Übungsbeispiel Verteiltes Programmieren mit Space Based Computing Middleware – WS 2010

Der Übungsteil der Lehrveranstaltung Verteiltes Programmieren mit Space Based Computing Middleware besteht aus zwei Beispielen, einem Terminvereinbarungssystem und einer Weihnachtsfabrik, von der Sie zwei Varianten programmieren sollen. Insgesamt sind also drei Lösungen zu präsentieren, wobei Sie mehrere Wahlmöglichkeiten bei den verwendeten Technologien haben (siehe Taskbeschreibungen). Außerdem ist noch eine Präsentation zu erstellen, die Ihre Lösungen erklärt.
Für die space-basierten Aufgabenteile können folgende Middleware-Systeme verwendet werden:

- **JavaSpaces (Task 1):** GigaSpaces, Outrigger, Blitz,… (Es dürfen nur die in der Vorlesung vorgestellten Methoden von Java-Spaces inkl. der Java-Space05 Erweiterungen verwendet werden, d.h. keine zusätzlichen Erweiterungen wie sie z.B. von GigaSpaces bereitgestellt werden.)

- **ApplicationSpace (Task 1)**: .NET

- **XVSM (Task 2.2)**: MozartSpaces (Java) oder XcoSpaces (.NET). (Optional kann für XcoSpaces auch die entsprechende High-Level-API benutzt werden.)

Achtung: Beachten Sie bitte die unterschiedlichen Deadlines bei den einzelnen Tasks. ALLE Beispiele müssen für eine positive Beurteilung rechtzeitig gelöst werden.

Die Deadline für das erste Beispiel ist beim individuellen Feedbackgesprächstermin am **06./07. Dezember 2010**.

Die Deadlines für alle anderen Tasks sind beim individuellen Abgabegesprächstermin am **17./18. Jänner 2010**.

Terminvereinbarungssystem

Es soll ein kollaboratives Terminvereinbarungssystem (ähnlich wie Doodle[1]) erstellt werden, in dem sich Peers über einen Space koordinieren. Folgende Anforderungen sollen dabei erfüllt werden:

- Peers
  - Neue Peers können sich registrieren
  - Existierende Peers können sich einloggen
  - Man kann nach registrierten Peers im System suchen (über ihren Namen).
  - Ein Peer kann gleichzeitig ein Organisator und Teilnehmer sein.
  - Jeder Peer kann sich die Events, die er organisiert, oder Events, an denen er teilnimmt, anzeigen lassen.

- Organisator
  - ist ein Peer, der Events mit Namen und mehreren Termin- und Zeitvorschlägen erstellt und die Einladungen an einige registrierte Peers schickt
  - darf den Namen, die Termine und Zeiten eines Events ändern, aber nur bevor sich einer der Peers zu dem Event angemeldet hat
  - darf neue Peers zu seinen Events zusätzlich einladen
  - darf eingeladene Peers wieder löschen, bevor sich diese zu dem Event angemeldet haben
  - darf die Events jederzeit löschen
  - darf zu seinen Events Kommentare posten
  - darf Kommentare zu seinen Events löschen (auch die von anderen Peers)
  - Nachdem sich alle Peers zu einer Einladung angemeldet haben, darf der Organisator einen fixen Termin festlegen.

- Teilnehmer
  - wählt aus den vorgeschlagenen Terminen und Zeiten eines Events (eine Checkbox pro Zeitvorschlag reicht)
  - darf seine Wahl ändern, bis ein fixer Termin existiert
  - darf Kommentare zu Events, an denen er teilnimmt, posten

- Notifications
  - Ein Teilnehmer wird über jede neue/editierte/gelöschte Eventeinladung notifiziert.
  - Weiters wird ein Teilnehmer notifiziert, sobald ein fixer Termin für ein Event festgelegt wird.
  - Ein Organisator wird notifiziert, sobald sich alle Teilnehmer zu einer seiner Events angemeldet haben.
  - Wenn ein Peer zur Zeit einer Notification offline ist, darf diese nicht verloren gehen. Der Peer bekommt alle seine versäumten Notifications, sobald er online kommt.

