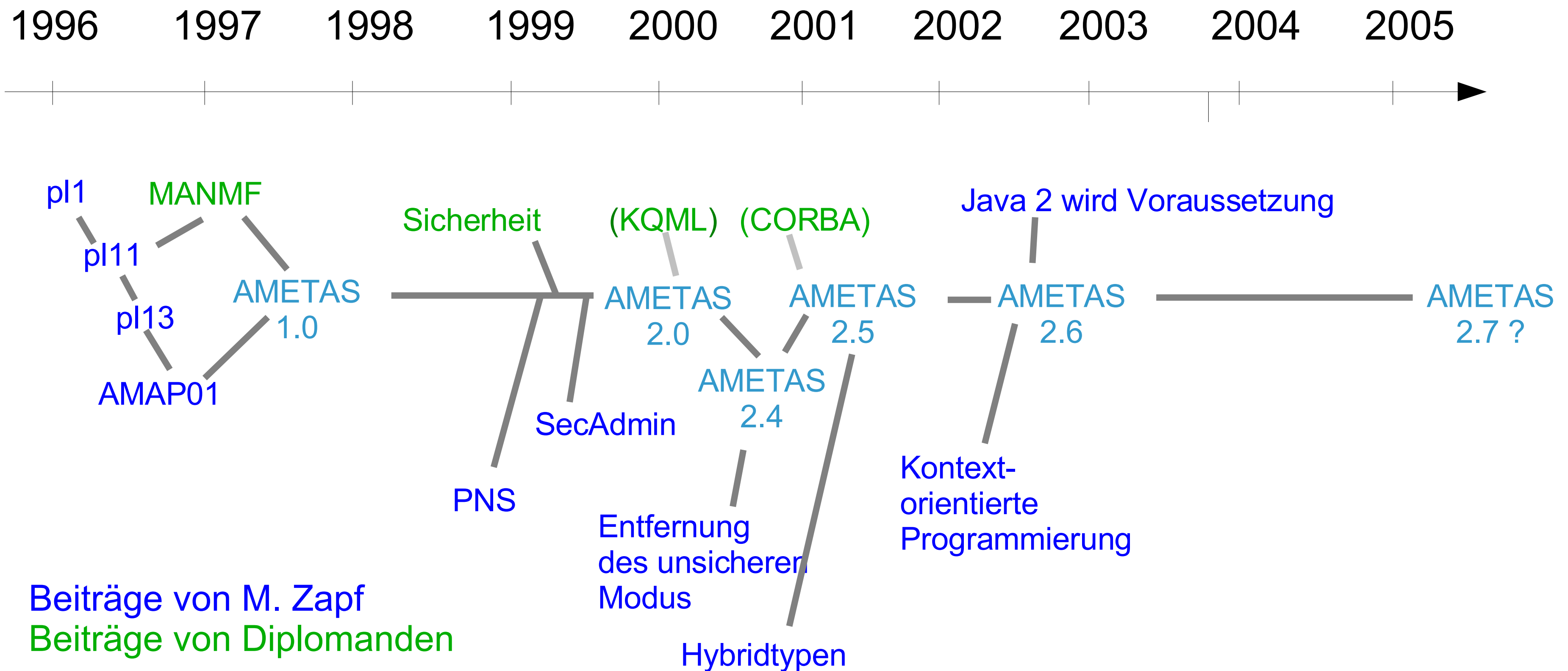
AMETAS

Asynchronous Message Transfer Agent System

02.05.2007

Geschichte



Geschichte

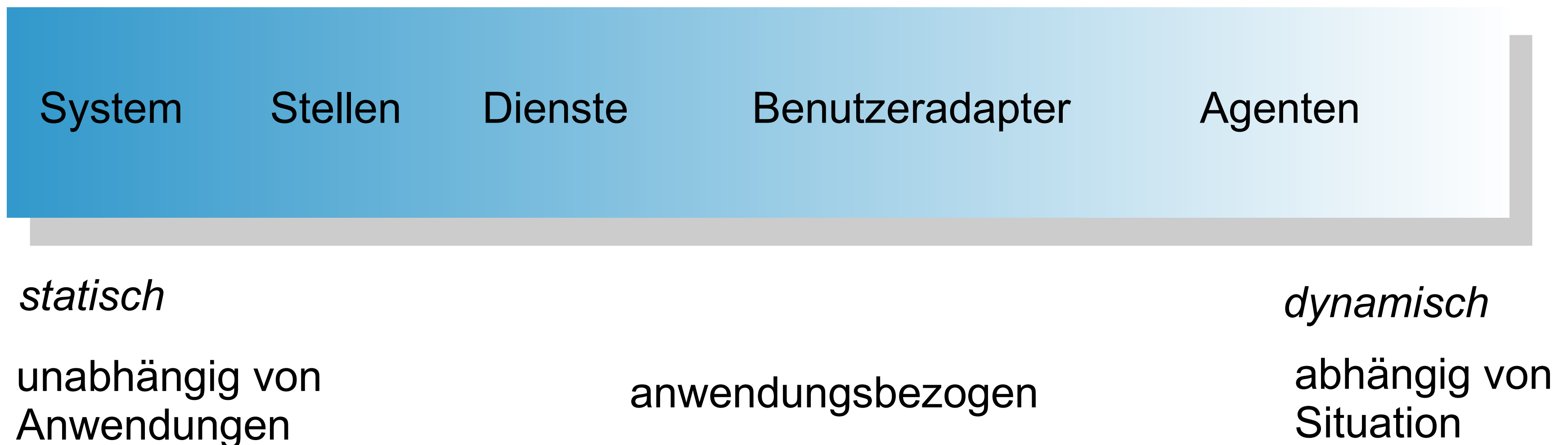
- Anfänge
  - **pl1**: „Place“ (Ver. 1), Nachfolger pl11
  - pl13 mit Serialisierung
  - MANMF: Mobile Agent Network Management Framework (K. Herrmann); Entwicklung des Asynchronnachrichtensystems
  - AMAP01: „Actively Mobile Agent Place“, betont *aktive Mobilität*
  - Vereinigung der Bemühungen in AMETAS „Asynchronous Message Transfer Agent System“
- Fortan gemeinsame Entwicklung

Geschichte

- Wichtigste Entwicklungen
  - Sicherheitssystem (H. Müller) auf Basis von Java 1.1
  - Place Name System
  - AMETAS 2.4 entfernt alte Teile; Betrieb nur noch unter dem „sicheren Modus“
  - Entwicklung der KQML-Unterstützung (jedoch nie eingebaut)
  - Entwicklung einer CORBA-Schnittstelle (jedoch nie eingebaut)
  - Hybridtypsystem als Ergebnis der Dissertation von M. Zapf
  - Kontextorientierte Programmierung
  - Aufgabe der Java-1-Kompatibilität

Eigenschaften

- Grundlegende Ideen frei nach Telescript
  - „Places“ = „Stellen“, mobile Agenten, *go*-Kommando
- Einteilung der Architektur in statische und dynamische Teile



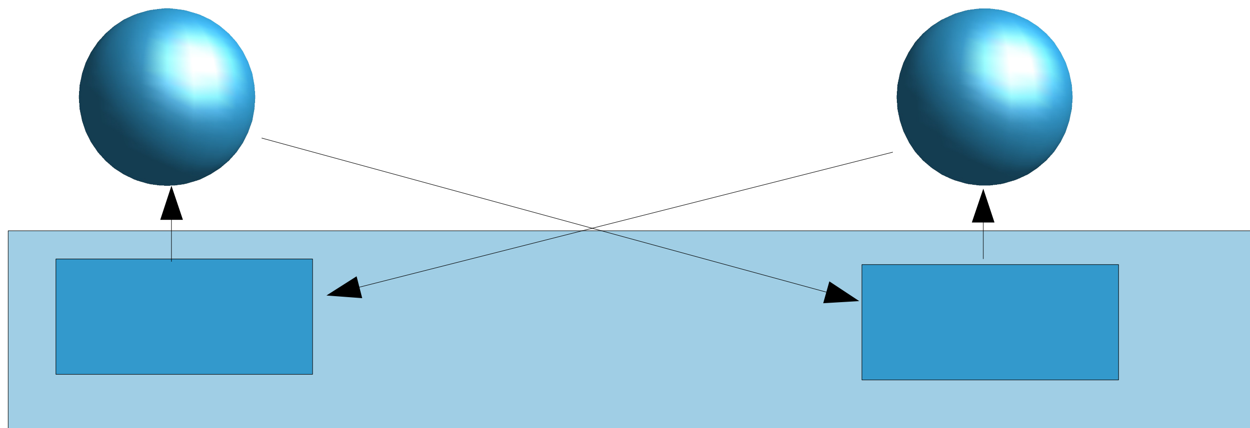
Eigenschaften

- Ursprüngliches Problem
  - Mobile Agenten: Irgendwo, irgendwann ... d.h. räumlich/zeitliche Entkopplung
  - Wie adressiere ich einen Agenten, der vielleicht noch gar nicht anwesend ist und erst noch eintrifft?
  - Wie kann ich verhindern, dass ein Agent wegmigriert, während ich mit ihm kommuniziere?

Einfache Punkt-zu-Punkt-Kommunikation ist nicht passend.

Postfachsystem

- Mailbox / Postfach
  - Jeder Agent hat ein Postfach
  - Das Postfach existiert auch ohne den Agenten und puffert eintreffende Nachrichten



Asynchroner Nachrichtentransfer

Eigenschaften

- Rolle der Agenten
  - Autonomie als „kleinster gemeinsamer Nenner“ aller Agententheorien
  - schließt die Rolle als Server aus

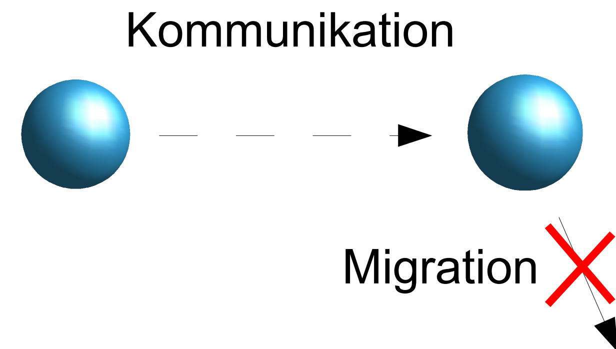
Später genauer spezifiziert: Serverrolle kann auf höherer Abstraktionsebene eingenommen werden, jedoch muss der Plan des Agenten jederzeit den Vorrang vor äußeren Einflüssen haben.

- Kooperation
  - anderer Agent oder Komponente
  - muss prinzipiell mit Nichtkooperation des Agenten rechnen

Autonomie

- Kooperation
  - Verhalten jedes Agenten wird durch Verhaltensbeschreibung (=Code) des Agenten bestimmt
  - Kooperation findet statt, wenn die Pläne der Agenten zueinander *kompatibel* sind.

