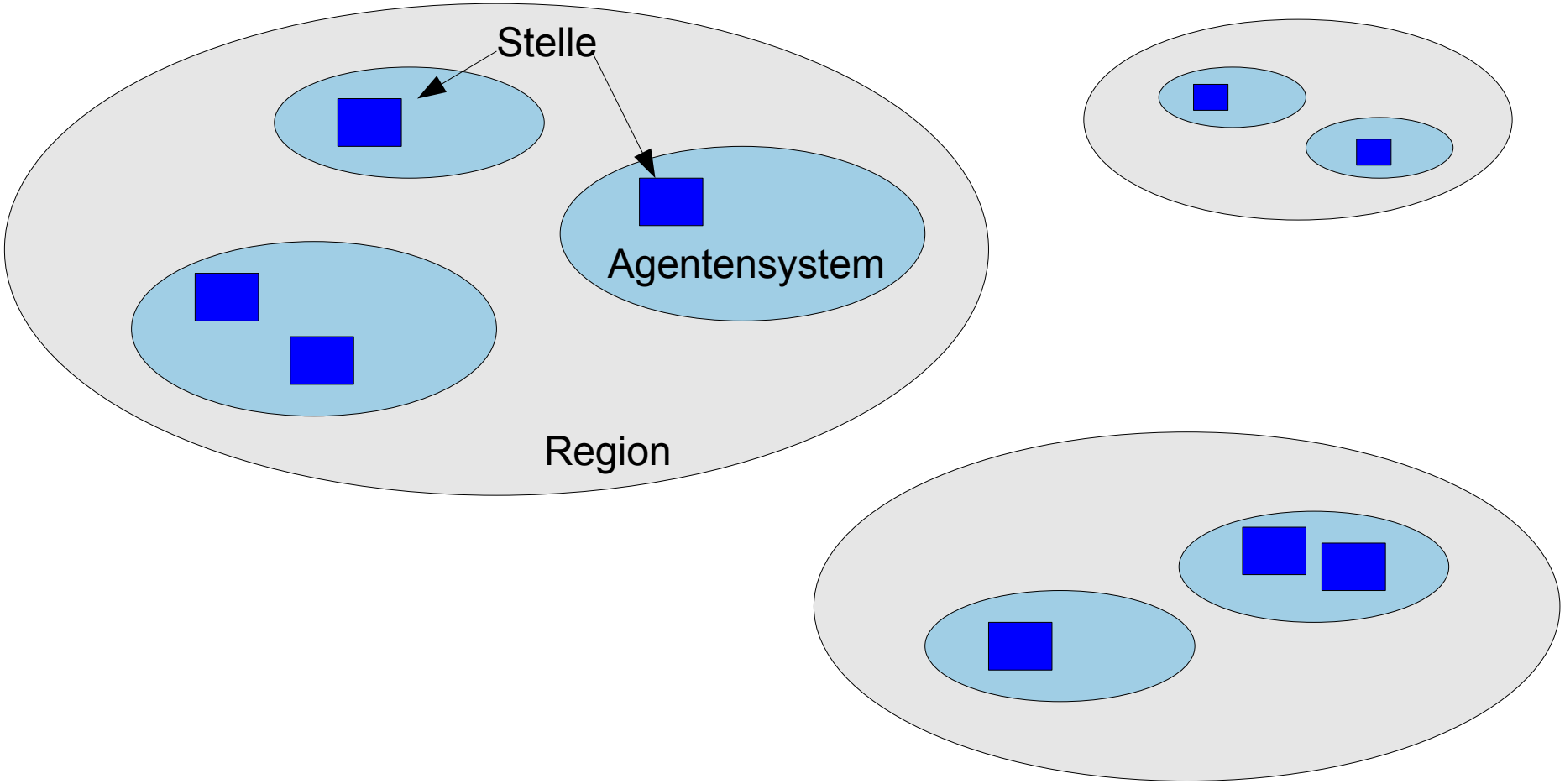
- Namen
  - Bezeichnen Agenten, benötigt für Managementvorgänge
  - bestehen aus
    - Autorität (wer den Agenten erzeugt)
    - Identität (eindeutig im Bereich der Autorität)
    - Agentensystemtyp

  } global eindeutig, referenziert genau einen Agenten, d.h. man kann den Namen als Schlüssel zum Suchen verwenden

- Autoritäten
  - Agentenautorität: Person oder Organisation, in deren Namen der Agent tätig wird
  - Stellenautorität: Person oder Organisation, in deren Namen die Agentenstelle betrieben wird

Zuordnung der Ressourcen und Zugriffskontrolle auf Basis realer Personen / Organisationen (als *Prinzipale*)

Struktur im Überblick



Struktur

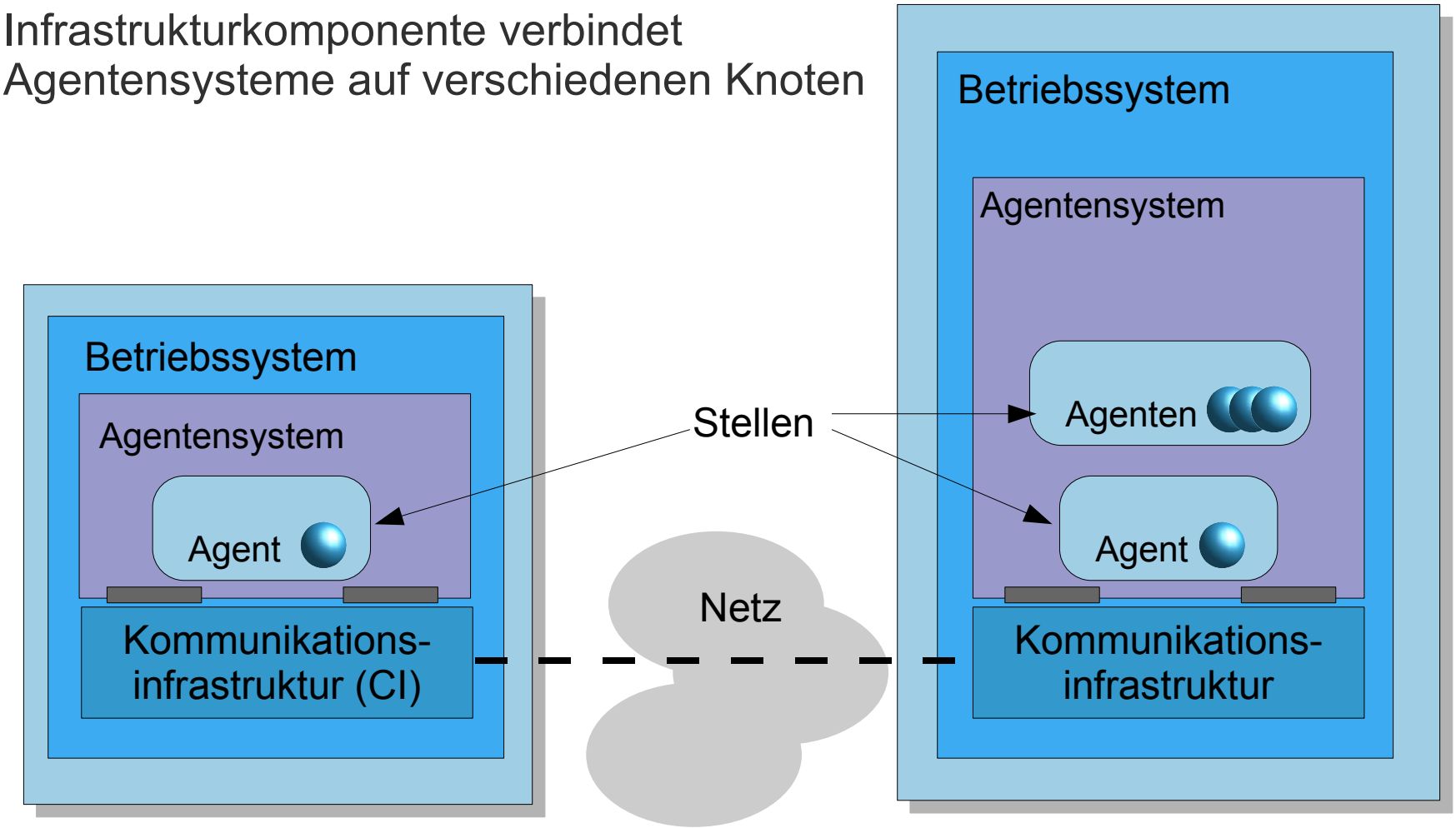
- Agentensystem
  - Plattform, die Agenten erzeugen, interpretieren, ausführen, transferieren und beenden kann
  - Jedes Agentensystem kann eine oder mehrere Stellen beinhalten
  - Jeder Rechner kann eines oder mehrere Agentensysteme betreiben
  - eindeutig identifiziert durch Name und Adresse
- Ort (Lokation) eines Agenten
  - Agenten befinden sich auf Stellen (Places)
  - Stellen sind Teil eines Agentensystems
  - Lokationsadresse besteht aus Stellenname, Agentensystemname
  - ggf. auch „Standardstelle“ des Agentensystems, wenn nicht spezifiziert

