Universität Dortmund
Lehrstuhl Verteilte Systeme

Dr. Michael Zapf

Michael.Zapf@cs.uni-dortmund.de
Michael.Zapf@uni-kassel.de

vkiv@ls4.cs.uni-dortmund.de

Übung zur Vorlesung „Verteilt-kooperative Informationsverarbeitung“ - SS 2007

Lösungen zu Blatt 4

Aufgabe 4.1 – 10 Punkte

a) Warum genügt der MAF-Spezifikation bei Stellen ein String, während für Agentensysteme ein Location-Objekt gewählt wird? (2 Punkte)

Das Location-Objekt ist selbst nichts anderes als ein String (siehe Spezifikation), der aber zur Adressierung im System benötigt wird. Die Unterscheidung dient der Typsicherheit. Mithilfe der Lokationsangabe muss die Position im Netz gefunden werden. Das Location-Objekt dient also der Adressierung des Agentensystems und muss eine physische Adresse beinhalten. Der Stellenname ist symbolisch und bezeichnet die Stelle im System. Durch Registrierung beim MAFFinder entsteht eine Abbildung des Stellennamens auf das Location-Objekt des Systems, das die Stelle beinhaltet. Das heißt, der Zugang zur Stelle erfolgt immer indirekt; der Stellenname selbst beinhaltet per se keine Lokationsinformation. So muss eine Stelle nicht einmal als CORBA-Objekt repräsentiert werden, solange das Agentensystem ein CORBA-Objekt ist.

b) Warum führte man den MAFFinder ein, obwohl es bereits einen CORBA-Namensdienst (CosNaming) gibt? (3 Punkte)

Das CosNaming-System vermittelt CORBA-Objekte und kann in dieser Funktion auch nur CORBA-spezifische Eigenschaften suchen. Das Suchen nach Profildaten des Agenten ist nicht vorgesehen. Weiterhin bekräftigt die Spezifikation, dass Agenten nicht als CORBA-Objekte auftreten müssen. Daher ist die Nutzung eines separaten Namensdienstes, der zudem dem häufigen Wechsel der Position von Agenten im System Rechnung trägt, vorzuziehen. Agenten, die jedoch als CORBA-Objekte agieren möchten, können sich zusätzlich bei CosNaming anmelden.

c) Nehmen Sie an, dass ein Rechner nur temporär mit dem Internet verbunden wird und bei jeder Einwahl eine neue IP-Adresse erhält. Dieser Rechner betreibe ein MAF-Agentensystem. Beschreiben Sie das sich ergebende Problem sowie eine mögliche Abhilfe. (5 Punkte)

MAF verwendet Location-Angaben, um Agentensysteme zu lokalisieren. Eine Location-Angabe liefert die Netzposition der Zielstelle, auf welcher ein „receive_agent“ aufzurufen ist, um den Agenten zu transferieren.

Ist diese Zielstelle auf einem Dialup-System, so wird der alte Eintrag ungültig, und Agenten, welche dorthin zu migrieren versuchen, werden eine Fehlermeldung erhalten. Erst wenn das System seinen Eintrag im MAFFinder gelöscht und neu eingetragen hat, können die Agenten die Stelle wiederfinden. Sie müssen aber auf jeden Fall einen neuen Aufruf beim MAFFinder durchführen, um ein neues Location-Objekt zu erhalten. Damit ist es prinzipiell erforderlich, vor jeder Migration eine Abfrage beim MAFFinder durchzuführen, wenn die Zielstelle gelegentlich die Netzlokation ändert.

Eine mögliche Abhilfe wäre die Angabe eines Ersatzsystems, zu dem die Agenten migrieren können, wenn ihr eigentliches Zielsystem nicht erreichbar ist. Diese Angabe kann den Agenten mitgegeben werden, oder sie verwenden einen speziellen Dienst, um die Ersatzposition zu erhalten. Alternativ könnte der MAFFinder so ausgestattet werden, dass er das Location-Objekt des Ersatzsystems liefert, wenn das primäre System nicht erreichbar ist. Dazu müsste die Standardfunktionalität entsprechend erweitert werden.