- Keine Serverrolle
  - Plan des „Serveragenten“ müsste Interaktionen mit anderen Agenten einschließen
  - dieser Plan müsste im Wesentlichen ereignisorientierte Struktur haben



Ressourcen

- Ursprüngliches Problem
  - Was passiert mit Fenstern, Dateien, Sockets, die der Agent verwendet, wenn er migriert?
- Prinzipiell drei Möglichkeiten der Behandlung
  - **transparent**, indem eine Strategie angewendet wird, sodass die Ressourcen auch entfernt erreicht werden **(größter Aufwand)**
  - **unbehandelt**, indem per „Vogel-Strauß-Haltung“ die Verantwortung auf den Programmierer verlagert wird **(kleinster Aufwand)**
  - **restriktiv**, indem dem Agenten alles verboten wird, was seine Ausführung beeinträchtigen könnte; Mechanismen außerhalb von Agenten erlaubt

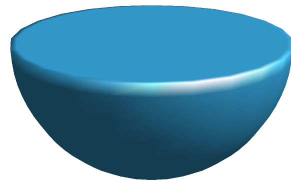
SecurityManager in Java

AMETAS verhält sich restriktiv.

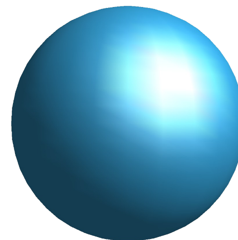
Stellennutzer

- Trennung der Zuständigkeiten in AMETAS („separation of concern“)
  - Sicht auf das System: **Benutzeradapter**
  - Logik des Systems: **Agenten**
  - Datenquellen des Systems: **Dienste**

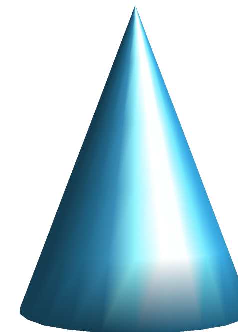
3-Tier-Architektur



Benutzeradapter



Agent



Dienst

Place User, „Stellennutzer“

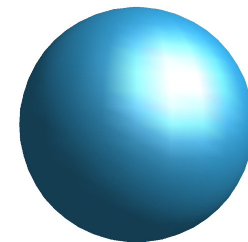
Stellennutzer

- Gemeinsamkeiten der drei Klassen
  - Gleiche Adressierung
  - Gleiche Kommunikation zwischen allen Komponenten (Postfachsystem)
  - Gleiche Verwaltung aller Stellennutzer im System
  - Alle Stellennutzer sind gegenüber den anderen völlig eigenständig und isoliert

Gleiche Elternklasse mit grundlegenden Funktionen

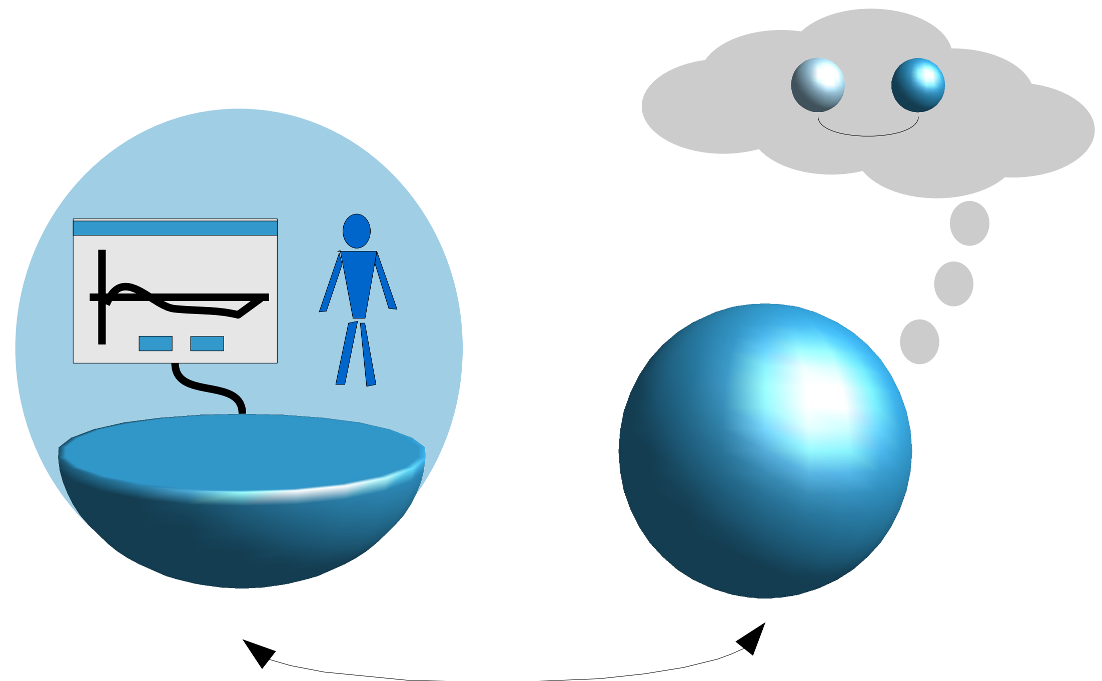
Agenten

- Agent als spezieller Stellennutzer
  - kann migrieren
  - darf keine externen Ressourcen nutzen (Fenster, Dateien, Sockets, ...)
  - kann nur mittels Nachrichten Informationen austauschen
  - kann Ereignisse empfangen (über das *Ereignissystem*)
  - kann mehrere Threads nutzen
- Träger der Logik der Anwendung
  - Kooperation zwischen Agenten
    in einer Multiagentenanwendung
- Ressourcenzugriff durch andere Stellennutzer



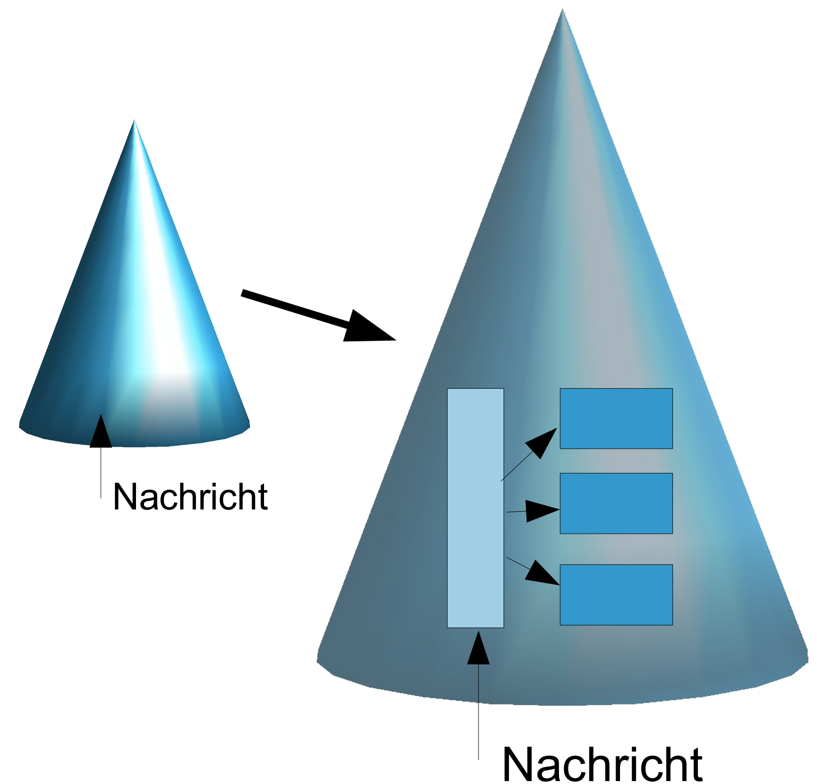
Benutzeradapter

- Schnittstelle zum Benutzer
  - darf beliebige Ressourcen verwenden, insbesondere Fenster oder Sockets
  - Anzeige der Resultate aus der Agentenanwendung
- Integration des Benutzers
  - Benutzeradapter macht den Anwender zu einem Teil des Agentensystems
  - Agenten oder Dienste kommunizieren mit BA in gleicher Weise wie untereinander



Dienste

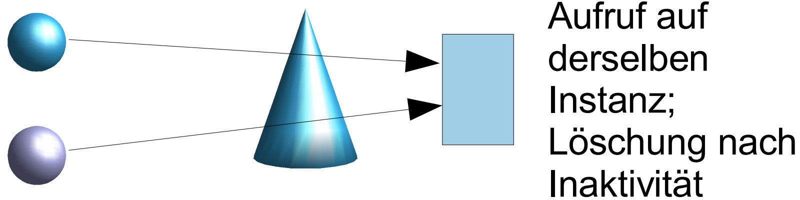
- Ankopplung des Systems
  - Einfacher Zugriff auf Sockets, Dateien
  - Zusammensetzung aus Dienstverwalter und Dienstobjekt
- Dienstverwalter
  - ist der Stellennutzer
  - repräsentiert den Dienst als einzelne,adressierbare Instanz im System
  - ist für alle Dienste gleich
- Dienstobjekt
  - eigentlicher, spezifischer Teil des Dienstes