[1] www.doodle.com

Aufgabenstellung:

Entwickeln Sie ein GUI-Programm, welches das Terminvereinbarungssystem realisiert. Bei der Abgabe müssen Sie die Aufgabe nur auf einem Rechner (mit mehreren gleichzeitig gestarteten Peers) präsentieren. Implementieren Sie Ihre Lösung aber bitte so, dass sie auch über ein Netzwerk funktionieren würde (d.h. „localhost“ nicht hartkodieren!).

Beim Starten des Programms müssen der gewünschte Benutzername und die Netzwerkadresse des Servers angegeben werden (kein Passwort erforderlich). Die Registrierung kann automatisch bei der ersten Anmeldung erfolgen.

Achten Sie bei der Implementierung auf die transaktionale Sicherheit. Überlegen Sie sich Situationen, in denen z.B. ein Peer versucht, eine Terminwahl zu einem in der Zwischenzeit gelöschten Event zu realisieren. Ihr Programm sollte auf solche und ähnliche Situationen entsprechend reagieren.

Beachten Sie bei der Implementierung, dass die Kommentare in derselben Reihenfolge aufgelistet werden müssen, in der diese von den einzelnen Peers abgeschickt wurden.
Sie brauchen sich keine Gedanken über die Persistenz der Informationen im Space zu machen. Wenn der Space herunterfährt, ist es OK, dass der gesamte Inhalt verloren geht.

Es reicht ein einfaches, aber funktionales GUI.

Task 1:

Implementieren Sie das Terminvereinbarungssystem gemäß den oben definierten Anforderungen. Entwickeln Sie das Terminvereinbarungssystem entweder mit JavaSpaces, oder ApplicationSpace. Bei allen Varianten sollen Notifications und Transactions (sofern vorhanden) sinnvoll eingesetzt werden. Jedes Event soll durch (mindestens) ein eigenes Objekt im Space repräsentiert sein (und nicht etwa in einer Liste, die als Ganzes in den Space geschrieben wird). Die Kommunikation zwischen den Peers darf nur über den gemeinsamen Space erfolgen, der vor dem Starten der Client-Programme gestartet und evtl. initialisiert werden muss.

Deadline: **06./07. Dezember 2010 beim individuellen Feedbacktermin**

Weihnachtsfabrik

Aufgabenstellung:
Im Rahmen dieses Beispiels soll ein Koordinationsproblem möglichst effizient gelöst werden. Es handelt sich um die Simulation einer Fabrik des Christkinds. An der Herstellung sind beteiligt:

- Produktionselfen
- Montagewichtel
- Prüfzwergen
- Logistikrentiere

Teddybären werden aus mehreren Einzelteilen hergestellt. Es gibt folgende Arten von Einzelteilen:

- Kopf
- Körper
- Arm
- Bein
- Mütze

Jeder Teddybär besteht aus einem Kopf, einem Körper, 2 Armen, 2 Beinen und hat eine Mütze auf, die entweder grün oder rot ist.

Die Einzelteile werden von spezialisierten Produktionselfen erzeugt, d.h. jede Elfe erzeugt entweder Köpfe, Körper, Arme, Beine, grüne Mützen ODER rote Mützen. Produktionselfen sind Akkordarbeiterinnen. Das heißt, sie produzieren die ihnen vorgegebene Anzahl an Teilen und haben dann ihre Arbeit erledigt. Sollen noch mehr Teile produziert werden, muss man eine neue Elfe damit beauftragen. Eine Elfe ist kein Roboter und deshalb macht sie ab und zu Fehler und erstellt defekte Teile. Die Produktion jedes Teiles soll eine gewisse Zeit dauern (ein Zufallswert von 1-3 Sekunden reicht).
Einzelteile haben eine eindeutige ID und man soll zu jeder Zeit überprüfen können, welche Elfe welches Teil erzeugt hat. Für jedes Einzelteil muss außerdem gespeichert werden, ob es defekt ist oder nicht.
Zu Arbeitsbeginn wird jeder Elfe das zu produzierende Teil und die Menge mitgeteilt. Außerdem hat sie eine bestimmte Fehlerrate (in %).

Montagewichtel nähen die Teddybären zusammen, sobald alle Teile verfügbar sind. Fertige Teddybären unterscheiden sich durch ihre eindeutige ID.
Die Montagewichtel arbeiten unabhängig voneinander und dürfen sich auf keinen Fall gegenseitig behindern, d.h. ein Wichtel darf z.B. nicht den letzten Kopf für sich beanspruchen und ein anderer die letzte Mütze, wodurch beide keinen fertigen Teddybären herstellen können, obwohl eigentlich noch genug Teile für einen Teddy vorhanden wären. Wenn alle Bärenbestandteile zur Verfügung stehen, müssen die Teile zusammengebaut werden.