Stellen

- Start- und Zielpunkte einer Agentenmigration
- Jedes Agentensystem birgt eine oder mehrere Stellen
  - Definition einer Standardstelle (Default place)
  - Wenn keine Strukturen wie Stellen definiert sind, wird eine Pseudo-Standardstelle angenommen
- Migration
  - innerhalb eines Agentensystems problemlos
  - zwischen Agentensystemen, wenn gleiches Agentenprofil unterstützt
- Stellen sind lokalisiert über Stellenname und Adresse des Agentensystems

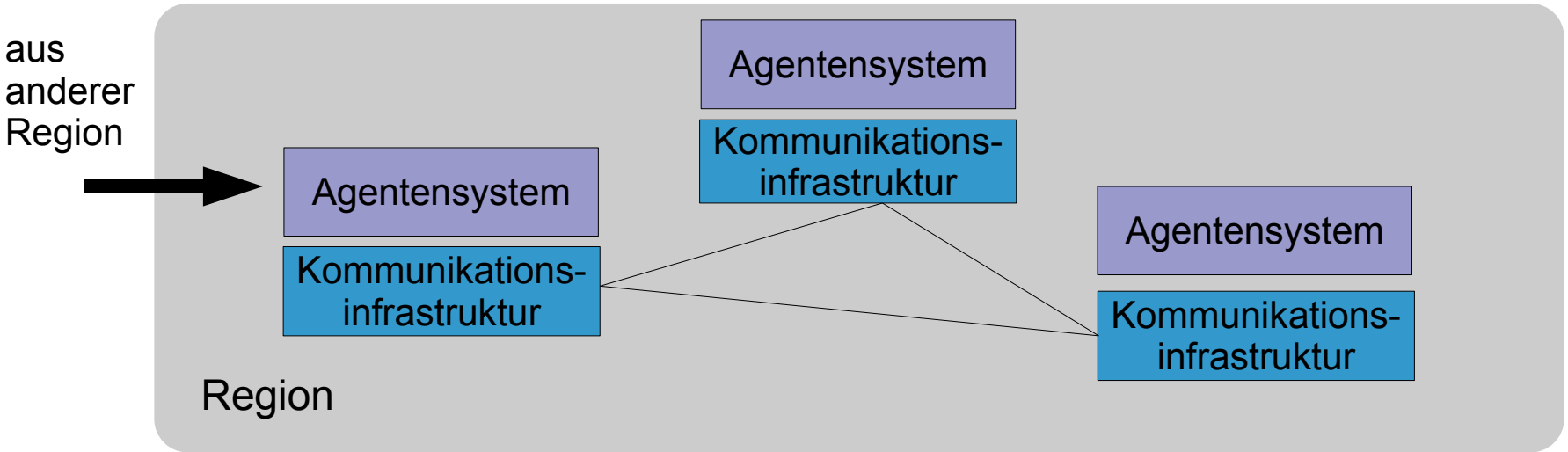
Standardisierung der Infrastruktur

- Infrastrukturkomponente verbindet Agentensysteme auf verschiedenen Knoten



Regionen

- Ansammlung von Agentensystemen
  - gleiche Autorität
  - ggf. unterschiedliche Agentensystemtypen
- Alle Agentensysteme sind untereinander direkt verbunden
  - jedes System ist unmittelbar erreichbar



Einsatz von Regionen

- Zuordnung einer Menge von Agentensystemen zu einer Autorität (z.B. einem Benutzer)
- Nutzen für die Lastverteilung
- Abstraktion der Verteilung der Agentensysteme
  - Ein Anfrager kann sich von außen an die Region wenden
  - Agenten können sich auf einer der Stellen der Region aufhalten
  - nur notwendig, Regionsadresse und Agentenname zu kennen, um mit dem Agenten zu interagieren

Regionen

- Regionen sind untereinander verbunden
  - spezielle Zugangspunkte je Region
- Auch Nicht-Agenten-Systeme können mit einer Region interagieren
- Zugänge
  - Agentensysteme, die von außen erreichbar sind
  - ähnlich Firewall
- Rechte
  - Zuweisung an Agenten basiert auf der Autorität, welche die Region betreibt

Standardisierung von Grundfunktionen

- Generische Funktionen
  - Agententransfer: Senden/Empfangen von Agenten sowie Klassentransfer
  - Erzeugen von Agenten
  - Global eindeutige Agentennamen und Lokationen
  - Unterstützung des Regionskonzepts
  - Auffinden von mobilen Agenten
  - Gewährleistung einer sicheren Umgebung (siehe Sicherheit)

Grundfunktionen

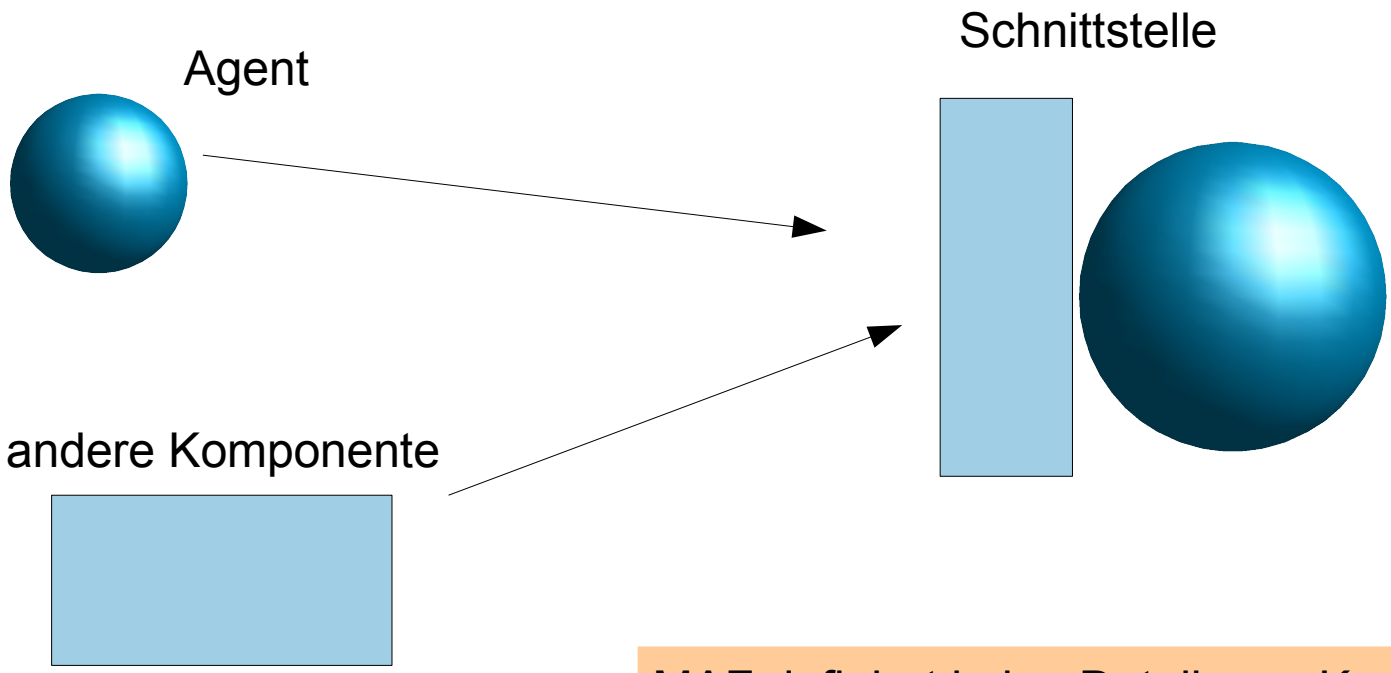
- Erzeugen eines Agenten auf Anfrage
  - Starten des Threads
  - Instanziierung der Agentenklasse
  - Zuweisung eines global eindeutigen Bezeichners
  - Starten der Ausführungen innerhalb des Threads