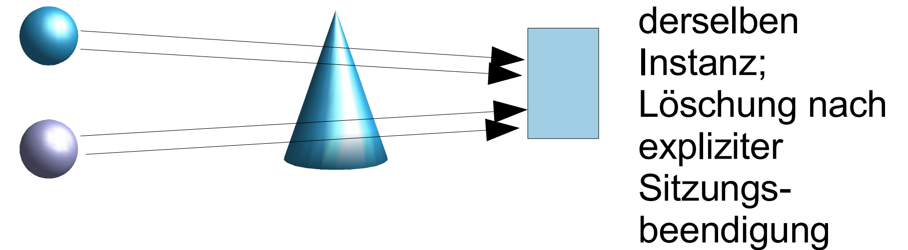
Dienste

- Aktivierungsmodi

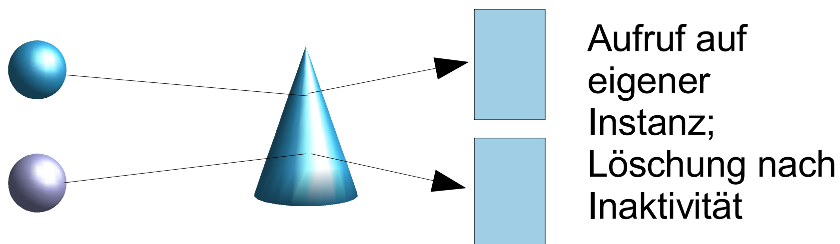
SHARED



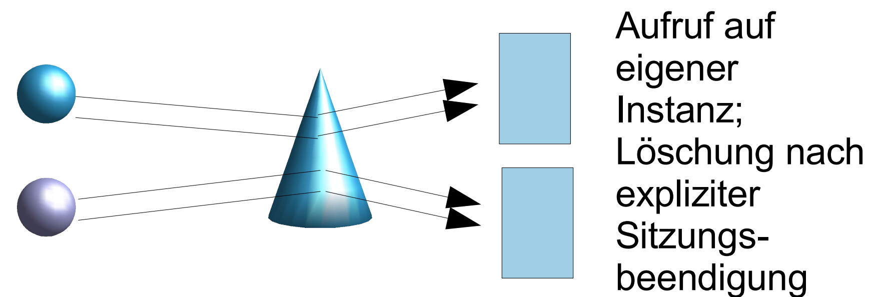
SHARED_SESSION



NON_SHARED



NON_SHARED_SESSION

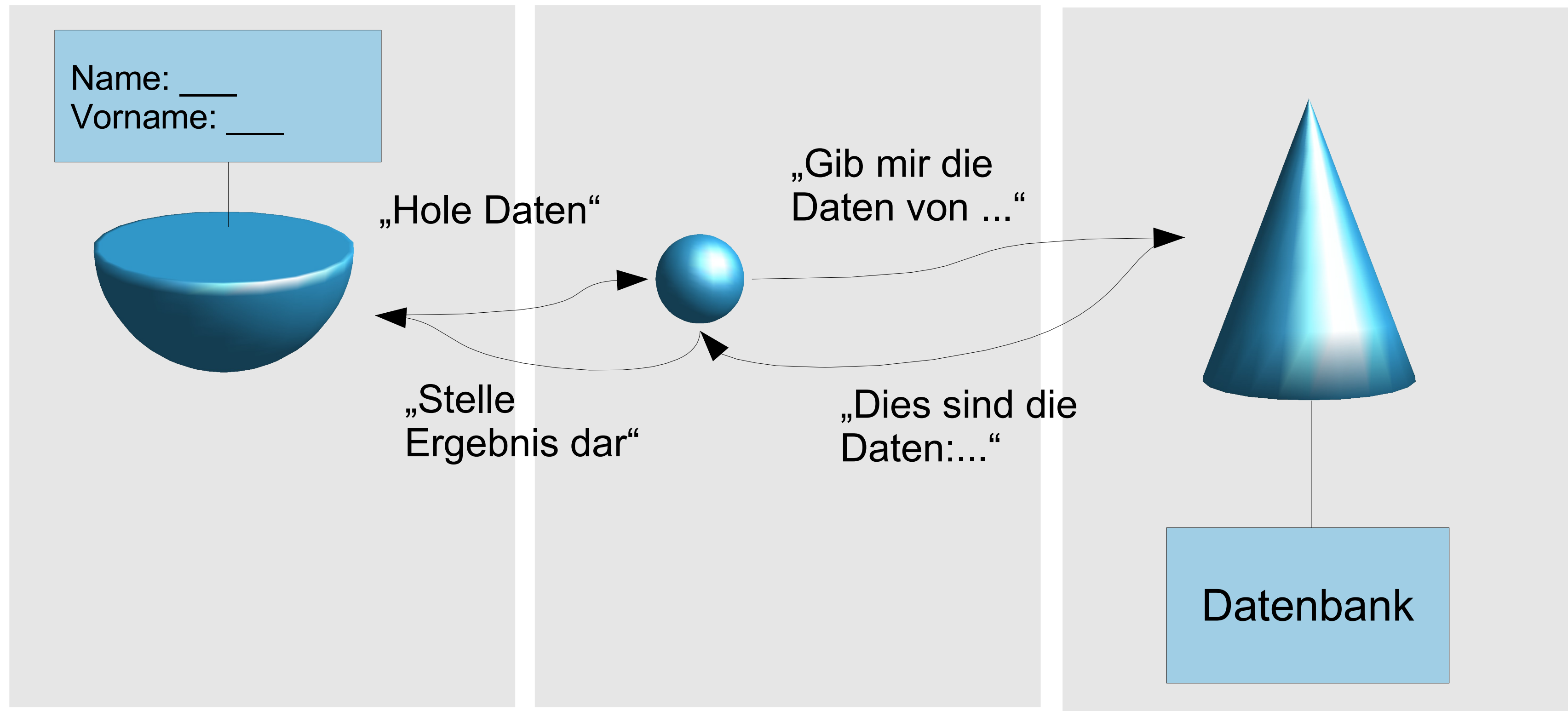


Dienste

- Flexibilisierung der Dienste
  - anfangs zu reiner Serverrolle „verdammt“
  - später (ab AMETAS 2.5) möglich, Dienste ebenfalls mit eigener Flusssteuerung auszustatten
  - Dienste können nun „initiativ“ werden, d.h. eine Kommunikation mit Agenten beginnen
- Ankopplung beliebiger externer Dienste
  - auch möglich, Webserver anzukoppeln
  - ersetzt damit Benutzeradapter!
- Aktivierungsmodi
  - von CORBA abgeschaut, aber allmählich an Bedeutung verlierend (siehe kontextorientierte Programmierung), nur noch „SHARED“

Zusammenarbeit zwischen Stellennutzern

- Kann ein Agent überhaupt sinnvoll zum Einsatz kommen, wenn er keine Ressourcen direkt verwenden darf?



Stellennutzerinteraktion

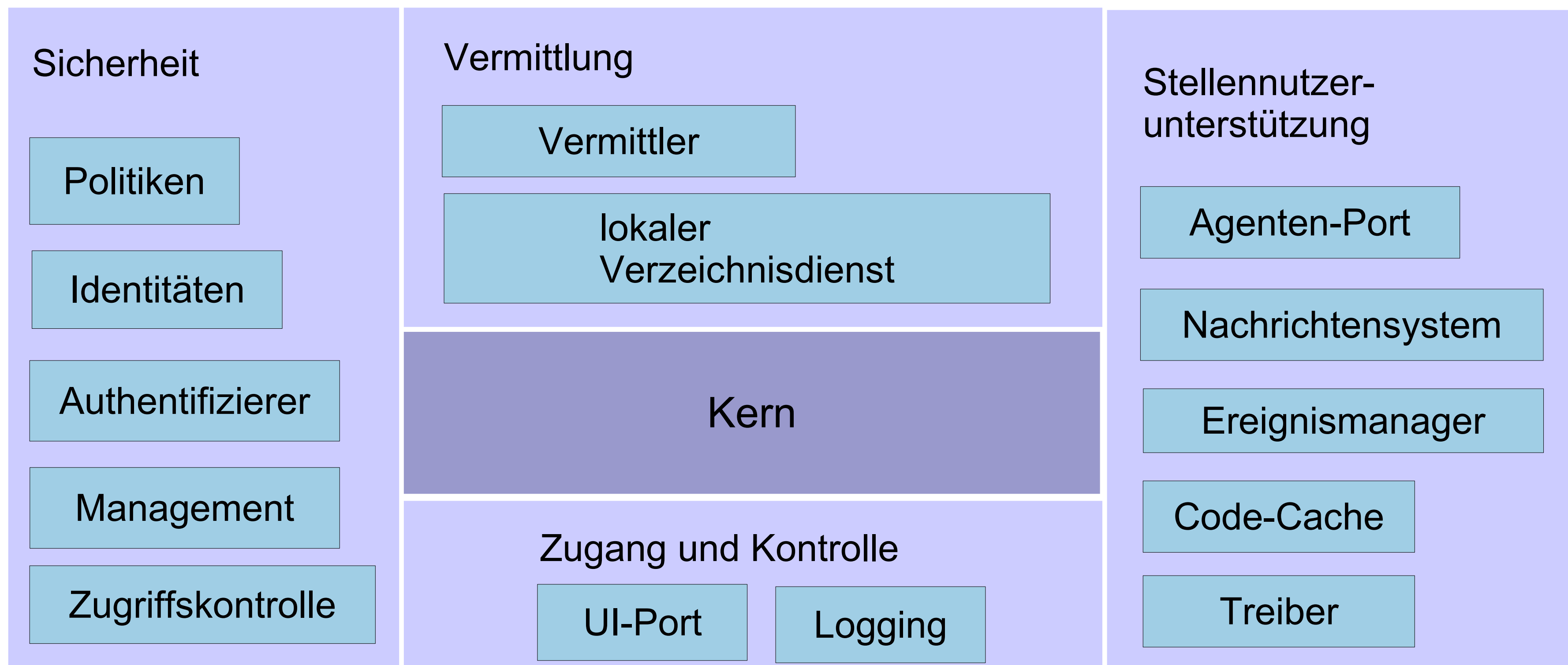
- „Kanonisches“ Vorgehen
  - Aktion anstoßen durch Benutzer
  - Benutzeradapter übersetzt in Agentennachricht
  - Agent bekommt Nachricht
  - Agent führt Plan durch
  - Agent kann Informationen von Dienst beziehen
  - Dienst kann sich externer Informationsquellen bedienen
  - Agent verarbeitet alle gewonnenen Informationen
  - Agent schickt Antwort an Benutzeradapter
  - Benutzeradapter sorgt für geeignete Aufbereitung der Ergebnisse

Die Stelle

- bietet den Stellennutzern eine Ausführungsumgebung
- trennt Stellennutzer voneinander (über die Klassenlader)
- bietet Basisfunktionen
  - Mobilität
  - Kommunikation
  - Vermittlung (Mediation)
  - Ereignissystem
  - Sicherheit
- erlaubt die Verbindung von externen Schnittstellen
  - ursprünglich zum Start von Anwendungen und zu deren Überwachung gedacht