Die Prüfzwerge überprüfen die fertigen Teddybären. Jeder Prüfzwerg ist für eine bestimmte Art von Überprüfung ausgebildet worden. Ein Bär muss vor der Auslieferung folgende Tests bestehen: Gewichtstest und Komponentenprobe. Beim Gewichtstest müssen Sie keine Logik implementieren. Es reicht, wenn der Test immer positiv ist. Bei der Komponentenprobe wird nach defekten Teilen gesucht. Falls ein Teddybär ein oder mehrere defekte Teile hat, wird der ganze Teddy als defekt markiert. Die Tests können in verschiedenen Reihenfolgen durchgeführt werden, abhängig davon, welcher Zwerg gerade Zeit hat, allerdings nie gleichzeitig an einem Bären.
Welche Messung ein Zwerg durchführen kann, wird beim Start angegeben.

In der Logistik werden die fertigen Teddybären von Logistikrentiere ausgeliefert (= als fertig markiert), nachdem sie alle Tests bestanden haben. Defekte Teddybären werden nicht transportiert, sondern enden an einer Sammelstelle.

Es soll in jeder Abteilung (Produktion, Montage, Prüfung, Logistik) festgehalten werden, welcher Mitarbeiter die Arbeiten ausgeführt hat. Jeder Mitarbeiter wird dabei durch eine eigene ID identifiziert. Alle Beteiligten können dynamisch hinzugefügt und weggenommen werden. Jeder Mitarbeiter stellt für sich ein kleines unabhängiges Programm dar (mit eigener main-Methode) mit der Ausnahme von Produktionselfen, die von anderen Programmen erzeugt und verwendet werden können.

Task 2.1
Implementieren Sie ein GUI für eine Fabrik, mit welchem Produktionselfen erzeugt und aktiviert werden können, die Einzelteile produzieren. Jede Elfe hat eine benutzerdefinierte ID und ist für Herstellung eines spezifischen Teils zuständig. Für jede Elfe soll festgelegt werden, wie viele Stück sie produzieren soll und welche Fehlerrate sie hat.
Außerdem soll das GUI eine Informationstafel für das Christkind enthalten.

Die GUI soll also folgende Features bieten:

- Erstellung von Elfen (Benennung und Auswahl des Produktionsteils, Menge und Fehlerrate)
- Es sollen beliebig viele Produktionselfen gleichzeitig gestartet werden können. (Implementierung als Threads)
- Automatisches Auslesen der ausgelieferten Teddybären. Dabei soll die gesamte Information des Teddys angezeigt werden (Einzelteile, beteiligte Mitarbeiter, usw.).
- Automatisches Auslesen von defekten Teddybären. Die gesamte Information des Teddys sollte angezeigt werden. Es muss ersichtlich sein, welche Teile defekt sind.
- Anzeige der Anzahl aller Einzelteile im Space.

Das GUI soll nur die Produzenten starten und die aktuellen Daten zur Fabrik auf der Anzeigetafel darstellen. Die eigentliche Erstellen der Teddybären wird durch eigenständige Programme, für die kein GUI notwendig ist, durchgeführt (siehe Tasks 2.2 und 2.3).

Achten Sie darauf, dass das GUI relativ unabhängig vom Koordinationsmodell ist, damit Sie es sowohl für Ihre Space- also auch Ihre Alternativ-Implementierung verwenden können.
Das GUI muss grafisch nicht aufwändig gestaltet sein, solange die gewünschte Funktionalität vorhanden ist.

Deadline: 17./18. Jänner bei den individuellen Terminen für die Abgabegespräche

Task 2.2

Realisieren Sie mithilfe des GUIs aus Task 2.1 das oben definierte Koordinationsproblem möglichst effizient mit einer XVSM Middleware-Technologie (MozartSpaces oder XcoSpaces). Bei allen Varianten sollen Notifications und Transactions sinnvoll eingesetzt werden.