- Generierung und Verwendung eindeutiger Bezeichner
  - für das Agentensystem
  - für jede Stelle
  - für jeden Agenten

Eigene Ablaufkontrolle (schon Autonomie?)

Kommunikation

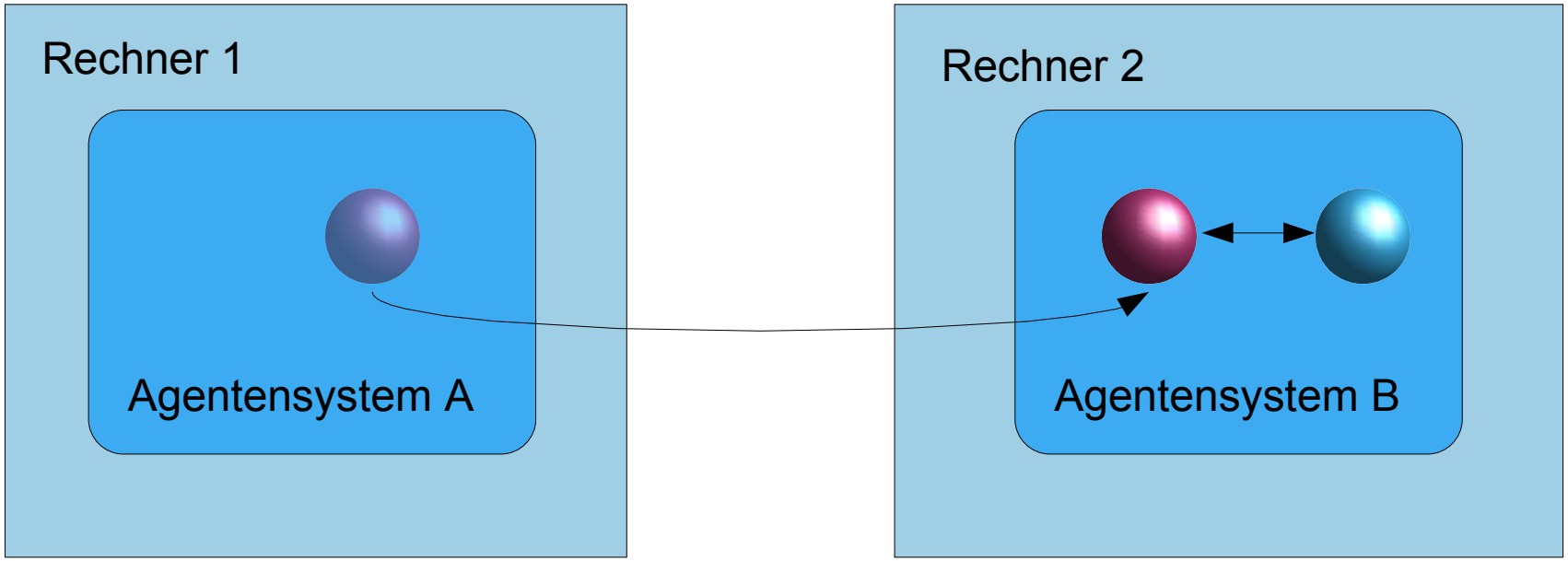
- Kommunikation mit Agenten



MAF definiert keine Details zur Kommunikation, nennt jedoch explizit den Methodenaufruf auf Agenten

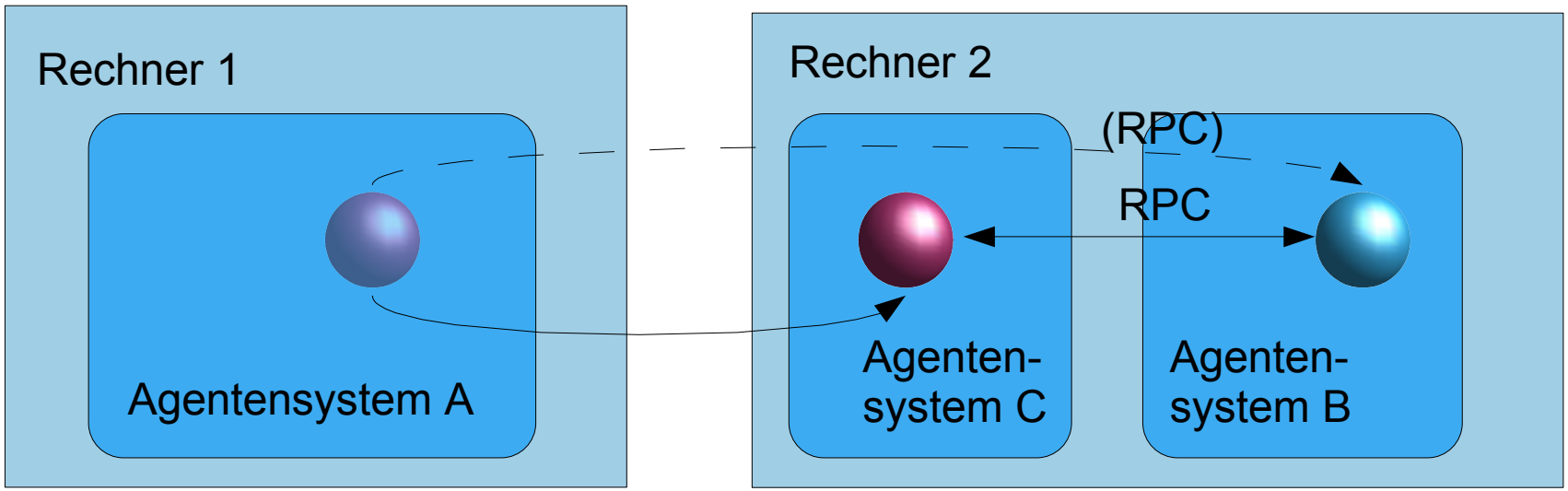
Agentensystemtypen

- Typgleiche Systeme
  - gleiches Agentenprofil unterstützt (Sprache usw.)
  - keine Interoperabilitätsprobleme



Agentensystemtypen

- Typungleiche Systeme
  - falls es ein typgleiches System auf dem entfernten Rechner gibt: Lokalität ausnutzbar, lokaler RPC
  - sonst entfernten RPC