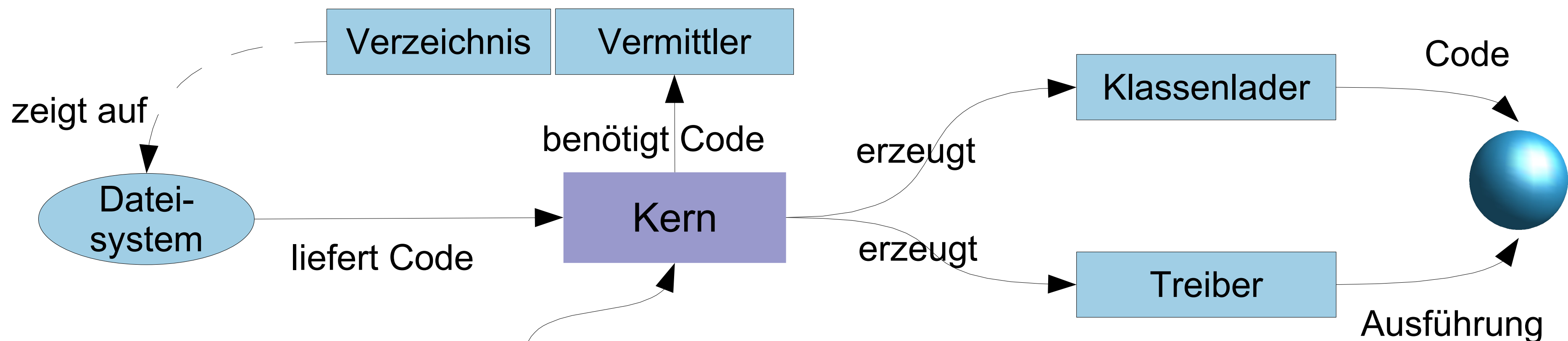
Die Stelle

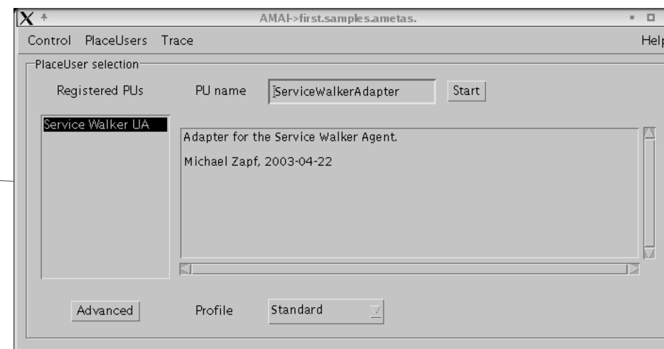
- Struktur



Szenario: Lokaler Start

- Stellennutzer (PU) wird lokal gestartet



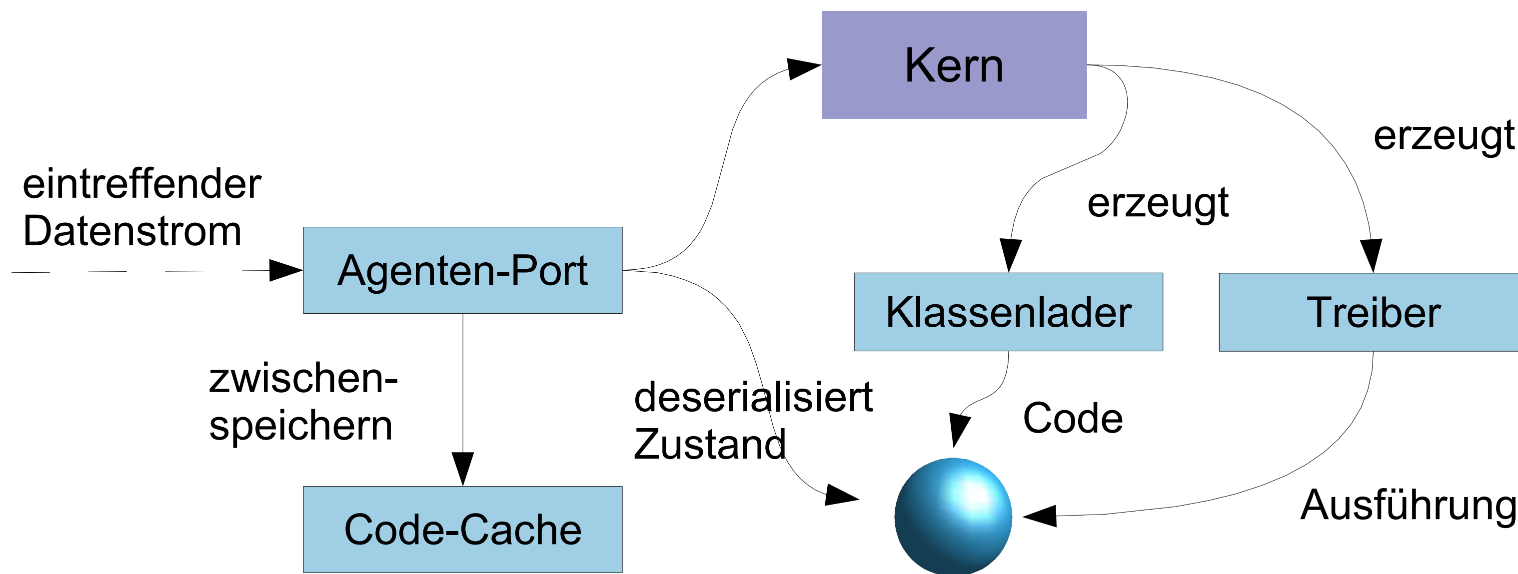


AMAI:

AMETAS
Attachable
Interface

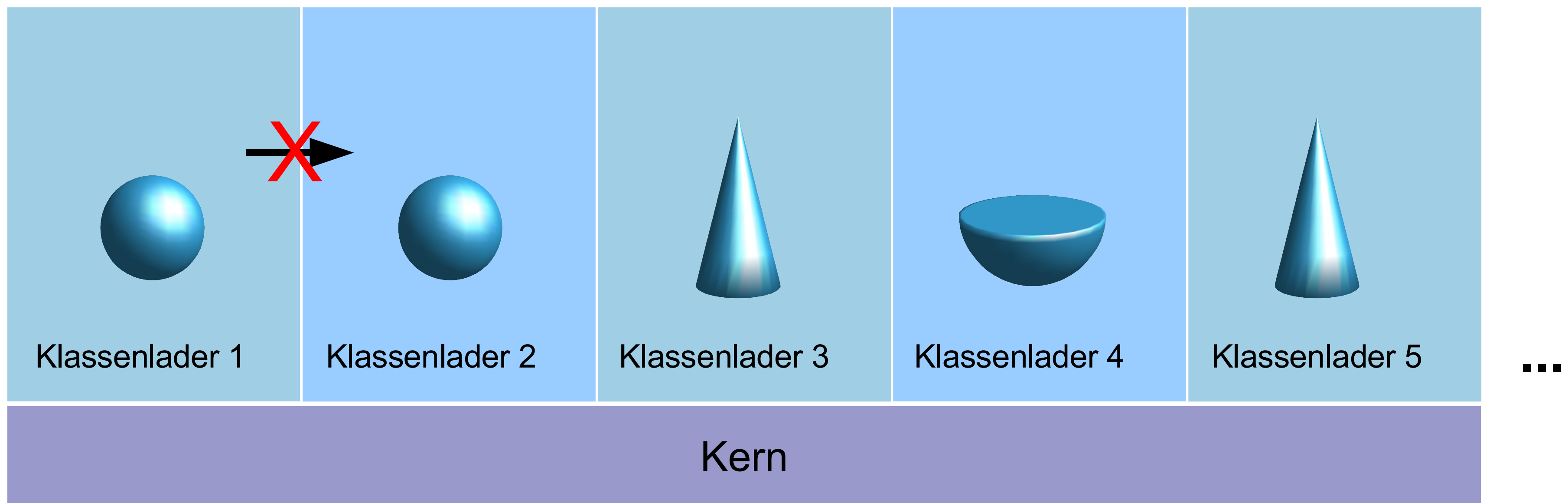
Szenario: Immigierter Agent

- Agent ist bei der Stelle angekommen



Trennung der Stellennutzer

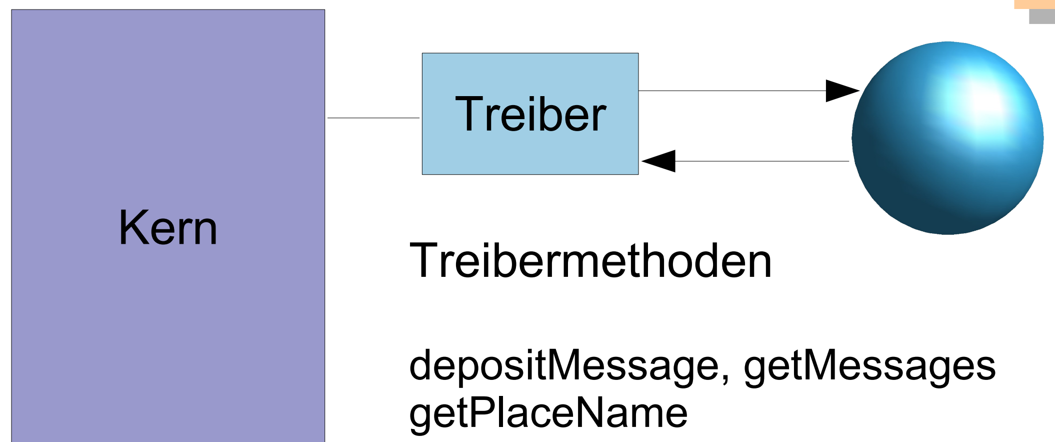
- Klassenlader erzeugen implizite Namensräume
  - Referenzierung fremder Klassen unmöglich
  - gleiche Klassennamen in verschiedenen Klassenladern möglich!



Treiber

- Schnittstelle zum System

Agent besitzt nur die Referenz auf den Treiber (also kein direkter Zugriff auf den Kern)



Referenz auf den Agenten nur dem Treiber bekannt

Treibermethoden

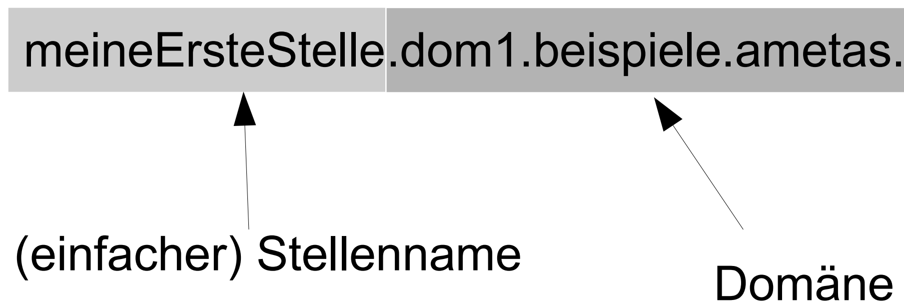
depositMessage, getMessages
getPlaceName
requestPUStartup, spawnAgent
output, trace
idle, wakeup
stopDriver, killPlaceUser
getPrivileges
getPUTypes, getTypesOf, request
registerEventListener
...

Agententreiber

Treibermethoden +
go
getLastPlace

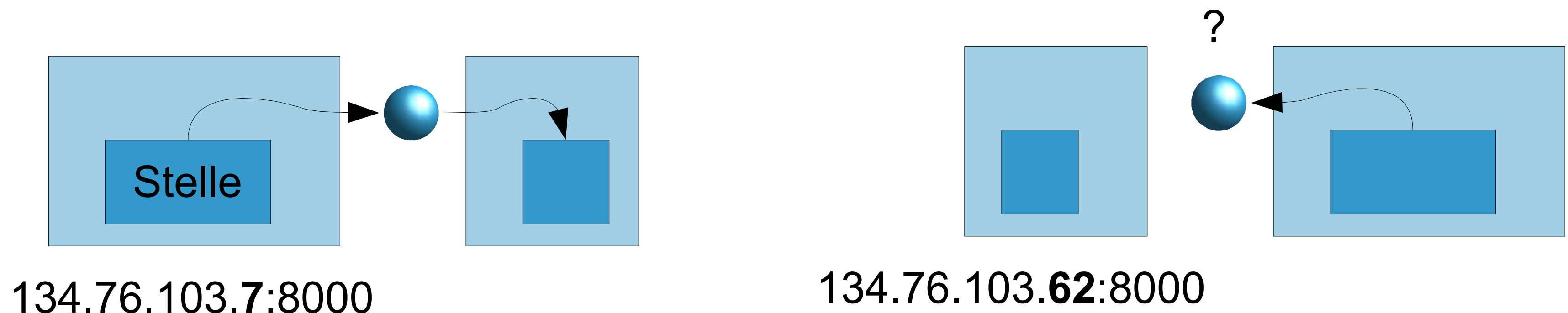
Stellennamen

- Adressierung wichtig für die Migration
- Früher nur „einfacher Stellenname“
- PNS: Place Name System, hierarchischer Namensraum
  - leichteres Management (vor allem bei sich ändernden Zuordnungen)



