Verteiltes Programmieren mit Space Based Computing Middleware – SS 2012
Übungsbeispiel 1

Der Übungsteil der Lehrveranstaltung Verteiltes Programmieren mit Space Based Computing Middleware besteht aus zwei Übungsaufgaben, von denen hier die erste beschrieben wird. Die Angabe des zweiten Beispiels wird bei der Abgabe des ersten Beispiels vorgestellt. Für die Lösung des ersten Beispiels sollte beachtet werden, dass das zweite Beispiel auf die Lösung des ersten Teils aufbauen wird, d.h. Sie müssen Ihre Implementierungen entsprechend adaptieren und erweitern. Die Beispiele sollen jeweils in einer Space-basierten und einer nicht Space-basierten Variante implementiert werden, die anschließend verglichen werden sollen.

Für die Space-basierten Aufgabenteile können folgende Middleware-Systeme verwendet werden:

- **JavaSpaces:** Outrigger, Blitz, GigaSpaces,…
- **Application Space**: .NET
- **XVSM**: MozartSpaces (Java) oder XcoSpaces (.NET). (Optional kann für XcoSpaces auch die entsprechende High-Level-API benutzt werden.)

ALLE Beispiele müssen für eine positive Beurteilung rechtzeitig gelöst werden.

Keine Plagiate! Es ist verboten, Lösungen von vorherigen Semestern zu verwenden. Wir werden dies, teilweise automatisiert, überprüfen. Sollten Teile der Übung abgeschrieben sein, wird ein negatives Zeugnis ausgestellt.

Die Deadline für das erste Beispiel ist beim individuellen ersten Abgabegesprächstermin am **10./11. Mai 2012**.

Die Deadline für das zweite Beispiel ist beim individuellen zweiten Abgabegesprächstermin am **21./22. Juni 2012**.

Beispiel 1 - Autofabrik

Deadline:
10./11. Mai bei den individuellen Terminen für die Abgabegespräche

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

- Produzenten/-innen (je einer für Karosserien, Motoren und Räder)
- Montage-Mitarbeiter/-innen
- Lackierer/-innen
- Logistik-Mitarbeiter/-innen

In der Fabrik werden Autos hergestellt. Jedes Auto besteht aus:

- 1 x Karosserie
- 1 x Motor
- 4 x Räder

Produzenten/-innen sind Akkordarbeiter/-innen. 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(n) neue(n) Arbeiter/-in damit beauftragen. 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(r) Arbeiter/-in welches Teil erzeugt hat.
Zu Arbeitsbeginn wird jedem/r Produzenten/-in die zu produzierende Menge mitgeteilt.

Um die Autos zusammenzusetzen, werden die **Montage-Mitarbeiter/-innen** benötigt. Sie nehmen 1 Karosserie, 1 Motor, und 4 Räder und fertigen daraus ein Auto.

Ein(e) **Lackierer/-in** kann eine Karosserie mit genau einer Farbe bemalen, welche beim Starten angegeben wird. Jede Karosserie muss genau einmal lackiert werden. Lackierer/-innen können an jeder Stelle der Produktionskette arbeiten, d.h. sowohl vor als auch nach der Montage. Es sollen aber fertig montierte Autos bevorzugt werden, damit diese schnell ausgeliefert werden können.

In der Logistik werden die vollständig montierten und lackierte Autos von **Logistik-Mitarbeitern/-innen** ausgeliefert (= als fertig markiert). Alle vollständigen Autos sollen dabei in FIFO-Ordnung verarbeitet werden (ältere zuerst!), damit sie nicht zu lange im Lager verstauben.

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

Allgemeines:
Betrachten Sie je drei Space- und drei Alternativ-Technologien und evaluieren Sie, welche sich für die Aufgabenstellung am besten eignet.
Dokumentieren Sie (im Rahmen von Task 3) die Vor- & Nachteile und begründen Sie Ihre Entscheidung für die von Ihnen gewählten Technologien.

Das zweite Übungsbeispiel wird auf dem ersten Beispiel aufbauen. Achten Sie deshalb auf eine gut erweiterbare Architektur.