Aufgabe 4.2 – 10 Punkte

Implementieren Sie mittels Pseudocode die Migration zwischen zwei MAF-konformen Stellen. Gehen Sie davon aus, dass die Systeme eine CORBA-Java-Abbildung verwenden, in der Sie die entfernten Methoden auf einem Stub-Objekt so aufrufen können, wie sie in der IDL-Datei beschrieben sind.

Für die Unterstützung der Mobilität sieht MAF zwei Methoden vor:

- MAFAgentSystem.receive_agent(in Name agent_name, in AgentProfile agent_profile, ...), aufgerufen auf dem Zielsystem
- MAFAgentSystem.fetch_class(in ClassNameList class_name_list, ...), aufgerufen auf dem Ursprungssystem

Der Agent startet also mittels einer „migrate“-Methode die Migration:

Zu beachten ist, dass MAF die Verwendung einer CORBA-Infrastruktur voraussetzt. Das heißt, die entfernte Stelle ist jene, auf der die receive_agent-Methode aufgerufen wird, und zwar durch die sendende Stelle. Umgekehrt muss die fetch_class-Methode auf der sendenden Stelle durch die empfangende Stelle aufgerufen werden.

Der Agent selbst löst die Migration aus, indem er sinnvollerweise eine Methode auf der eigenen Stelle aufruft, welche sich dann der MAF-Methoden bedient. Diese Methode sei „migrate“ genannt.

public void migrate(Agent ag, String sPlacename);

Wir gehen davon aus, dass wir ein „naming“-Objekt kennen, das aus einer Location-Instanz eine Objektreferenz bilden kann (mittels „bind“).

Im Folgenden stellen wir einen Java-basierten Ansatz vor, der in dieser Form zwar noch Pseudocode-Charakter hat, jedoch die wesentlichen Aspekte aufweist.

Für weitere Informationen empfehlen wir die Einführung unter:

http://java.sun.com/javase/6/docs/technotes/guides/idl/GShome.html

---

Agent:

```
public class Agent {
  ...
  public void invoke() {
    ...
    place.migrate(this, sPlacename);
  }
}
```

Place (Sender):

```
public class MyPlace implements Place {
   MAFAgentSystem m_masContainingSystem;

  /** Die vom Agenten aufgerufene Methode. */
  public void migrate(Agent ag, String sPlacename) {

    MAFFinder finder = m_masContainingSystem.get_MAFFinder();
    /* Verweist zunächst auf das entfernte Agentensystem; kann aber auch Hinweise
       auf die Stelle beinhalten (je nach Adressierungsschema). Wir gehen davon aus, dass es keine
       Hinweise auf die Stelle selbst im Location-Objekt gibt. */
    Location locDistantSystem = finder.lookup_place(sPlacename);

    /* Wir nehmen an, dass das naming-Objekt uns eine Referenz liefert. Tatsächlich wird dabei
       der lokale Stub von mafASReceiver an den entfernten Stub gebunden und die Referenz auf
       den lokalen Stub geliefert. */
    MAFAgentSystem mafASReceiver = naming.bind(locDistantSystem);
    try {
       m_mafFinder.unregisterAgent(ag.getName());
       mafASReceiver.receive_agent(ag.getName(), ag.getProfile(), ag.getState(),
                                   sPlacename, ag.getClassNames(), ag.getCodeBase(),
                                   m_masContainingSystem);

       // Agent muss nun systemspezifisch entsorgt werden.
     }
     catch (Exception e) { ... }
  }
}
```

Dies hier ist nun das Agentensystem, das die sendende Stelle oder die empfangende Stelle beinhaltet.

```
public class MyAgentSystem implements MAFAgentSystem {

  /** Referenz auf den MAFFinder */
```

```
  MAFFinder m_mafFinder;

  /** Wird von empfangender Stelle aufgerufen. */
  public List<byte[]> fetch_class(ClassNameList cnl, String sCodebase, AgentProfile profile) {
    List<byte[]> classes = new LinkedList<byte[]>();
    CodeProvider prov = getProvider(sCodebase);  // entsprechend zu implementieren
    for (ClassName cl : cnl.classes()) {
        classes.add(prov.getBytes(cl);
    }
    return classes;
  }

  /** Wird von sendender Stelle aufgerufen. */
  public void receive_agent(Name agentName, AgentProfile profile, byte[] agent, String sPlace,
                            ClassNameList lst, String sCodeBase, MAFAgentSystem origPlace) throws ... {
    // sind alle Klassen da?
    ClassNameList lstRequired = checkForMissingClasses(lst);
    if (lstRequired.size()>0) {
      // OctetStrings hier als List<byte[]> umgesetzt
      List<byte[]> classes = origPlace.fetch_class(lstRequired, sCodeBase, profile);
      ...
    }
    m_mafFinder.registerAgent(agentName, this.location, profile);
    // Agent muss nun systemspezifisch instanziiert werden.
  }
}
```