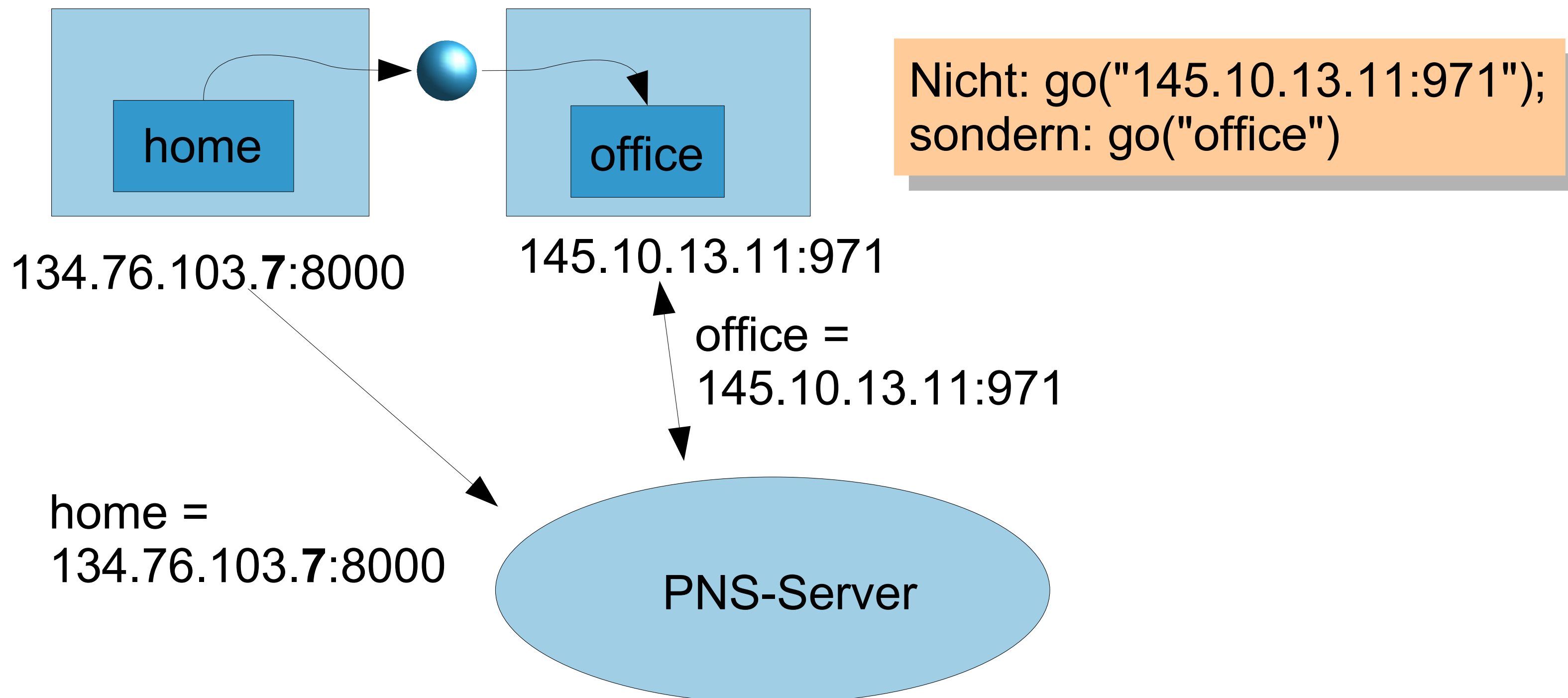
Stellenarten

- Warum nicht DNS (Domain Name System)?
  - DNS adressiert Rechner, nicht Prozesse
  - DNS nicht flexibel genug
- Stellenarten
  - **permanente** Stellen: immer auf demselben Rechner ausgeführt
  - **temporäre** Stellen: können unter wechselnden Rechnern ausgeführt werden (Einwählverbindungen [Dialup]!)



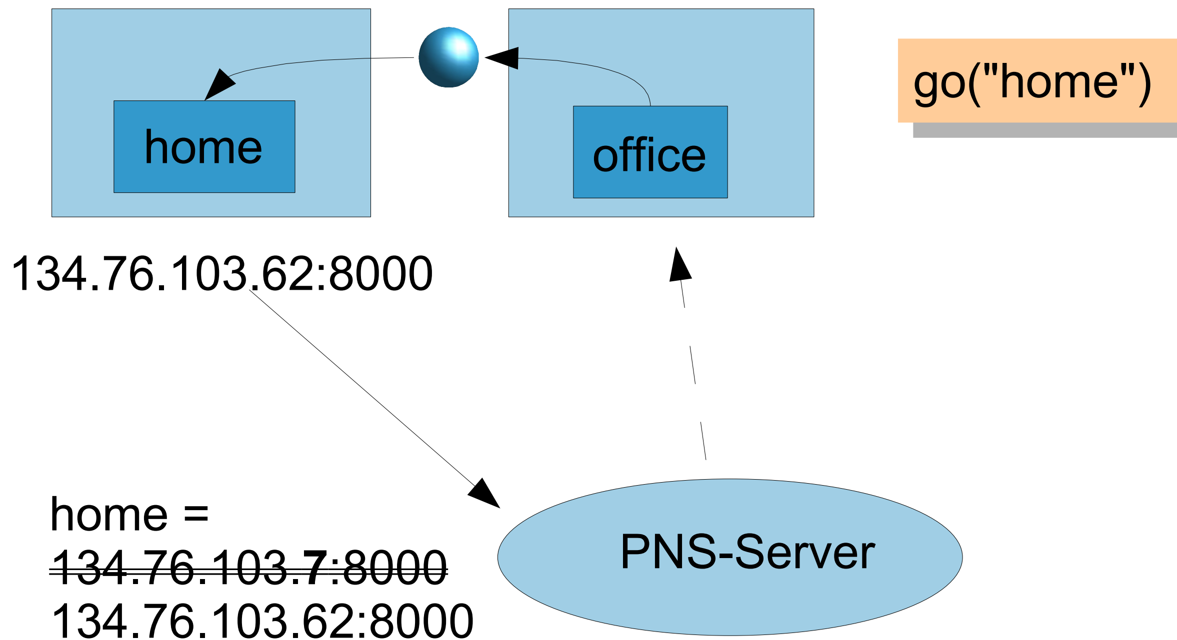
Place Name System

- Lösung durch Entkopplung von Rechnerabhängigkeit
  - Stellen melden ihre Position an den „PNS“
  - permanente Stellen vergleichen nur mit dem Eintrag im PNS



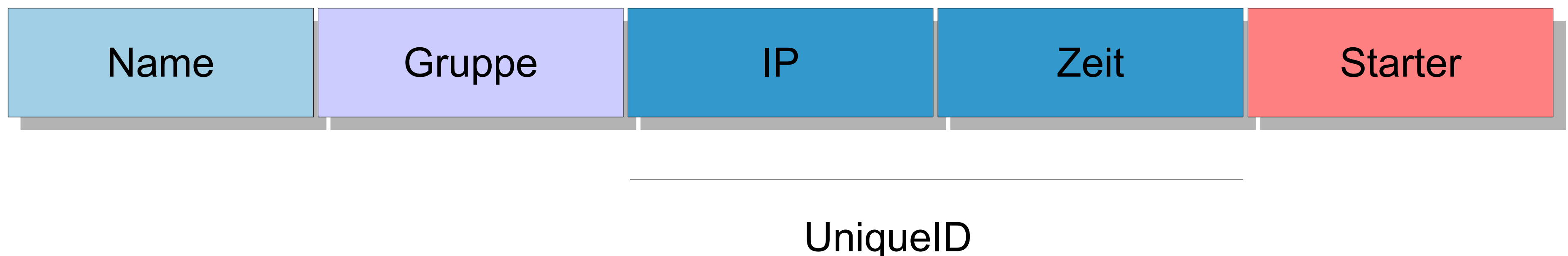
Place Name System

- Änderungen bei temporären Stellen
  - nur möglich durch Anwendung eines Authentifikationsschlüssels



Benamung von Stellennutzern

- Adressierung von Stellennutzern durch eindeutige IDs („Place User ID“)
  - beinhaltet Zeitstempel plus IP-Adresse des Rechners der Stelle, auf der dieser Stellennutzer gestartet wurde („Unique ID“)
  - beinhaltet „Name“ und „Gruppe“ für eine Gruppenadressierung
  - beinhaltet ID der Identität des Starters



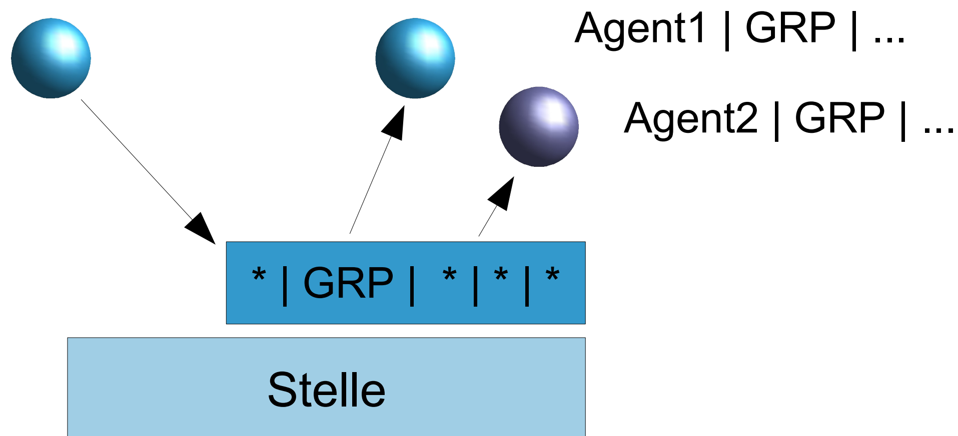
Benamung von Stellennutzern

- Name
  - beliebige Bezeichnung, nach Konvention der Klassenname
  - kann zur Adressierung einzelner Stellennutzer verwendet werden, wenn es nur eine Instanz dieses Stellennutzers gibt
- Gruppe
  - wie Name, gedacht für verschiedene Stellennutzer
- Sichere Adressierung nur über Unique ID!
  - wird bei der Erzeugung dem Erzeuger geliefert
  - kann dieser anderen Stellennutzern kundtun

Multicast

- Multicast zur Adressierung
  - einer Gruppe von Stellennutzern
  - unbekannter Stellennutzer

Annahme: Unbekannter Stellennutzer gehört einer bekannten Gruppe an oder hat einen bekannten Namen



Multicast

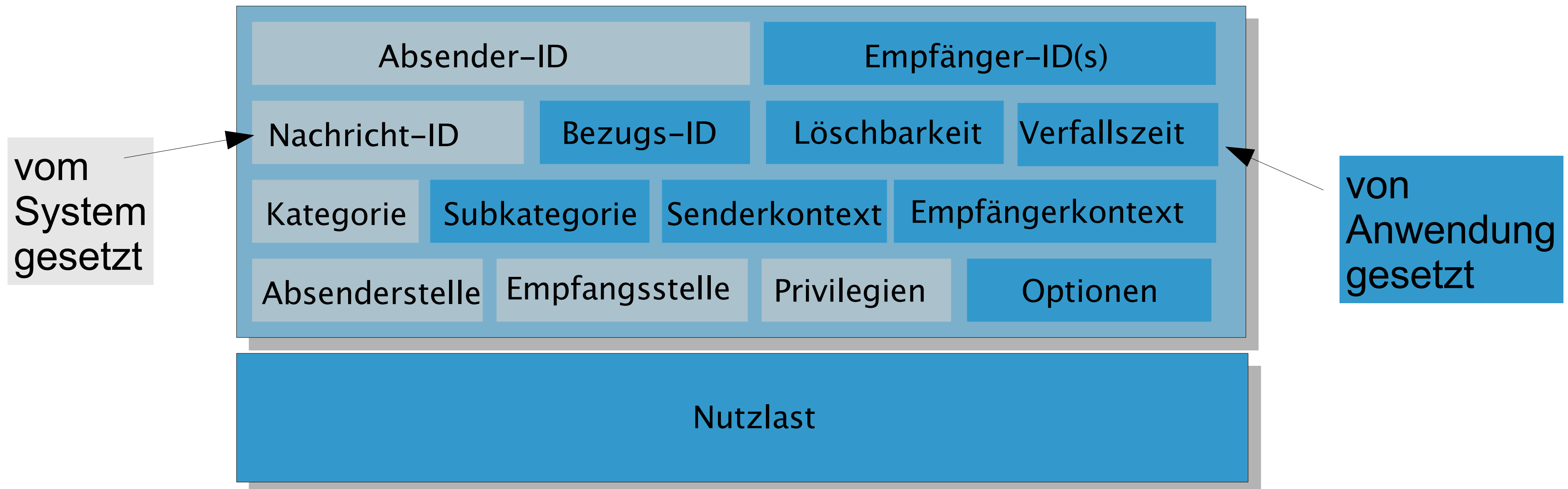
- Probleme
  - kein Authentifikationsmechanismus: Jeder unbekannte Agent von „außen“ kann einer Gruppe beitreten
  - unflexibel: führt eine „hartkodierte“ Struktur einer abstrakten Beschreibung ein (keine anderen Adressierungskriterien möglich außer Name/Gruppe)
  - Ressourcenproblem: Nachrichten müssen vervielfacht werden, wenn ein Gruppenmitglied eine Nachricht abholt; Nachrichtenspeicher kann zunehmend mehr Speicher verbrauchen
  - Performanzproblem: Bei jedem Abholen von Nachrichen muss das Postfachsystem alle „passenden“ Postfächer des Stellennutzers abfragen und alle Nachrichten zusammenhängen
  - Reihenfolgen: Es ist nicht gesichert, welche Nachricht aus mehreren Postfächern die „nächste“ ist