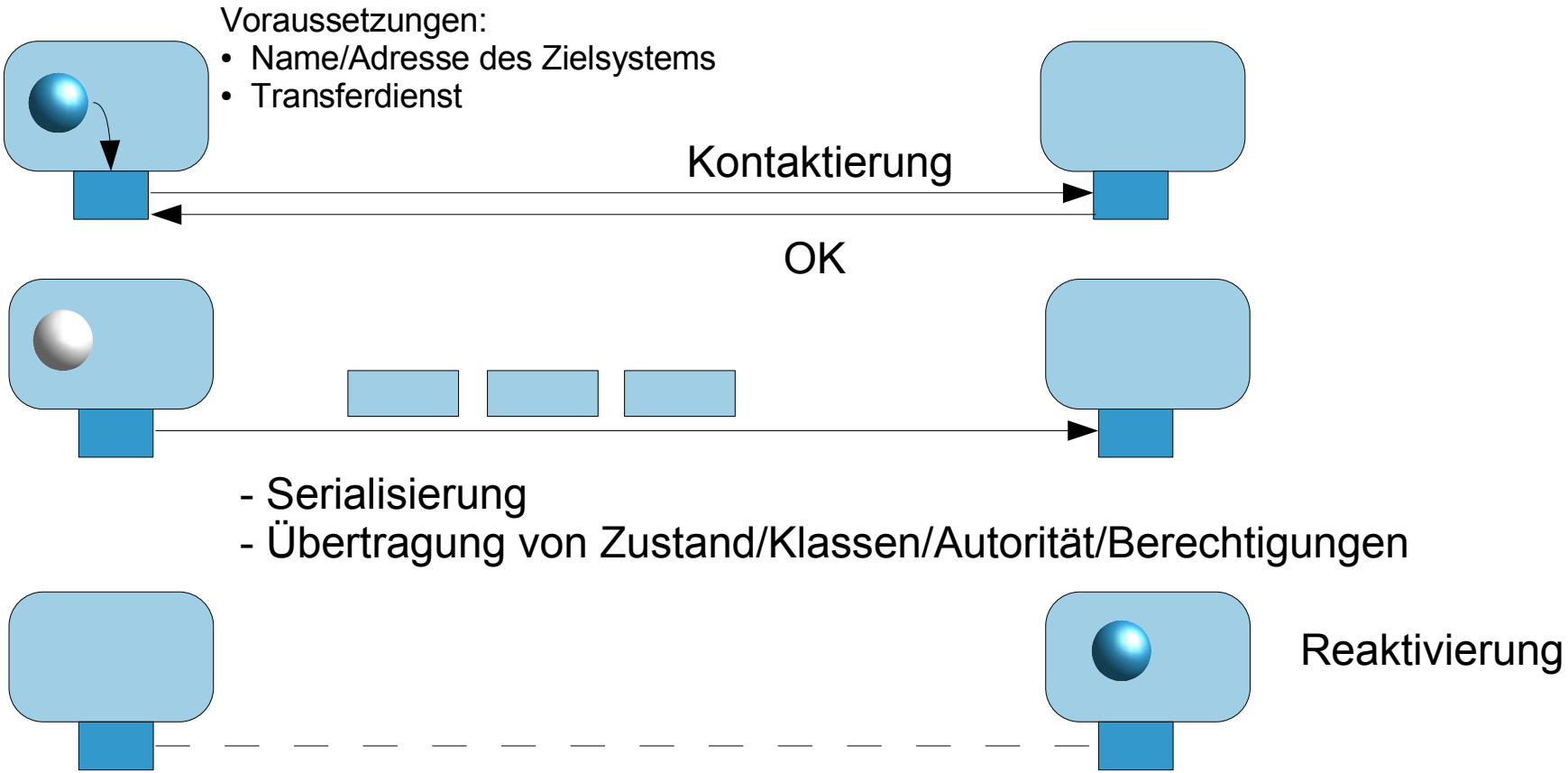
Mobilität aus Sicht der MAF

- Vorteil der Mobilität
  - Lokale Kommunikation kostet weniger als entfernte Kommunikation
  - sonst etwa aufwändiger Datenaustausch via RPC
  - Lokale Kommunikation kann besser abgesichert werden
- Agenten sollten daher die Möglichkeit haben, ihren Ausführungsort zu verlagern

MAF möchte insbesondere die Interoperabilität zwischen verschiedenen Systemen ermöglichen, sodass etwa Agenten zwischen den Systemen ausgetauscht werden können.

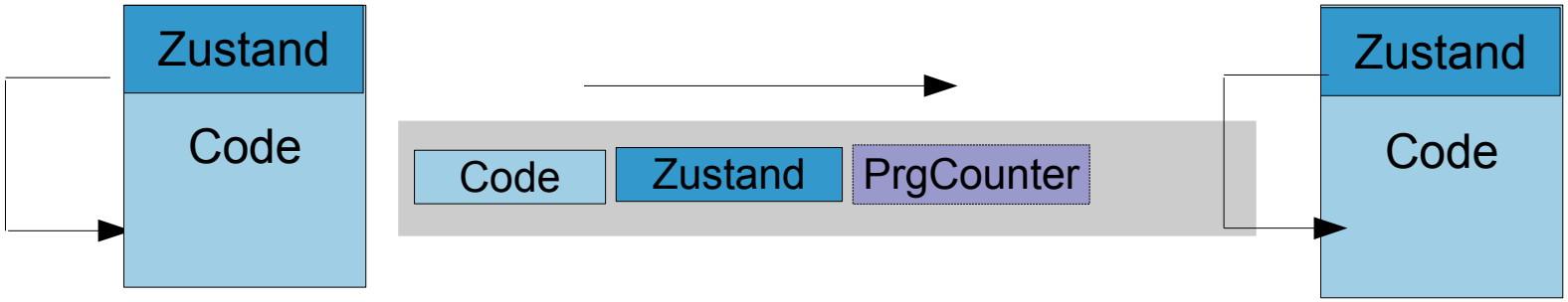
MAF-Modell der Mobilität

- Agententransfer



MAF-Modell der Mobilität

- Ausführungszustand
  - Agentencode und Zustand müssen bei Wanderung übertragen werden.*
  - Ausführungszustand betrifft Werte der Speicherzellen (Zustand) sowie (falls möglich) den Punkt der Ausführung, an dem diese unterbrochen wurde



* setzt voraus, dass das System starke Mobilität beherrscht, was bei Java nicht der Fall ist, aber allgemein nicht unmöglich ist.

MAF-Modell der Mobilität

- Serialisierung/Deserialisierung
  - Vorgang der Konvertierung eines Agenten in eine transportable Form (als Bytestrom) sowie die Umkehrung (Rekonstruktion)
  - bekannt aus Java
  - andere Sprachen müssen ggf. eigene Verfahren definieren (oder speziellen CORBA-Dienst nutzen)
- Codebasis
  - Prinzip bekannt aus Java
  - Lokation, an der der Agentencode gespeichert ist
  - z.B. Agentensystem oder Webserver
  - Agentensystem in dieser Rolle wird „Class Provider“ (Klassenlieferant) genannt

MAF-Modell der Mobilität

- Initiierung
  - Agent muss Ziel identifizieren (ggf. „Standardstelle“ der Zielregion)
  - Anfrage an das lokale System, den Transfer einzuleiten (über interne Schnittstellen)
- Wenn Transfer genehmigt ist:
  - Agent anhalten
  - Identifizieren, welcher Teil des Zustands zu übertragen ist
  - Agentenzustand und Klassen serialisieren
  - Transportkodierung anwenden
  - Klient authentifizieren
  - Agent transferieren

Entsprechend bei Zielstelle:
- Entgegennahme
- Deserialisierung
- Wiederaufnahme der Ausführung