Task 1.1 - GUI
Implementieren Sie ein GUI für eine Fabrik, mit welcher die Arbeiter/-innen erzeugt und aktiviert werden können, die Einzelteile produzieren. Für jede(n) Produzenten/-in soll festgelegt werden, wie viel Stück er/sie produzieren soll.
Außerdem soll das GUI eine Informationstafel für das Management enthalten.

Die GUI soll also folgende Features bieten:

- Erstellung von Produzenten/-innen (inkl. Mengenangabe)
- Es sollen beliebig viele Produzenten/-innen gleichzeitig gestartet werden können. (Implementierung als Threads)
- Automatisches Auslesen der ausgelieferten Autos. Dabei soll die gesamte Information über Autos angezeigt werden (Einzelteile mit IDs, Farbe, beteiligte Mitarbeiter, usw.).
- Anzeige der Anzahl aller Einzelteile im Space (aufgeschlüsselt je Typ inkl. Unterscheidung lackierter und unlackierter Karosserien).

Das GUI soll nur die Produzenten/-innen starten und die aktuellen Daten zur Fabrik auf der Anzeigetafel darstellen. Das eigentliche Erstellen der Autos wird durch eigenständige Programme, für die kein GUI notwendig ist, durchgeführt (siehe Tasks 1.2 und 1.3).

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.

Task 1.2 – Space-Implementierung

Realisieren Sie mithilfe des GUIs aus Task 1.1 das oben definierte Koordinationsproblem möglichst effizient mit einer space-basierten Middleware-Technologie (JavaSpaces, XVSM oder Application Space). Notifications und Transactions sollen sinnvoll eingesetzt werden.

Die in Task 1.1 durch die Produzenten/-innen erstellten Teile sollen in den Space geschrieben werden. Anschließend soll die Lösung aus den Einzelteilen fertig montierte und lackierte Autos erstellen, die schließlich ebenfalls im Space gespeichert werden.

Jedes Einzelteil und jedes Auto soll durch (mindestens) ein eigenes Objekt im Space repräsentiert sein (und nicht etwa in einer Liste, die als Ganzes in den Space geschrieben wird). Jede(r) Arbeiter/-in 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 Mitarbeitern/-innen darf nur über den gemeinsamen Space erfolgen, der vor dem Starten der Mitarbeiter/-innen gestartet und evtl. initialisiert werden muss. Die Anzeigetafel aus Task 1.1 erhält alle Informationen nur über den Space. Prozesse für Mitarbeiter/-innen sollen jederzeit dynamisch gestartet oder geschlossen werden können, ohne dass die Funktionalität beeinträchtigt wird.

Task 1.3 – Alternativ-Implementierung

Implementieren Sie dasselbe Problem mit einer nicht space-basierten Technologie. Sie können z.B. Java RMI, CORBA, Sockets, WCF, JMS, etc. verwenden. Jede(r) Arbeiter/-in wird dabei als eigener Prozess gestartet. Diese Prozesse können als einfache Konsolen-applikationen implementiert werden, bei denen die ID als Argument übergeben wird. Prozesse für Mitarbeiter/-innen sollen jederzeit dynamisch gestartet oder geschlossen werden können, ohne dass die Funktionalität beeinträchtigt wird.

Die Lösungen aus Task 1.2 und 1.3 müssen nicht kompatibel sein, d.h. Mitarbeiter/-innen des space-basierten Programms müssen nicht mit Mitarbeitern/-innen des Alternativ-Programms interagieren können. (Wenn Sie Task 1.2 und 1.3 nicht mit derselben Programmiersprache (Java bzw. C#) implementieren, müssen Sie das GUI vermutlich noch einmal implementieren.)

Beispiel 2 - Erweiterte Autofabrik

Deadline:
21./22. Juni 2012 bei den individuellen Terminen für die Abgabegespräche

Die Angabe für das zweite Beispiel wird bei der Abgabe des ersten Beispiels vorgestellt.

Dokumentation

Deadline:
21./22. Juni 2012 bei den individuellen Terminen für die Abgabegespräche

Task 3:

Erstellen Sie eine kurze Präsentation Ihrer Lösungen mit Architektur, Vorteilen, Nachteilen, Aufwandsabschätzung, etc. und vergleichen Sie dabei die von Ihnen evaluierten bzw. 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

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 • 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 2. Abgabe erhältlich)