Aufgabe 4.3 – 15 Punkte

Herr Meier möchte eine bestimmte Ware kaufen und schickt einen Agenten los, um sich ein Angebot einzuholen. Er startet seine Anwendung (ANW). Ein lokales Agentensystem „MeiersSystem“ ist bereits am Laufen, und ANW nutzt dieses System, um weitere Agentensysteme zu finden.

Genauer gesagt benötigen wir die Stelle „MeierHome“ auf einem der erreichbaren Agentensysteme (deren Namen nicht bekannt ist). Diese Stelle wurde zuvor gestartet und hat sich registriert.

```
MeierHome: {

   // gerade gestartet
   MAFFinder finder = eBizSystem.get_MAFFinder();
   finder.register_place("MeierHome", eBizLocation);

}

ANW: {

    ...

    MAFFinder finder = meiersSystem.get_MAFFinder();
    //     Locations locs = finder.lookup_agent_system();
    Location locSystemMH = finder.lookup_place("MeierHome");

    // Wir haben bislang kein "naming"; dies sei ein Dienst, der aus "Location" eine IOR produziert und auch
    // gleich die Bindung an den entfernten Dienst vornimmt.
    MAFAgentSystem eBiz = naming.bind(locSystemMH);
    ...
```

Ein Agent wird auf „MeierHome“ gestartet. Ihm werden zum Start von ANW Parameter übergeben. Er registriert sich. Der Agent soll nun herumreisen. ANW hat dem Agenten eine Route mitgegeben.

```
ANW (Fortsetzung):
   ...
   // profile entsprechend definiert; code ist der Klassencode des Agenten.
   // agentname entsprechend definiert; identity = "Agent"
   // args = { MAFFinder finder, String[] route, eBiz }
   Name agent = eBiz.createAgent(agentname, profile, code, "MeierHome", args, null, codebase, this);
   ...


Agent: {

  ...
```

```
MAFFinder finder = getSystem().get_MAFFinder();
finder.register_agent(agentname, eBiz, profile);
...
```

Der Agent reist zum nächsten Ort. Er ruft eine migrate-Methode auf seiner Stelle auf und löst damit die Migration aus. Der Zielort verfügt jedoch nicht über alle Klassen und bittet das abschickende System, die Klassen nachzuschicken.

Agent (Fortsetzung):

```
...
index++;
eBiz.migrate(route[index].place, route[index].system);
...
```

Aktuelles System:

```
...
public void migrate(String place, Location locSystem) {
   MAFAgentSystem nextSys= naming.get(locSystem);
   nextSys.receive_agent(agentname, profile, agentstate, place, classnames, codebase, this);
}
...
```

nextSys:

```
...
public void receive_agent(..., MAFAgentSystem mafLast) {
    // Code liegt hier nicht vor
    ClassBytes[] acb = mafLast.fetch_class(classlist, codebase, profile);
    ...
}
```

Es ist ein wenig Zeit vergangen, und Herr Meier möchte wissen, ob sein Agent noch lebt oder irgendwo abgestürzt ist. Es ist aber nicht bekannt, auf welcher Stelle der Agent weilt, lediglich die Menge der Agentensysteme ist bekannt. ANW geht systematisch vor und testet diese Agentensysteme der Reihe nach.

ANW (Fortsetzung):

```
...
for (system in route) {
   NameList names = system.list_all_agents();
   for (name in names) {
      if (name.equals(agentname)) {
          AgentStatus as = system.get_agent_status(name);
          if (as==CfMAFRunning) {
              ...
          }
      }
   }
}
...
```

Der Agent ist gefunden und läuft noch.

```
... Location systemWithMyAgent = system;
s.o.
```

ANW kontaktiert den Agenten, der seine Daten applikationsspezifisch liefert. (Wir gehen davon aus, dass der Agent die speziellen Methode „getData“ und „setData“ besitzt, um Daten zu liefern oder entgegenzunehmen.) Diese Daten verraten, dass er ein Agentensystem ausließ.