Multicast

- Verwendung in künftigen Anwendungen nicht mehr empfohlen
  - Frühere Anwendungen hängen strukturell von dieser Adressierung ab!
- Wie kann man unbekannte Stellennutzer sonst adressieren (z.B. einen erwarteten Agenten)?
  - 1. Möglichkeit: Stellennutzer-ID des erwarteten Agenten muss eher die Stelle erreichen als der Agent (Konfigurationsphase der Anwendung)
  - 2. Möglichkeit: Vermittlung (Mediation)

Nachrichten

- Nachrichten sind Instanzen einer Nachrichtenklasse



- Absender/Empfänger sind Stellennutzer-IDs
  - Empfänger ggf. Multicast-Adresse
- Nutzlast ist eine Liste von Objektreferenzen (Object[])

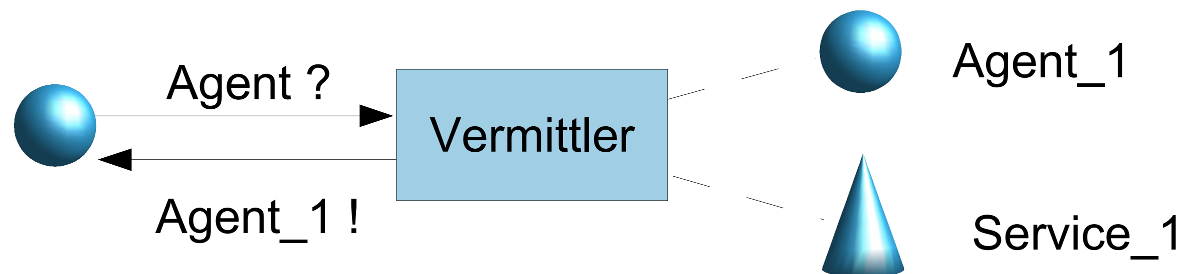
Nutzlast

- Problem der Serialisierung von beliebigen Objekten
  - Werden anwendungsspezifische Klassen verwendet, müssen diese Klassen zum Empfänger transportiert werden
  - Was macht man, wenn zwei Nachrichten eine Klasse gleichen Namens, aber unterschiedlicher Implementierung transportieren?

Die Nutzlast darf nur Instanzen von Klassen beinhalten, die über den Klassenladepfad des Systemklassenladers erreichbar sind („Systemklassen“). Instanzen anwendungsspezifischer Klassen werden nicht transportiert.

Mediation und Typisierung

- Mediation: Vermittlung
  - auch „Trading“ genannt
- Aufgabe
  - Gegeben sei eine abstrakte Beschreibung eines gesuchten Objekts
  - Übersetze diese abstrakte Beschreibung in eine konkrete Beschreibung eines oder mehrerer Objekte, wenn möglich
  - Eine konkrete Beschreibung charakterisiert das zugehörige Objekt eindeutig, d.h. man kann das Objekt nach dieser Vermittlung „erreichen“ (ansprechen, Methoden darauf aufrufen usw.)



Mediation und Typisierung

- Abstrakte Beschreibung
  - im Allgemeinen ein logisches Prädikat, welches eine Menge von Objekten beschreibt, die es erfüllen
  - Beschreibung wird auch „Typ“ genannt
  - entspricht in der implementierungszentrierten, objektorientierten Sicht den Klassen
- Alle Stellennutzer werden mit einem Typ versehen.
- Dieser Typ muss vom lokalen Mediator „verstanden“ werden.
- Stellenübergreifende Mediation nur auf Applikationsebene
  - Einführung eines Mediationsagenten denkbar

Mediation und Typisierung

- Zurzeit zwei Mediatoren
  - **TrivialServiceMediator**: kann „Stringtypen“ vergleichen, kann aber nur aktuell laufende Stellennutzer finden
  - **HybridTypeMediator**: kann komplexe Typen („Hybridtypen“, beinhalten syntaktische und semantische Informationen) vergleichen und auch lokal registrierte Stellennutzer, die noch nicht laufen, finden (und damit deren Start ermöglichen)
- Stringtyp
  - speziell strukturierte Zeichenkette

Ähnlich den Stellennutzer-IDs

„PersonalAgent # MichaelZapf / Agents“

Mediation und Typisierung

- Platzhalter erlauben Lockerung der Abfrage und damit größeren Ergebnisraum

„PersonalAgent # MichaelZapf / Agents"

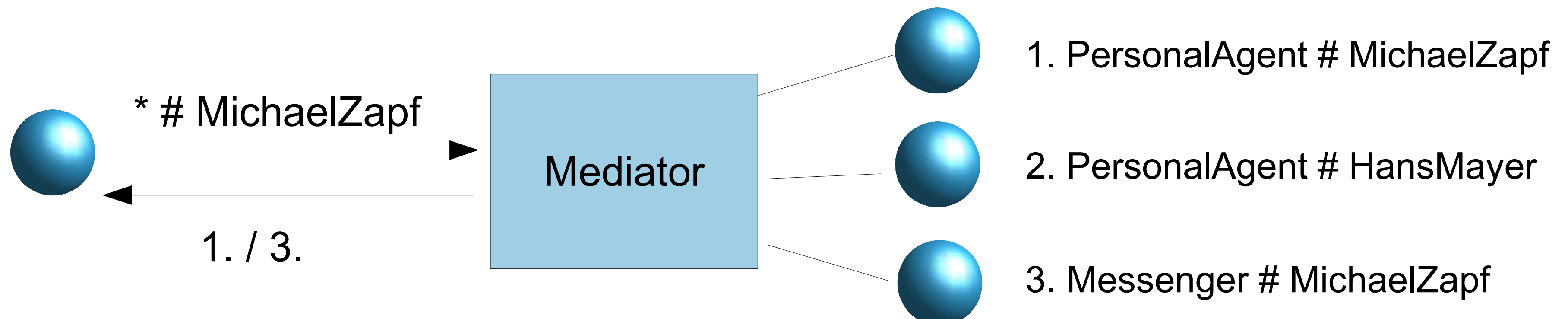
„PersonalAgent # * / Agents"

„PersonalAgent # MichaelZapf / *"

„PersonalAgent # * / *"

Sternchen können weggelassen werden

- Vorgang: Suche alle Agenten von „MichaelZapf"

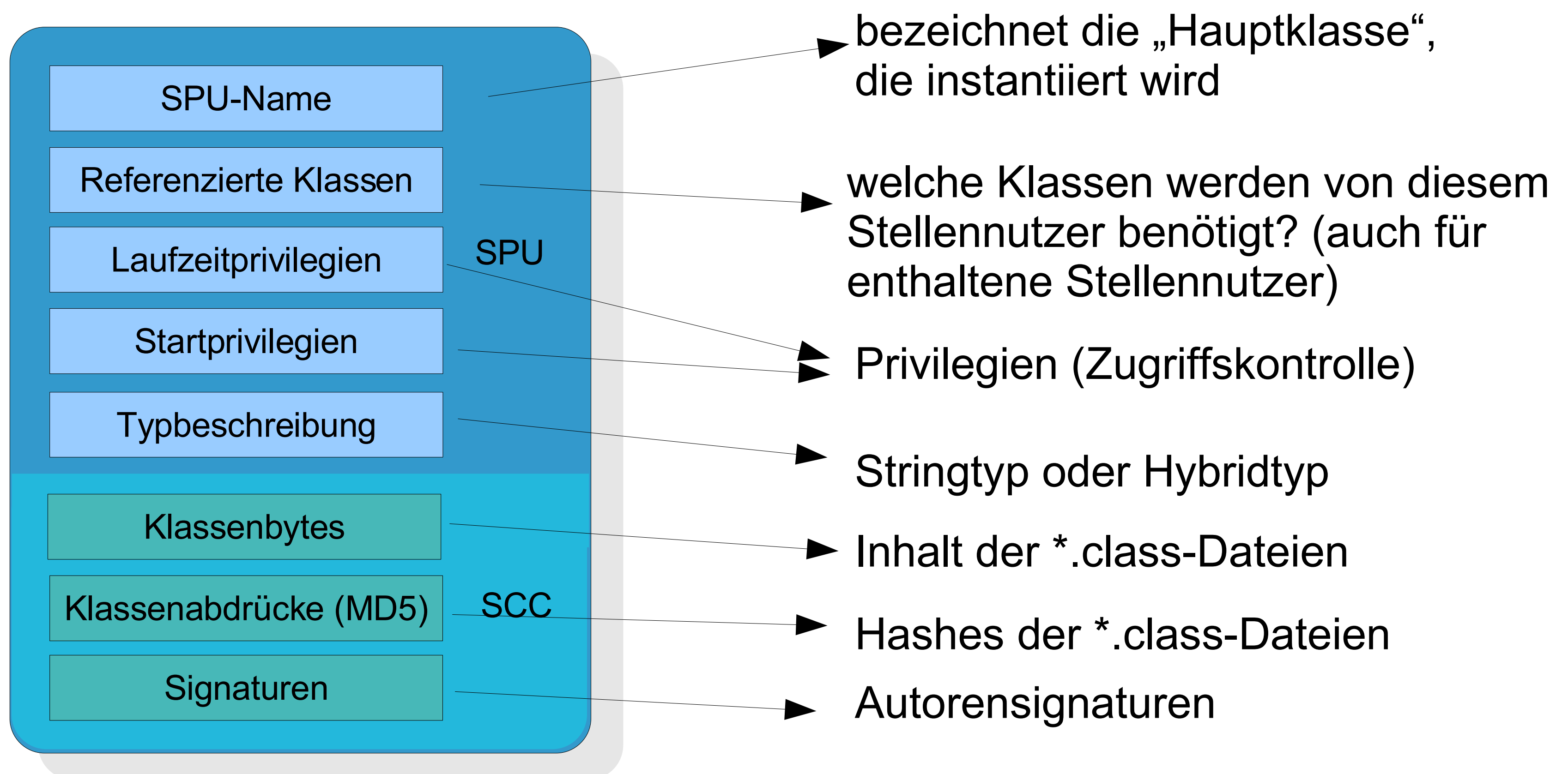


Hybridtypen

- Kurzer Überblick über Hybridtypen
  - **Syntaktischer Typ**: Beschreibt Nachrichtentypen, die der Stellennutzer akzeptiert oder sendet (anhand der Typen der Nachrichtenkomponenten)
  - **Transitionstyp**: Beschreibt Nachrichtenfolgen durch Zustandsübergänge
  - **Semantischer Typ**: Beschreibt „Bedeutungen“, also semantische Informationen (des gesamten Agenten oder von Nachrichten, Zuständen, Nachrichtenelementen)
- Hybridtypen sind geeignet, eine Typhierarchie auf Agenten zu etablieren
  - Subtypkonzept: Nutzung von spezielleren Agenten anstelle des präzise gesuchten Agenten