MAF-Modell der Mobilität

- Transfer der Klassen
  - Notwendig für objektorientierte Systeme, um Objektinstanzen zu erzeugen
- Fälle, in denen Klassentransfer benötigt wird
  - Entfernte Agentenerzeugung
  - Agententransfer
  - Erzeugung von Gebrauchsobjekten während der Ausführung des Agenten an der Zielstelle
  - Fehlende Klassen müssen in der Regel vom Ursprungssystem nachgeladen werden (siehe auch Situation bei Java-Applets)

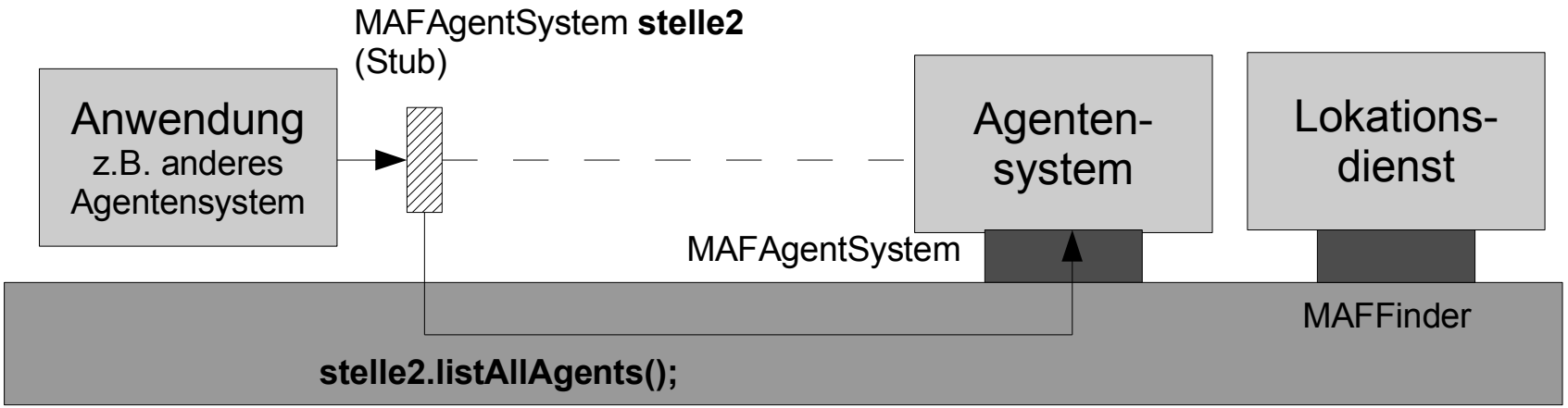
MAF-Modell der Mobilität

- Methoden des Klassentransfers, die vom Modell unterstützt werden
  - Alle Klassen auf einmal übertragen
    - Vermindert Nachladen von Klassen
    - Höherer Bandbreitenverbrauch, missachtet evtl. lokal vorliegende Klassen (Cache)
  - Transfer der Agentenklassen, weitere Gebrauchsklassen bei Bedarf
    - effizienter als der erste Vorgang
    - jedoch darauf angewiesen, jederzeit die Ursprungsstelle anzusprechen (Gefahr bei Abkopplung, bspw. Laptops!)
  - Bei Transfer Bedarfsladen, bei entfernter Erzeugung Kompletttransfer
    - dann sind die notwendigen Klassen auf jeden Fall ein Mal im System
  - Verschicken einer Bedarfsliste notwendiger Klassen
    - Nutzt Caches an der entfernten Stelle

Wie wird ein System MAF-konform?

Wie wird ein System MAF-konform?

- Zwei Schnittstellen
  - **MAFAgentSystem**: Agentenmanagement
    - Starten, Stoppen, Klassen holen, Zustand, Infos, Listen
  - **MAFFinder**: Aufsuchen
    - Finden von Agenten, Agentensystemen, Stellen
- MAF bietet dazu eine IDL-Definition an



Wie wird ein System MAF-konform?

- Übliches Vorgehen
  - IDL-Beschreibung besorgen; mit IDL-Compiler Stubs und Schnittstellen für die jeweilige Programmiersprache erzeugen lassen
  - Dienste, die eine solche Schnittstelle aufweisen und über CORBA angesprochen werden sollen, müssen im ORB registriert werden
  - In den Anwendungen können die entsprechenden Funktionen in der üblichen Art durch Stub-Aufruf genutzt werden

MAF-IDL: Übersicht der Datentypen

```
typedef sequence<octet>          OctetString;
typedef short                    LanguageID;
typedef short                    AgentSystemType;
typedef short                    Authenticator;
typedef short                    SerializationID;
typedef any                      Property;
typedef sequence<octet>          Arguments;
typedef string                   Location;
```

```
typedef sequence<OctetString>          OctetStrings;
typedef OctetString                    Authority;
typedef OctetString                    Identity;
typedef sequence<SerializationID>      SerializationIDList;
typedef sequence<Property>             PropertyList;
typedef sequence<Name>                 NameList;
typedef sequence<LanguageMap>          LanguageMapList;
typedef sequence<ClassName>            ClassNameList;
typedef sequence<Location>             Locations;
```

```
enum AgentStatus {
        CfMAFRunning, CfMAFSuspended, CfMAFTerminated;
};
```

```
struct Name {
        Authority           authority;
        Identity            identity;
        AgentSystemType agent_system_type;
};
```

```
struct AuthInfo {
        boolean             is_authenticated;
        Authenticator       authenticator;
};
```

```
struct LanguageMap {
        LanguageID                  language_id;
        SerializationIDList         serializations;
};
```

```
struct ClassName {
        string              name;
        OctetString         discriminator;
};
```

Location

- Adresse eines CORBA-Objekts
  - dient der Referenzierung auf CORBA-Ebene
- Nutzung von CORBA-Namensdienst (*CosNaming*) bei abstrakten Namen oder direkt IIOP bei Internet-Adressen
- Location muss Struktur aufweisen
  - um in ein CosNaming-Objekt umgewandelt werden zu können
  - oder um eine Internetadresse zu kennzeichnen
- Muss nach Lieferung in eine Objektreferenz umgewandelt werden
  - über einen „NamingContext“ wird ein „resolve“ durchgeführt (Details siehe *CosNaming*-Spezifikation)
  - Ergebnis ist eine Objektreferenz; darüber können Methoden aufgerufen werden

Location als CORBA-Name

- Location als „mafuri“: bezeichnet einen CORBA-Namen
  - man kann den CORBA-Namensdienst direkt verwenden
  - lokationsunabhängig

```
mafuri   := "CosNaming:" location
location := components | "/" location
components := component | component "/" components
component := id "!" kind
id := xpalphas (RFC 1630: [A..Z], [a..z], "+", ...)
kind:= xpalphas
```

```
module CosNaming {
    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence<NameComponent> Name;
    ...
}
```

IDL von CosNaming

Beispiel: Der URI **CosNaming:/user!domain/user_name!u3**

wird als CosNaming::Name so umgewandelt: (Schlüssel-Wert-Paare)
{ { "user", "domain"}, { "user_name", "u3"} }

und kann so dem Resolver zugeleitet werden, um eine Bindung zu erreichen

Location als Internetadresse

- Location als „mafurl“: bezeichnet eine Internet-Adresse
  - Nutzen von IIOP
  - Wird in eine IOR umgewandelt (Interoperable Object Reference)

```
mafurl   := "mafiiop:" location
location := "//" [host ":" port "/"] agentsystem ["/" place ]* [?components]
host := hostname | hostnumber
agentsystem := uchar+    (URL char, muss ggf. als Escapesequenz dargestellt werden)
place := uchar+
components := component ["&" components]
component := tagname "=" tagvalue
tagname := "TAG_ORB_TYPE" | "TAG_CODE_SETS" | "TAG_SEC_NAME"
                | "TAG_ASSOCIATION_OPTIONS" | "TAG_GENERIC_SEC_MECH"
tagvalue := uchar+
```

ab IIOP 1.1

mafiiop://host.beispiel.net:9000/meinSystem/pfad/zur/Stelle1?TAG_ORB_TYPE=0x4f425400
(„ORBit“)