ANW (Fortsetzung):

```
...
byte[] agentstate = system.getDataFromAgent(name);
...
Location systemNotVisited = ... // jenes, das er nicht besuchte.
...
```

```
System:
  ...
  byte[] getDataFromAgent() {
     return agent.getData();
  }
  ...
```

Das ausgelassene System nimmt offenbar zurzeit keine Agenten an, ist aber ansprechbar. Er lässt den Agenten über ANW anhalten.

```
ANW (Fortsetzung):
  ...
  systemWithMyAgent.suspend_agent(agentname);
  ...
```

ANW bittet den Agenten, eine geeignete Stelle in der Nähe (des ausgelassenen Systems) aufzusuchen.

```
  ...
  Location locAlternative = systemNotVisited.find_nearby_agent_system_of_profile(profile);
  systemWithMyAgent.setDataOfAgent(agentname, locAlternative);
  ...


System:
  public void setDataOfAgent(Agent a, Object data) {
     a.setData(data);
  }
```

ANW lässt den Agenten weiterlaufen.

```
ANW (Fortsetzung):
  ...
  systemWithMyAgent.resume_agent(agentname);
  ...
```

Der Agent hat alle Angebote eingesammelt und kehrt zurück zu „MeierHome“.

```
Agent:
  ...
  currentSystem.migrate("MeierHome");
  ...
  s.o.
```

Er liefert Herrn Meier die Angebote, und ANW stoppt den Agenten endgültig.

```
ANW (Fortsetzung):
  ...
  eBiz.terminate_agent(agentname);
  ...
}
```

Aufgabe 4.4 – 12 Punkte

Nun versuchen wir, das Szenario aus Aufgabe 4.3 in AMETAS zu implementieren. Dabei sollte wie folgt vorgegangen werden:

a) Analysieren Sie, welche Agenten beteiligt sind. Welche Rolle spielt der Anwender? (2 Punkte)

Wir benötigen einen Agenten, der die Route verfolgt und sich Angebote holt (BuyerAgent) sowie einen Agenten, den wir später zum Anhalten des BuyerAgent brauchen. Die Angebote holt sich der Agent von den jeweils lokalen Diensten.
Der Anwender übergibt die Route und die Parameter des Angebots mittels eines Programms, in AMETAS typischerweise mittels eines Benutzeradapters, an den BuyerAgent. Er ist derjenige, der über seinen Adapter den Agenten anhält oder startet, der den Auftrag anfangs erteilt und am Ende das Ergebnis entgegennimmt.

b) Analysieren Sie die Anwendungsfälle, die im System auftreten. (2 Punkte)

Anwendungsfall 1: Starten der Anwendung

Anwendungsfall 2: Erzeugen des Auftrags

Anwendungsfall 3: Übermittlung des Auftrags an den Agenten.

Anwendungsfall 4: Überprüfung des Status des Agenten während der Abarbeitung, einschließlich Beeinflussung seines Verhaltens. Dies kann genauer spezifiziert werden:

4.1. Formulieren einer Statusüberprüfungsnachricht
4.2. Übermitteln dieser Nachricht und der Route an einen Botenagenten
4.3. Entgegennahme der Information des Boten bei seiner Rückkehr
4.4. Übermitteln einer aktualisierten Nachricht an den Botenagenten, der diese zum Agenten bringt, außerdem Ingangsetzen des entfernten Agenten

Anwendungsfall 5: Entgegennahme der Ergebnisse über den Benutzeradapter

Anwendungsfall 6: Terminierung des Agenten und der Anwendung.

c) Welche Interaktionen treten auf? Stellen Sie die Nachrichten zusammen sowie die zugehörigen Protokolle (Nachrichtenabfolgen). (3 Punkte)

Wir stellen das Protokoll textuell in der Weise dar, wie sie in der Vorlesung vorgestellt wurde. Eine Darstellung mit Sequenzdiagrammen ist natürlich auch akzeptabel.

```
auftrag: { route:String[], ware:String, parameter:String };
warenanfrage: { ware:String, parameter:String };
angebot: { parameter:String };
anhalten: { "anhalten_und_status" };
weiter: { "weiter_mit_neuen_daten", parameter:String };
terminieren: { "ende" };
ergebnis: { String };
```