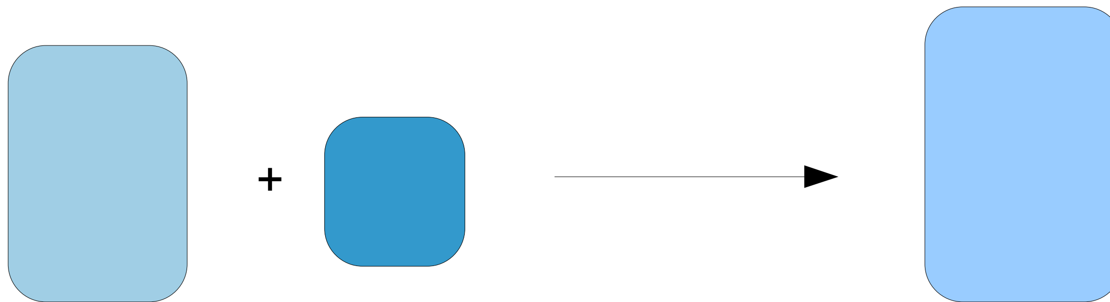
Stellennutzer-Container

- Signed Place User Container: „SPU“
  - Basisklasse „Signed Class Container“ für allgemeine Klassenpakete



Stellennutzer-Container

- Möglichkeit, SPUs zu schachteln (seit Version 2.6)



- Neuer SPU-Container beinhaltet neue Verzeigerungen sowie die Vereinigungsmenge der Klassen
- „Referenzierte Klassen“ führt Buch, welche Klassen zu welchem verschmolzenen SPU gehören
- Start des eingebetteten Agenten durch **spawnAgent**
  - nicht möglich, Dienste oder Benutzeradapter einzubetten

Implementierung von Stellennutzern

- Agenten und Benutzeradapter

```
public class MyAgent extends AMETASAgent {

    public MyAgent() {
        super("MyAgent");
        // Initialisierungen...
    }

    public void invoke() {
        // Verhalten ...
    }
}
```

Die **invoke**-Methode wird nach Instantiierung des Stellennutzers aufgerufen. Nach deren Verlassen wird der Stellennutzer entsorgt.

Implementierung von Stellennutzern

- Dienste

```
public class MyService extends AMETASServiceObject {

    public MyService() {
        ...
    }

   public void initService(Vector vctInit) {
      // Setzen der Initialparameter des Dienstes
   }

   public void startService(AMETASMessage msg) {
      // Verhalten ...
   }
}
```

initService wird nur Erzeugung des Dienstobjekts aufgerufen. **startService** wird bei jedem Empfang einer Nachricht aufgerufen.

Migration

- AMETAS erlaubt nur die schwache Migration

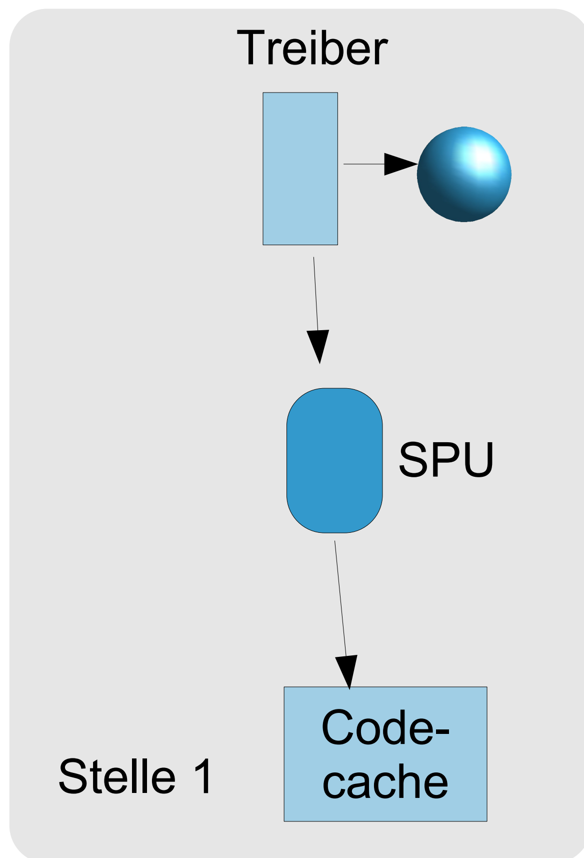
```
public class MyAgent extends AMETASAgent {

   public void invoke() {
        ...
        try {
           m_Driver.go("Place2");
        }
        catch (MigrationException mx) {
           // hat nicht geklappt
        }
   }
}
```

Die Ausführung wird an der neuen Stelle fortgesetzt, jedoch wird **invoke** wieder von vorne gestartet. Der Zustand bei Neustart von invoke ist derselbe wie nach dem Aufruf von **go**.

Migration

- Codetransport und Zustandstransport



Agent (Instanz) beinhaltet Zustand und wird vom Treiber referenziert

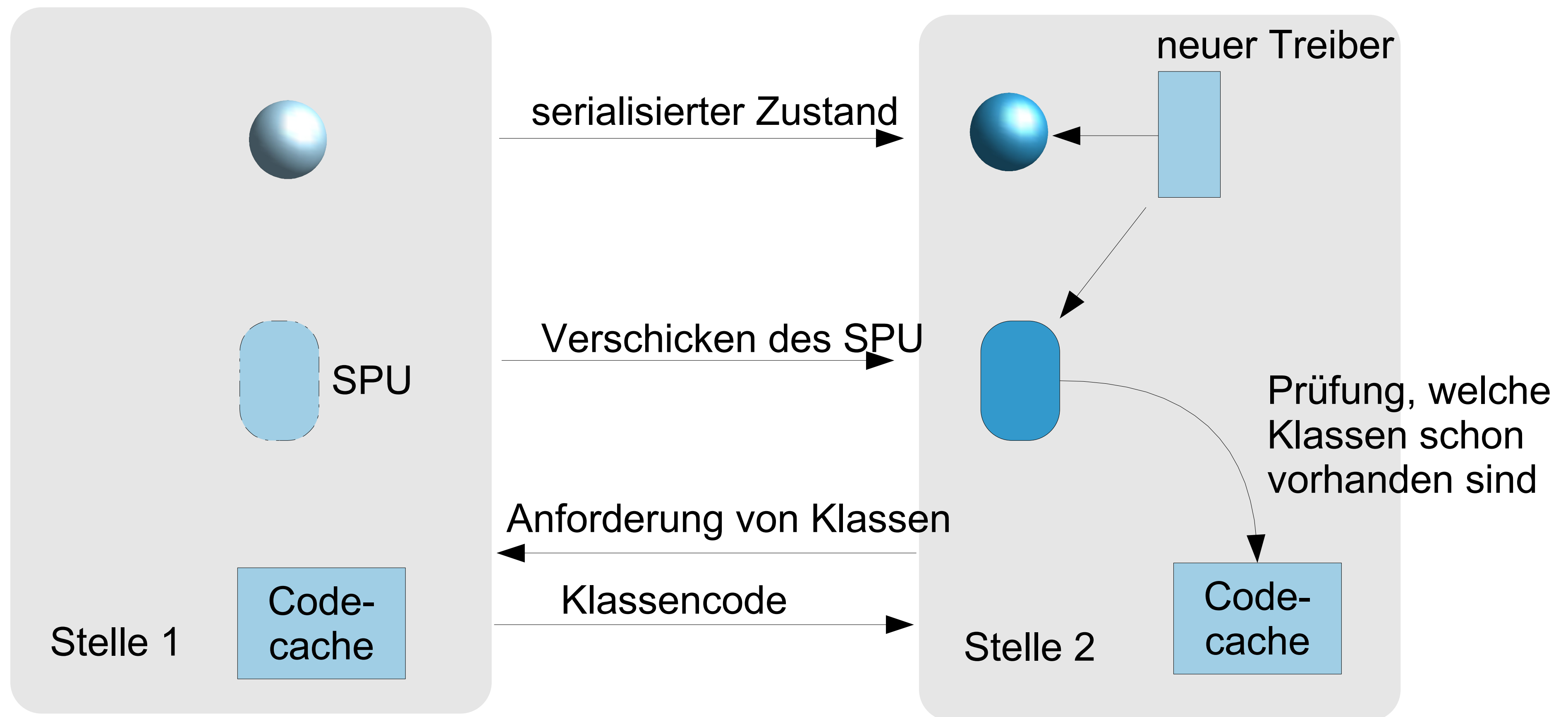
Treiber referenziert zugehörigen SPU

SPU beinhaltet Codeabdrücke

Codecache beinhaltet u.a. die Klassen dieses SPU

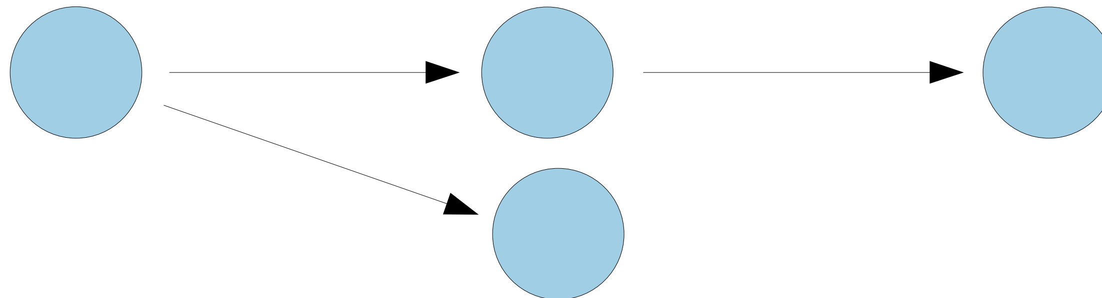
Migration

- Klassen werden nur bei Bedarf übertragen



Zustandsorientierte Programmierung

- Vereinfachung der Programmstruktur durch Verwendung von Zuständen und Zustandsimplementierungen
  - Schleife innerhalb von invoke
  - Auswahl des aktuellen Zustands, Ausführung des Zustands

