8. Aufgabenblatt

Objekt-Migration, JavaParty

Galactic Traders 2

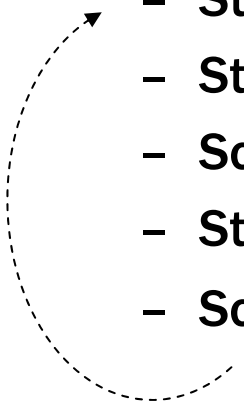
- Erweiterte Spezifikation
  - Auseinandersetzungen mit Polizei und Piraten
    - `Station.encounter`
    - Schiff besitzt Verteidigungsmaßnahmen und Panzerung
      - `Ship.getDefenseLevel`, `Ship.changeDefenseLevel`
      - `Station.equip`
      - `Ship.getArmor, Ship.changeArmor`
      - `Station.getRepairPrice, Station.repair`
    - Angreifer besitzen Angriffsstärke
      - abhängig vom TechLevel des Systems

Galactic Traders 2

- Erweiterte Spezifikation
  - Strafregister
    - `Ship.getCriminalRecord`, `Ship.changeCriminalRecord`
    - illegaler Handel erhöht Strafpunkte
    - Hyperraumsprung reduziert sie
    - Polizei greift ab bestimmter Punktezahl an

Galactic Traders 2

- Schiffe sind passive Objekte
  - ermöglicht problemlose Migration mit JavaParty
  - Schiff wird an Station übergeben (`enter`, asynchron!)
  - Station löst Begegnungen aus (`encounter`)
  - Station aktiviert "KI" des Schiffes (`interact`)
  - Schiff teilt Station neues Ziel mit (Rückgabewert `interact`)
  - Station löst Hypersprung aus (`hyperjump`)
  - Schiff wird an nächste Station übergeben...



zur Erinnerung: asynchrone Methode

```
public class Test {
    public void handle(final Object o) {
        new Thread() {
            public void run() {
                try {Thread.sleep((int)(Math.random() * 1000));}
                catch (Exception e) {}
                System.out.println("o: " + o);
            }
        }.start();
    }
}
```

JavaParty

- JavaParty
  - http://wwwipd.ira.uka.de/JavaParty
  - erweitertes Java mit hoher Verteilungsabstraktion für Parallelrechnen im Lokalnetz – ohne (sichtbaren) RMI-Code
- JavaParty-Umgebung
  - erstreckt sich über mehrere Hosts
  - Objekte von `remote`-Klassen sind fernaufrufbar
  - statische Attribute sind von überall aus erreichbar, statische Methoden sind verwendbar
- Möglichkeiten zur Objekt-Platzierung
  - basierend auf statischer Code-Analyse
  - explizite Fernerzeugung
  - explizite Migration

explizite Ortsangabe und Migration

```
import jp.lang.*;
public remote class Station {
    private String name;
    public Station(String name) { this.name = name; }
    public void location() {
        System.out.println(name + " at " + DistributedRuntime.getMachineID());
    }
    public static void main(String[] args) {
        /** @at 0 */   // auch Variable verwendbar
        Station s1 = new Station("alice");
        s1.location();

        /** @at 1 */
        Station s2 = new Station("bob");
        s2.location();

      DistributedRuntime.migrate(s1, DistributedRuntime.getLocation(s2));
      s1.location();
   }
}
```

JavaParty

- Versionen:
  - Windows/Unix: nur unterschiedliche Archivformate
  - KaRMI/RMI: eigener performanter RMI-Ersatz, ermöglicht Synchronisation von Threads in verteilten Systemen
  - (experimentelle Java-5-Unterstützung)
- Installation
  - `bin`-Verzeichnis in den Pfad
  - `CLASSPATH` für eigene Klassen

JavaParty

- Übersetzen:
  - `jpc <source-files>`
- Aufsetzen Variante A (Unix/Windows):
  - zentralen Verwalter starten:
    `jprm [-verbose] [-port <port>]`
  - pro Station virtuelle Maschine starten:
    `jpvm [-verbose] [-host <rm-host>] [-port <rm-port>]`
    (andernfalls Verwalter über Broadcast gesucht)
  - Programm ausführen:
    `jp [-host <rm-host>] [-port <rm-port] <Class>`
  - Umgebung abbauen*:
    `jprk`

JavaParty

- Variante B (Unix):
  - Automatisches Aufsetzen der Umgebung via `jpinvite`
  - ssh für Login ohne Passwort konfigurieren
    http://linuxproblem.org/art_9.html
  - `CLASSPATH` setzen
  - Datei `~/.jp-nodefile` mit Hostnamen anlegen
  - Start mit `jpinvite [-purify] <Class>`

```
> jpinvite -purify HelloJP
vm2: Hello JavaParty!
vm2: Hello JavaParty!
vm2: Hello JavaParty!
vm0: Hello JavaParty!
vm3: Hello JavaParty!
```