Die in Task 2.1 durch die Produktionselfen erstellten Teile sollen in den Space geschrieben werden. Anschließend soll die Lösung die Einzelteile zu Teddybären zusammensetzen, diese testen lassen und schließlich die fertigen Teddybären im Space speichern.

Jedes Einzelteil und jeder Teddybär soll durch (mindestens) ein eigenes Objekt im Space repräsentiert sein (und nicht etwa in einer Liste, die als Ganzes in den Space geschrieben wird). Jeder Montagewichtel, jeder Prüfzwerg und jedes Logistikrentier wird dabei als eigener Prozess gestartet. Diese Prozesse können als einfache Konsolenapplikationen implementiert werden, bei denen die ID als Argument übergeben wird. Die Kommunikation zwischen den Mitarbeiter darf nur über den gemeinsamen Space erfolgen, der vor dem Starten der Mitarbeiter gestartet und evtl. initialisiert werden muss. Die Anzeigetafel aus Task 2.1 erhält alle Informationen nur über den Space. Prozesse für Mitarbeiter sollen jederzeit dynamisch gestartet oder geschlossen werden können, ohne dass die Funktionalität beeinträchtigt wird.

Deadline: 17./18. Jänner bei den individuellen Terminen für die Abgabegespräche

Task 2.3

Implementieren Sie dasselbe Problem mit einer nicht Space-basierten Technologie. Sie können z.B. Java RMI, Corba, Sockets, WCF, JMS, etc. verwenden. Jeder Montagewichtel, jeder Prüfzwerg und jedes Logistikrentier wird dabei als eigener Prozess gestartet. Diese Prozesse können als einfache Konsolenapplikationen implementiert werden, bei denen die ID als Argument übergeben wird. Prozesse für Mitarbeiter sollen jederzeit dynamisch gestartet oder geschlossen werden können, ohne dass die Funktionalität beeinträchtigt wird.

Die Lösungen aus Task 2.2 und 2.3 müssen nicht kompatibel sein, d.h. Mitarbeiter des Space-basierten Programms müssen nicht mit Mitarbeitern des Alternativ-Programms interagieren können. (Wenn Sie Task 2.2 und 2.3 nicht mit derselben Programmiersprache (Java bzw. C#) implementieren, müssen Sie die GUI vermutlich noch einmal implementieren.)

Deadline: 17./18. Jänner bei den individuellen Terminen für die Abgabegespräche

Präsentation

Task 3:

Erstellen Sie eine kurze Präsentation Ihrer Lösungen aus Task 1 und 2 mit Architektur, Vorteilen, Nachteilen, Aufwandsabschätzung, etc. und vergleichen Sie dabei die verwendeten Technologien. Vergessen Sie nicht folgende Punkte zu berücksichtigen:

- Welche Koordinationsmodelle haben Sie benutzt und warum?
- Wie viele Zeilen Code umfasst jeder (Sub-)Task?
- gesamter Arbeitsaufwand in Stunden pro (Sub-)Task

Deadline: 17./18. Jänner bei den individuellen Terminen für die Abgabegespräche

Zu beachten:

Versuchen Sie alle Aufgaben möglichst effizient und einfach zu lösen. Für eine positive Beurteilung müssen alle Tasks rechtzeitig gelöst werden. Beachten Sie auch die unterschiedlichen Deadlines der Tasks.

Sie sollen ein „Build-Tool“ für die Aufgabe verwenden. Hierfür kann zwischen Maven und Ant gewählt werden. Die Maven-Dateien (pom.xml usw.) bzw. Ant-Dateien, die zum Kompilieren und Ausführen des Programms benötigt werden, müssen Sie auch abgeben. Alternativ können auch Scripts zum Starten des Programms mitgeliefert werden (wenn möglich für Windows und Linux). Außerdem muss in einem Readme-File genau dokumentiert werden, wie die einzelnen Applikationen kompiliert und gestartet werden können.

Als Programmiersprache können Sie Java und/oder C# verwenden.

Hilfreiche Dokumentation:	Bei der Abgabe:
• VO-Folien • GigaSpaces Intro • MozartSpaces Tutorial • XcoSpaces Tutorial	• Abgabe des Source Codes (kommentiert) und der restlichen Dokumente inkl. Präsentationsfolien (z.B. über USB-Stick) • Präsentation des Programms auf ihrem Rechner (Laptop) • Abgabe des Feedbackformulars (bei der Abgabe erhältlich)