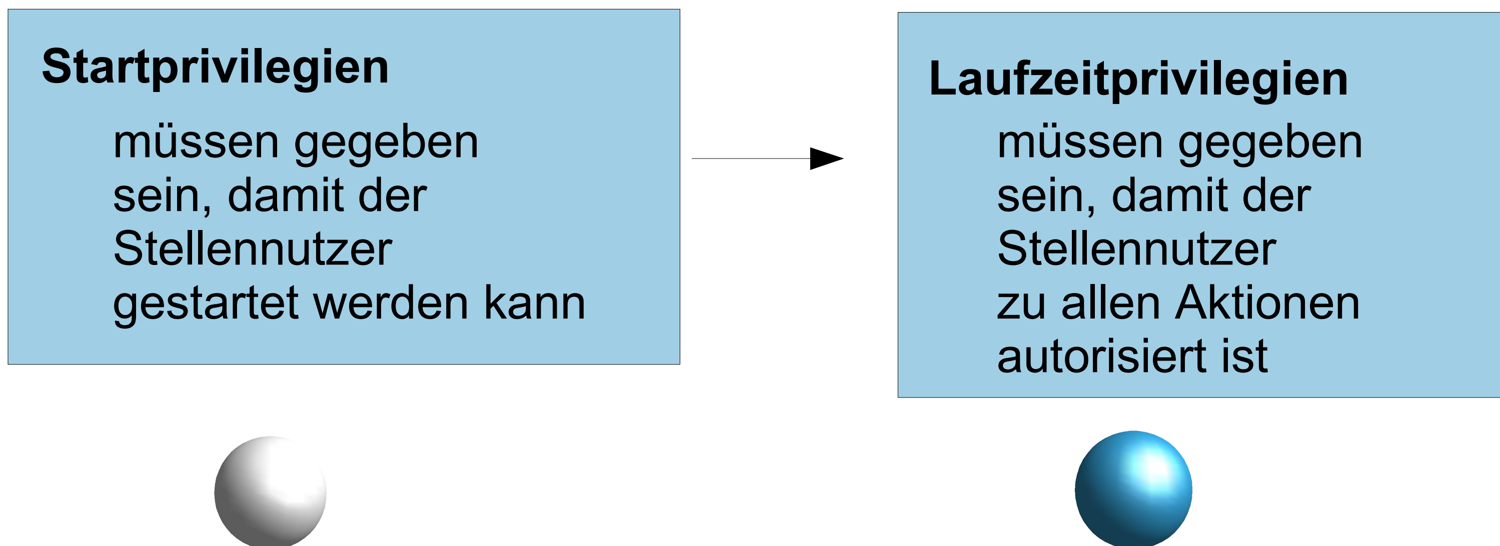


- Sehr hilfreich bei der schwachen Migration
  - denn der Fortschritt im Programm ist dann ein Zustandsübergang wie jeder andere auch

Kontextorientierte Programmierung → User's Guide

Privilegien

- Authentifikation und Autorisation
  - Verwendung eines RSA-Schlüsselpaares zur Authentifikation des Anwenders
  - Agent gilt dann als „von einem authentifizierten Benutzer gesendet“
  - Autorisation durch Privilegien (steuern auch die Zugriffskontrolle)



Privilegien

- Benannt durch eine Zeichenkette
  - z.B. ADMIN, RESFILEACC, NETACC
- Privilegien können Berechtigungen (für Systemzugriffe) enthalten
  - RESFILEACC = Zugriff auf Dateien in einem festen Teilbaum
- Privilegien können „leer" sein (keine Berechtigungen beinhalten)
  - dann dienen sie vor allem der Zugriffskontrolle auf Komponenten einer Anwendung (also z.B. Zugriff auf Dienste, Starterlaubnis für Agenten)
- Privilegien erhalten ihre Bedeutung nur lokal an der Stelle, an der sie definiert sind
  - daher kann die Zuteilung eines Privilegs an unterschiedlichen Stellen verschiedene Auswirkungen haben

Privilegien

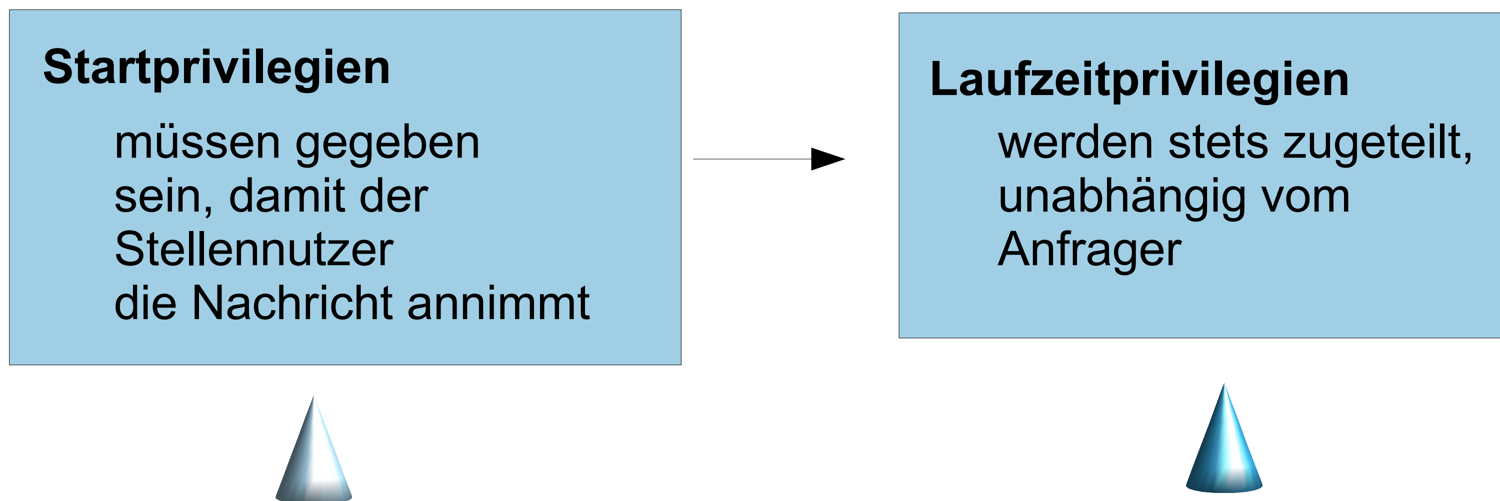
- Privilegien werden nur dem Anwender zugeteilt!
- Beim Startversuch wird geprüft, ob eine Liste von Privilegien dem Anwender zusteht.
- Die Menge der effektiven Laufzeitprivilegien sind die geforderten Privilegien (gemäß SPU), geschnitten mit der Menge der dem Anwender zugeteilten Privilegien

> Der Anwender, nicht der Agent, ist der Prinzipal (Rechteträger).
> Die Rechte werden an jeder Stelle lokal definiert.
> Die Autorisation findet an jeder Stelle erneut statt.

- Agenten können unterschiedliche Privilegierungen erfahren
  - Administratorberechtigung an einer Stelle, an der nächsten schon nicht mehr

Privilegien

- Spezialfall Dienste
  - Dienste werden immer von der lokalen Stelle gestartet und in deren Namen ausgeführt



  - erlaubt es, Dienste als Systemerweiterungen zu betreiben, ohne dass der Anwender die Rechte haben muss
  - ähnlich wie „suid“ (set-user-id) in UNIX-Dateisystemen

Sicherheit

- Angriffspunkte
  - Kommunikation zwischen den Stellen
  - Auf Datenspeichern befindlicher Code
  - Übergriffe zwischen den Stellennutzern
  - Unberechtigter Zugriff auf Ressourcen
  - Fehlverhalten der Stelle
  - Replikation von Agenten
  - Übermäßiger Ressourcenverbrauch durch Stellennutzer
  - Änderung des Migrationspfads (Marschroute, "itinerary")
  - Vorgabe, eine bestimmte Stelle zu sein
  - Vorgabe, ein bestimmter Stellennutzer zu sein
  - Defekter Stellennutzer

Sicherheit

- Kommunikation zwischen Stellen
  - Etablierung eines verschlüsselten Kanals zwischen den Stellen, ähnlich SSL (asymmetrisch verschlüsselte Aushandlung eines symmetrischen Schlüssels)
- Auf Datenspeichern befindlicher Code
  - Stellennutzer sind in SPUs gespeichert
  - SPUs tragen Autorensignaturen auf ihrem Code
  - Änderungen sind damit feststellbar
- Übergriffe zwischen den Stellennutzern
  - Verwendung eigener Klassenlader
  - Stellennutzer können sich keine Referenzen aufeinander zuschicken
  - Referenz wird von Stelle nicht an einen anderen Stellennutzer weitergegeben

Sicherheit

- Unberechtigter Zugriff auf Ressourcen
  - Agenten können auf Systemressourcen nicht zugreifen.
  - Dienste können dies, sie müssen aber lokal installiert werden (also unter Verantwortung des lokalen Administrators).
  - Benutzeradapter können ebenfalls einige Ressourcen adressieren, sie können dies aber nur auf dem lokalen Rechner.
- Fehlverhalten der Stelle
  - Malicious-Host-Problem
  - Stelle ist selbst modifiziert oder wird von anderen Stellennutzern modifiziert
  - In AMETAS nicht möglich, solange die Originalbibliotheken verwendet werden. Dies kann jedoch i.A. **nicht garantiert** werden!
  - keine Abwehr in AMETAS

Sicherheit

- Replikation von Agenten
  - Agent wird ein zweites Mal an eine Stelle geschickt (von einem Angreifer in der Mitte)
  - Funktioniert nur, wenn der Angreifer eine Stelle simulieren kann
  - Indirektes "Malicious-Host-Problem"
  - keine Abwehr in AMETAS
- Übermäßiger Ressourcenverbrauch
  - while(true) {}
  - In Java gibt es keine plattformübergreifende Methode zur Feststellung der Systembelastung
  - keine Abwehr in AMETAS

Sicherheit

- Änderung der Marschroute
  - Eine Marschroute kommt in AMETAS nur auf Anwendungsebene vor.
  - Der Agent muss daher selbst für die Einhaltung und die Überprüfung der Marschroute sorgen.
  - keine Abwehr in AMETAS
- Vorgabe, eine bestimmte Stelle zu sein
  - "PNS-Spoofing"
  - PNS-Server sind für lokale Angriffe durch die jeweiligen berechtigten Personen auf den Rechnern anfällig
  - Einsatz eines fremden PNS nur bei Kontrolle über die Konfiguration der jeweiligen Stelle

Sicherheit

- Vorgabe, ein bestimmter Stellennutzer zu sein
  - Typbeschreibung wird vom Autor mitsigniert
  - Bei sicheren Umgebungen können Stellennutzer fremder Autoren ausgeschlossen werden.
  - Stellennutzer-ID ist **nicht immun** gegen Fälschung.
  - Indirektes Malicious-Host-Problem.
- Defekter Stellennutzer
  - z.B. erhöhter Ressourcenverbrauch, stoppt nicht
  - zurzeit „abschießbar“: System sieht Stopp der gesamten Threadgruppe durch **Thread.stop** vor
  - aber „deprecated“! Lösung steht noch aus.

Weitere Informationen

- www.ametas.de
- Aktuelle Version: 2.6.14
- In Vorbereitung: 2.7
  - Nutzung der aktuellen Java-Sicherheitsarchitektur (JSA)
    - keine eigenen Permission-Klassen mehr
  - Nutzung der aktuellen Java-Kryptounterstützung (JCE)
- Ausblick: 3
  - andere Adressierungsschemata (z.B. X500)
  - möglicherweise FIPA-konform
  - keine Thread.stop-Verwendung mehr
  - Vereinfachung des Nachrichtensystems (noch immer asynchron, aber auf Unicast optimiert)