Location

- mafurl: Stellen können auch adressiert werden
  - nicht verpflichtend, da nicht unbedingt über CORBA erreichbar; üblicherweise wird das Agentensystem angesprochen
- Mit mafurl wird das entfernte Objekt direkt über IIOP angesprochen
- Für statische Objekte gut geeignet
- IIOP-Umlenkung für bewegliche Objekte notwendig
  - um zur tatsächlichen Objektreferenz (IOR) zu kommen
  - diese ist jedoch optional (dann müssen Namensdienste diese IORs ständig aktualisiert bereithalten)

Vordefinierte Konstanten in der MAF-IDL

- Agentensystemtyp (short)
  - NonAgentSystem = 0
  - Aglets = 1 (IBM)
  - MOA = 2 (*)
  - AgentTcl = 3
- Authentifizierung (short)
  - none = 1
  - one-hop = 2
- Sprachen (short)
  - LanguageNotSpecified = 0
  - Java = 1
  - Tcl = 2
  - Scheme = 3
  - Perl = 4
- Serialisierung (short)
  - SerializationNotSpecified = 0
  - Java Object Serialization = 1

OMG ist Namensvergabeautorität (Naming authority) für MAF. Dies hier sind die laut MAF vorgegebenen Anfangswerte für die Namensvergabe.

(*) Mobile Objects and Agents: The Open Group (insb. Beiträge von HP)

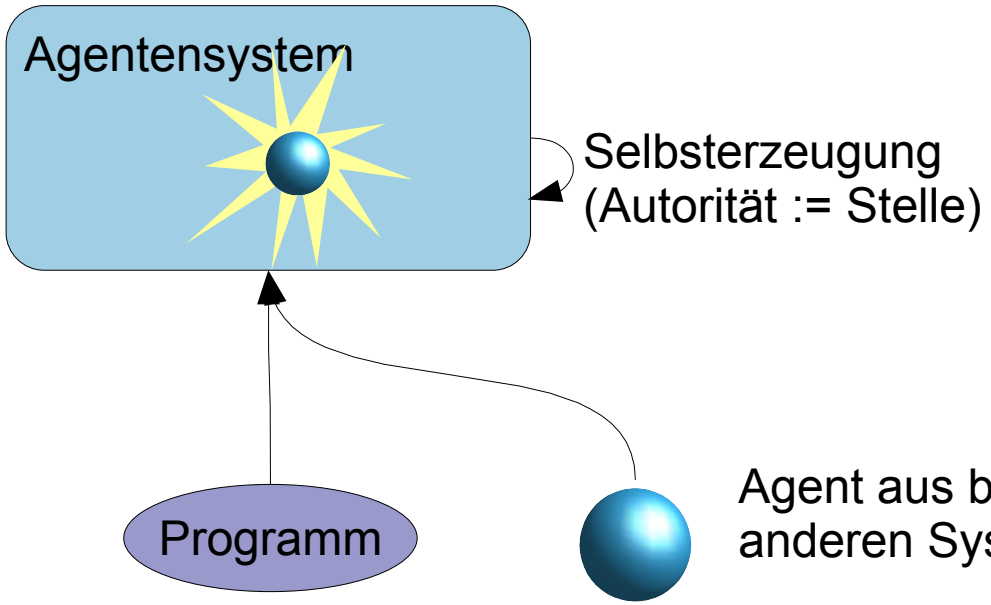
Agentenprofil

- Feststellen der Kompatibilität mittels *Profil*
  - beschreibt relevante Merkmale der Agentenverarbeitung
  - zur Prüfung genutzt
    - ob Agent an der Zielstelle ausführbar ist
    - ob eine bestimmte Funktionalität verfügbar ist
- Taucht als Parameter auf bei
  - Erzeugung des Agenten
  - Migration / Klassentransfer
  - Suche nach kompatiblen Agentensystemen
  - Suchen / Registrieren von Agenten

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

MAFAgentSystem: Agentenerzeugung

- Entfernte Agentenerzeugung
  - Anfrage eines Klienten oder
  - Initiative des Agentensystems
  - Authentifikation erforderlich




```
Name create_agent(
        in Name             agent_name,
        in AgentProfile     agent_profile,
        in OctetString      agent,
        in string           place_name,
        in Arguments        arguments,
        in ClassNameList    class_names,
        in string           code_base,
        in MAFAgentSystem   class_provider)
    raises (
        ClassUnknown,
        ArgumentInvalid,
        DeserializationFailed,
        MAFExtendedException);
```

MAFAgentSystem: Agentenerzeugung

- Erklärungen zur IDL
  - **name**: Rückgabe gleich dem Parameterwert, falls Name bestimmbar
  - **agent_profile**: Liefert das Profil für den neuen Agenten
  - **agent**: Agenten-spezifischer, opaker Parameter
    - kann alles Mögliche sein, ggf. Agentenklassen, Zustand
    - Agentensystem muss Wert verarbeiten können
  - **place_name**: Stelle, an welcher der Agent entstehen soll
  - **arguments**: Argumente für den Konstruktor
  - **class_names**: nennt notwendige Klassennamen
    - darf leer bleiben
    - abhängig von Codetransfermechanismus
  - **code_base**: Angabe, wo die Klassen zu finden sind (abh. von Lieferant)
  - **class_provider**: Klassenlieferant

MAFAgentSystem

- Klassen holen
  - class_name_list: Namen der Klassen
  - code_base: Verweist auf die Codebasis
    - damit weiß das Agentensystem, ob es der Lieferant ist
  - agent_profile: wegen Sprache und Serialisierung
- Agentensystem wählen
  - findet ein anderes Agentensystem (in der „Nähe“), das den Agenten ausführen kann
  - „hochgradig anwendungsspezifisch“
  - delegiert an den MAFFinder

```
OctetStrings fetch_class(
        in ClassNameList    class_name_list,
        in string           code_base,
        in AgentProfile     agent_profile)
    raises (
        ClassUnknown,
        MAFExtendedException);
```

```
Location find_nearby_system_of_profile(
        in AgentProfile     agent_profile)
    raises (
        EntryNotFound);
```

MAFAgentSystem

- Agentenstatus
  - running, suspended, terminated

```
AgentStatus get_agent_status(
                in Name          agent_name)
        raises (
                AgentNotFound);
```

- Infos zum Agentensystem
  - Agent kann damit bestimmen, ob das System für ihn geeignet ist
  - **language_maps**
    - Programmiersprachen
    - Serialisierung
  - **agent_system_description**
    - kein besonderes Format

```
AgentSystemInfo get_agent_system_info();

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

MAFAgentSystem

- Authentifizierung
  - ist der Agent authentifiziert
  - Methode der Authentifizierung
    - zurzeit nur one-hop oder nichts

```
AuthInfo get_auth_info(
            in Name          agent_name)
      raises (
            AgentNotFound);
```

- MAFFinder-Referenz
  - Auf diese Weise erhält der Agent* den Zugriff auf den MAFFinder

```
MAFFinder get_MAFFinder()
      raises (FinderNotFound);
```

*was nicht impliziert, dass der Agent ein CORBA-Objekt ist, dass er aber Zugriff auf CORBA-Objekte prinzipiell erlangen kann

MAFAgentSystem

- Agenten auflisten
- Agenten einer Autorität auflisten
- Alle Stellen eines Agentensystems listen
  - jene Stellen, die beim MAFFinder registriert sind