Benutzeradapter:

```
Start = none > Warten:auftrag(BuyerAgent);          // Auftrag abschicken
Warten = ergebnis(BuyerAgent) > Ende;
Warten = none > WarteAufStatus:anhalten(Messenger);    // beauftrage den Mess., den Agenten anzuhalten
WarteAufStatus = ergebnis(Messenger) > Warten:weiter(Messenger);
```

BuyerAgent:

```
Start = auftrag > Aktiv;
Aktiv = none > Aktiv:migrate(place);                          // Weiterwandern
             + Anfrage: warenanfrage(Anbieterdienst);         // oder Dienst ansprechen
             + Fertig:ergebnis;                               // oder Ergebnis mitteilen
Aktiv = anhalten(Messenger) > Warten:ergebnis(Messenger);
Warten = weiter(Messenger) > Aktiv;
Anfrage = angebot(Anbieterdienst) > Aktiv;
```

Messenger:

```
Start = anhalten > AgentSuchen;
Start = weiter > AgentNochmalSuchen:migrate(place);
AgentSuchen = none > AgentSuchen:migrate(place)
                   + Gefunden:anhalten(Agent);
Gefunden = ergebnis(Agent) > Heimreise;
Heimreise = none > Heimreise:migrate(place)
                 + Fertig:ergebnis;
AgentNochmalSuchen = none > AgentNochmalSuchen:migrate(place)
                          + Ende:weiter(Agent);
```

Anbieterdienst:

```
Start = warenanfrage > Ende:angebot;
```

d) Entwerfen Sie ein System von Stellen, zwischen denen der Agent reisen soll. Es sollen mindestens zwei verschiedene Stellen definiert sein. (1 Punkt)

Definiere Stellen:

1. Heimatstelle
2. Place1 (eine Zielstelle)
3. Place2 (eine weitere Zielstelle)

Auf den Zielstellen muss jeweils ein Anbieterdienst installiert sein. Die Heimatstelle beherbergt den Benutzeradapter. Die Stellen können temporär oder permanent sein.

Der BuyerAgent wird auf der Heimatstelle gestartet, reist zu den anderen Stellen und kehrt dorthin zurück.

e) Welche Stellennutzer kommen für welche Komponenten des Systems zum Einsatz? (2 Punkte)

Der **Benutzeradapter** dient als Schnittstelle zum Anwender und wird verwendet, um

- den Auftrag an den BuyerAgent zu formulieren
- den BuyerAgent lokal zu starten
- den Messenger lokal zu starten und eine Anweisung zur Überprüfung des BuyerAgent zu formulieren
- das Ergebnis darzustellen

Der **BuyerAgent** migriert gemäß seiner Reiseroute autonom von Stelle zu Stelle.

Der **Messenger** ist zwar in obigem Szenario nicht erwähnt, in AMETAS aber notwendig, um dem entfernten Agenten eine Nachricht zuzustellen. Er übernimmt die Steuerungsfunktionen des MAF-Systems in Bezug auf die Beeinflussung des BuyerAgents.

Der **Anbieterdienst** wartet auf Anfragen des BuyerAgent und liefert Ergebnisse, die der BuyerAgent mitnimmt.

f) Entwerfen Sie eine einfache Benutzerschnittstelle. (2 Punkte)

Hier sind verschiedene Ansätze denkbar – wie üblich bei Benutzerschnittstellen.

Beispiel: Man benötigt ein Eingabefeld für die gesuchte Ware sowie den akzeptablen Preis, außerdem eine Liste der zu besuchenden Stellen. Weiterhin ist eine Darstellung des Agentenzustands notwendig (welcher vom Messenger überbracht wird). Der Start des Messengers kann mit einem einfachen Knopf geschehen (etwa "Aktueller Stand"). Schließlich sollte vorgesehen sein, dem Agenten eine Nachricht zukommen zu lassen; etwa, dass er eine andere Stelle besuchen soll. Auf die grafische Darstellung sei an dieser Stelle verzichtet.

Aufgabe 4.5 – 10 Zusatzpunkte (freiwillig)

Aufgabe gestrichen