```
NameList list_all_agents();
```

```
NameList list_all_agents_of_authority(
        in Authority        authority);
```

```
Locations list_all_places();
```

MAFAgentSystem: Migration

- Agent entgegennehmen
  - vom empfangenden Agentensystem verwendet
- Parameter
  - agent
    - opak, kann Zustand und/oder Klassen beinhalten; wenn Klassen beinhaltet sind, müssen diese nicht in der class_names-Liste genannt sein
  - place_name
    - Zielstelle in diesem Agentensystem
  - class_names
    - Klassennamen, die ggf. besorgt werden müssen

```
void receive_agent(
            in Name                 agent_name,
            in AgentProfile         agent_profile,
            in OctetString          agent,
            in string               place_name,
            in ClassNameList        class_names,
            in string               code_base,
            in MAFAgentSystem       agent_sender)
      raises (
            ClassUnknown,
            DeserializationFailed,
            MAFExtendedException);
```

  - code_base
    - Ort der Klassendefinitionen

MAFAgentSystem

- Mögliche Implementierung dieser Methode
  - Nachprüfen, ob im Parameter **agent** alle Klassen mitgeschickt wurden oder lokal im Zwischenspeicher liegen
  - Aufruf von **fetch_class**, um fehlende Klassen zu laden
  - Deserialisieren des Agenten an der angegebenen Stelle (oder an der Standardstelle, wenn nicht angegeben)
  - dann dort instanziieren
- Alternative zur Ausführung
  - Agent wird in Warteschlange geschoben
  - Agent wird an ein anderes Agentensystem in der Region verwiesen

MAFAgentSystem

- Agent einfrieren
  - Aufwecken durch resume
- Agent wieder starten
- Agent endgültig terminieren
- Agentensystem anhalten
  - AS kann Agenten benachrichtigen und ggf. Zustand sichern

```
void suspend_agent(
        in Name          agent_name)
    raises (
        AgentNotFound,
        SuspendFailed,
        AgentIsSuspended);
```

```
void resume_agent(
        in Name          agent_name)
    raises (
        AgentNotFound,
        ResumeFailed,
        AgentIsRunning);
```

```
void terminate_agent(
        in Name          agent_name)
    raises (
        AgentNotFound,
        TerminateFailed);
```

```
void terminate_agent_system()
    raises (TerminateFailed);
```

Management: Auffinden von Agenten

- Auffinden eines Agenten
  - erforderlich, um Kommunikation durchzuführen
  - zunächst wichtig, Agentensystem zu finden
  - Auffinden des Agenten selbst wichtig für Management
- Namensdienst
  - kann von Agentensystemen angeboten werden
  - basierend auf Agentennamen

Nützliche CORBA-Dienste

- CORBA-Dienste (Module)
  - Namensdienst (CosNaming)
  - Lebenszyklusdienst (CosLifeCycle)
  - Exportdienst (CosExternalization)
- Sicherheitsdienst (Common Secure Interoperability)
  - Teil des CORBA-Kerns

CosNaming

- Nutzen für Agenten
  - können beim Eintritt in eine Region den Namensdienst von CORBA nutzen, um den *MAFFinder* zu bekommen
- Agenten können sich selbst im Namensdienst publizieren
  - bieten sich dann als gewöhnliche CORBA-Objekte an
  - Anwendungen können Referenzen auf Agenten erhalten
  - damit können die Agentenmethoden direkt aufgerufen werden

CosLifeCycle / CoSExternalization

- Verwendung im Agentensystem
  - MAFAgentSystem und MAFFinder können über CosLifeCycle gestartet und gestoppt werden
  - ggf. sogar verlagert werden
  - create_agent und terminate_agent können intern die Dienste von CosLifeCycle nutzen (wenn die Agenten CORBA-Objekte sind)
  - Externalization nur benötigt, wenn kein Java zum Einsatz kommt

MAFFinder

- Agenten finden
  - **agent_profile**: kann verwendet werden, um Suchkriterien festzulegen
    - Damit kann eine Gruppe von Agenten bezeichnet werden
  - Keine Verbindlichkeit des Ergebnisses!
    - „Only an agent can control when and where it travels.“
- Agentensysteme finden
  - Agentensysteme müssen im MAFFinder registriert sein
  - Suchkriterien in Info-Parameter

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

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

MAFFinder

- Stelle finden
  - für den Fall, dass nur der Stellenname (und nicht das Agentensystem) bekannt sind
- Agent registrieren
  - wird von mobilen Agenten häufig aufgerufen
  - Eintrag wird bei gleichem Namen überschrieben
- Agentensystem registrieren
  - Wiederholter Aufruf nicht erlaubt
    - erst deregistrieren

```
Location lookup_place(
                in string      place_name)
        raises (
                EntryNotFound);
```

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

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

MAFFinder

- Stelle registrieren
  - Wiederholter Aufruf nicht erlaubt
    - erst deregistrieren
- Agent deregistrieren
- Agentensystem deregistrieren
- Stelle deregistrieren

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

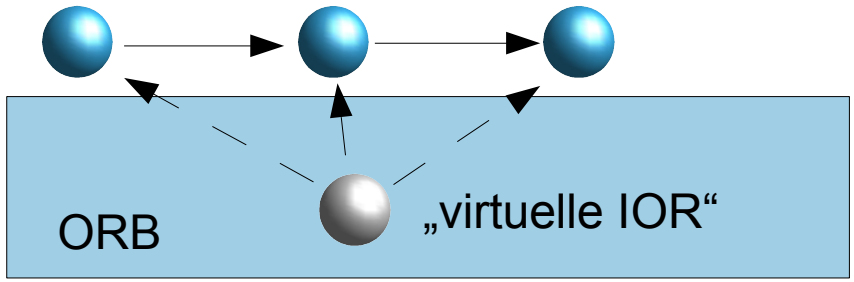
```
void unregister_agent(
        in Name         agent_name)
    raises (
        EntryNotFound);
```

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

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

Referenzierung von Agenten

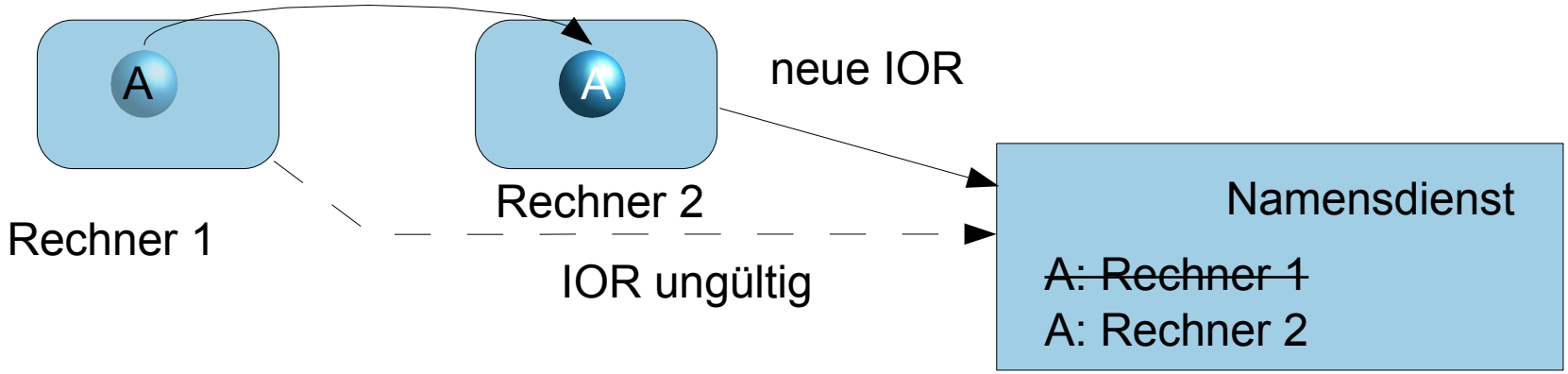
- Besondere Herausforderung
  - Agenten sind mobil
  - IORs beinhalten den Rechnernamen
  - also wie eine verlässliche Referenz finden?*
- Erste Möglichkeit
  - ORB muss Abbildung zwischen IOR und „echter“ IOR verwalten und diese aktualisieren
    - beschrieben unter GIOP in CORBA-Spezifikation, aber nicht verpflichtend!



*sofern die Agenten sich als CORBA-Objekte anbieten.
Andere Mechanismen sind in der MAF nicht spezifiziert.

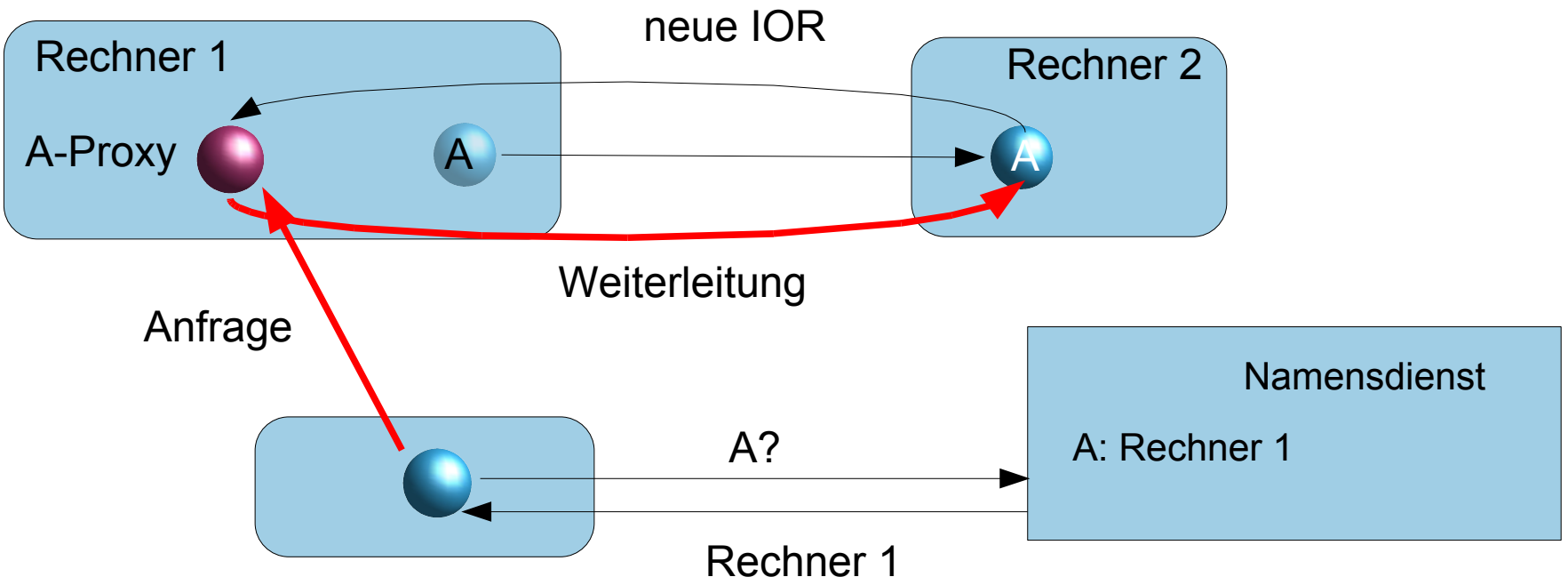
Agentenreferenzen

- Zweite Möglichkeit
  - IOR nach jeder Migration im Namensdienst aktualisieren (durch die Agentensysteme)
  - Bei Kontaktierung kann eine Ausnahmebedingung (Exception) erkennen lassen, dass die IOR ungültig ist und dass man eine neue braucht



Agentenreferenzen

- Dritte Möglichkeit: Proxy-Agent
  - führt eine lokale Übersetzung der „offiziellen IOR“ zur tatsächlichen IOR durch
  - leitet Nachricht weiter



Proxy-Agent

- **Aber**: Drastische Nachteile!
  - Proxy-Agent und echter Agent müssen bei jeder Migration Kontakt aufnehmen (um neue IOR zu übermitteln)
  - Heimatsystem (auf Rechner 1) muss ständig erreichbar sein
    - nichts für Laptops!
  - Ständige Weiterleitung von Anfragen bewirkt hohe Netzlast

Der Einsatz von mobilen Agenten wird fragwürdig, wenn für den Betrieb hohe Netzlasten zu erwarten sind.

Sicherheit

- MAF führt keine eigenen Mechanismen ein
  - Weitgehende Nutzung der CORBA-Mechanismen (Common Secure Interoperability)
- Notwendige Informationen und Mechanismen
  - Agentennamen (Prinzipal, Identität)
  - Klientenauthentifizierung für entfernte Agentenerzeugung
  - Gegenseitige Authentifizierung für Agentensysteme
  - Zugriff auf Authentifizierungsergebnisse und Berechtigungen für das Agentensystem
  - Agentenauthentifizierung und Delegation
  - Sicherheitspolitiken für Agenten und Agentensysteme
  - Integrität, Vertraulichkeit, Replay-Erkennung, Authentifikation

Sicherheitsdienst

- Agentensystem muss liefern können
  - Prinzipal, Agentenidentität, Authentifizierungsstatus und -art
  - Bei sicheren ORBs werden Daten über den Prinzipal übermittelt, auf die das Agentensystem Zugriff hat
  - Fehlen Sicherheitsdienste, ist eine Authentifikation meist unmöglich
  - Authentifizierter Agent wird als „nicht authentifiziert“ behandelt, wenn ein AS keinen sicheren ORB verwenden kann
- Entfernte Erzeugung
  - PrincipalAuthenticator-Objekt in CORBA zur Authentifizierung des entfernten Klienten
  - Fehlt der Sicherheitsdienst, ist der erzeugte Agent „nicht authentifiziert“

Sicherheitsdienst

- Agentensystem-Authentifizierung
  - Optionen in IORs: EstablishTrustInTarget, EstablishTrustInClient
  - erzwingen Credential-Austausch vor Transfer
- Zugriff auf Authentifizierungsresultat
  - SecureCurrent-Schnittstelle bei sicheren ORBs
  - Bei geringeren Sicherheitsstandards sind entfernte Aufrufe und Agententransfervorgänge anonym

Sicherheitsdienst

- Agentenauthentifikation und Delegation
  - Sichere ORBs propagieren die Berechtigungen (Credentials) mit dem migrierenden Agenten
  - Bei CSI 0 oder 1 können nur entweder die Agentencredentials oder die Systemcredentials übertragen werden, bei Stufe 2 beide
- Sicherheitspolitiken
  - Objekte können generell die Ausführung verweigern
  - Bei sicheren ORBs kann dies aufgrund der Aufrufer-Credentials geschehen, ansonsten nur aufgrund der Aufruf-Parameter

Zusammenfassung

- MAF-Spezifikation
  - bringt CORBA und Agenten zusammen
  - spart Details aus, um größtmögliche Akzeptanz zu erreichen
- MAF-kompatible Plattformen
  - Grasshopper (IKV++, Spinoff von Fraunhofer FOKUS, Berlin)
  - Aglets
  - MOA (? lediglich „gegenseitiger Einfluss“ MOA/MAF betont)
- Der Großteil der Agentenplattformen hat MAF nicht implementiert
  - dennoch wird MAF als eine Standardisierung angesehen