Verteiltes Programmieren mit Space Based Computing Middleware – WS 2014 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 der 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.

Abgaben:

Die Deadline für das erste Beispiel ist am **09. Dezember 2014 23:55**.

Die Deadline für das zweite Beispiel ist am **11. Jänner 2015 23:55**.

Ihre Abgabe umfasst eine ZIP-Datei, inklusive aller verwendeten Sourcen, Build-Scripte und geforderten Dokumente.

Die Abgabe ist zwingend über einen Upload in TUWEL (vor dem eigentlichen Abgabegespräch) zu tätigen! Beim Abgabegespräch soll dann die Funktionsweise der Lösung demonstriert und erklärt werden.

Beispiel 1 - Feuerwerksfabrik

Deadline:
09. Dezember 2014 23:55 in TUWEL

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

- Lieferanten/-innen
- Produzenten/-innen
- Qualitätskontrolleure-/innen
- Logistiker/-innen

In der Fabrik werden Feuerwerksraketen hergestellt. Jede Rakete besteht aus:

- 1 x Holzstab
- 1 x Gehäuse mit Zünder
- ca. 130g Treibladung
- 3 x Effektladungen

Lieferanten/-innen arbeiten nach Einzelaufträgen. Das heißt, sie liefern einmalig die bestellte Anzahl an Bestandteilen ab und haben dann ihre Arbeit erledigt. Werden noch mehr Bestandteile benötigt, muss man eine(n) neue(n) Lieferanten/-in damit beauftragen. Das Abladen der einzelnen Bestandteile vom LKW des/der Lieferanten/-in in das Lager der Fabrik soll eine gewisse Zeit dauern (ein Zufallswert von 1-2 Sekunden pro Bestandteil reicht). Bestandteile haben eine eindeutige ID und man soll zu jeder Zeit überprüfen können, welche(r) Lieferant/-in welchen Bestandteil geliefert hat. Zu Arbeitsbeginn wird jedem/r Lieferanten/-in der zu liefernde Bestandteil und die gewünschte Menge mitgeteilt.

Die *Holzstäbe*, *Gehäuse mit Zünder* und *Effektladungen* werden als vorkonfektionierte Einzelteile geliefert und können direkt verwendet werden.

Effektladungen sind außerdem fehleranfällig und werden zu einem gewissen Prozentsatz defekt geliefert. Beim Starten eines/-r Lieferanten/-in soll daher auch eine Fehlerrate angegeben werden können, die angibt, wie viele Effektladungen im Schnitt defekt angeliefert werden. Defekte Ladungen werden einfach als defekt markiert.

Treibladungen werden nicht als Einzelkomponenten, sondern als Pulver in Großpackungen zu je 500g geliefert. Eine Packung zählt dabei als ein Bestandteil (mit jeweils eigener ID). Eine solche Großpackung reicht somit für die Produktion mehrerer Feuerwerksraketen.

Um aus den einzelnen Bestandteilen die Feuerwerksraketen zu fertigen, werden die **Produzenten/-innen** benötigt. Sie nehmen pro Rakete 1 Holzstab, 1 Gehäuse mit Zünder, 130g Treibladung (+/- 15g, zufällig bestimmt bei jeder Rakete) und 3 Effektladungen und stellen daraus eine Rakete her.

Dabei soll für jede Rakete festgehalten werden, aus welchen einzelnen Bestandteilen diese hergestellt wurde (IDs der Komponenten), welche Mengen an Treibladung aus welchen

Packungen verwendet wurde und auch welche Mitarbeiter/-innen an der Herstellung beteiligt waren.
Die benötigte Menge an Treibladung muss aus einer der lagernden Packungen entnommen werden. Dazu sieht ein/e Produzent/-in zuerst nach, ob eine angebrauchte Packung Treibladung im Lager steht. Falls dies der Fall ist, sollen zuerst diese Packung verwendet werden, bevor eine neue geöffnet wird. Ist nicht genug Treibladung in der gewählten Packung, muss der Rest aus der nächsten verfügbaren Packung entnommen werden. Ist keine ungeöffnete Packung verfügbar, muss eine beliebige neue Packung geöffnet werden. Während ein/e Produzent/-in eine Packung zum Füllen einer Rakete verwendet, steht diese keinem/keiner anderen Produzenten/-in zur Verfügung. Nach der Benutzung werden die benutzten Packungen wieder zurück ins Lager gestellt, leere Packungen werden verworfen.

Das Fertigen einer einzelnen Feuerwerksrakete soll eine gewisse Zeit dauern (ein Zufallswert von 1-2 Sekunden ist auch hier ausreichend), in der der/die Produzent/-in beschäftigt ist und keine andere Tätigkeit durchführen kann. Jede gefertigte Rakete bekommt eine eindeutige, fortlaufende Seriennummer (ID) und wird zur Qualitätskontrolle weitergereicht.

Qualitätskontrolleure/-innen nehmen immer eine der hergestellten Feuerwerksraketen und überprüfen die Füllmenge der Treibladung sowie die Funktion der Effektladungen. Ist eine Rakete mit weniger als 120g Treibladung befüllt oder ist mehr als eine der verbauten Effektladungen defekt (eine einzelne defekte Ladung wird akzeptiert), wird die entsprechende Rakete als defekt markiert. Alle überprüften Raketen werden anschließend zur Logistikabteilung weitergeleitet.

In der Logistik werden die fertigen und überprüften Feuerwerksraketen von **Logistik-Mitarbeitern/-innen** ausgeliefert (d.h. als abholbereit markiert). Die Raketen sollen dabei in Packungen zu je 5 Stück verpackt werden. Um die potentiell gefährlichen Raketen nicht zu lange lagern zu müssen, sollen diese in FIFO-Reihenfolge (ältere zuerst) abgearbeitet werden. Von der Qualitätskontrolle als defekt markierte Raketen sollen aussortiert und entsorgt werden (z.B. in einen Abfallcontainer verschoben werden).

Es soll in jeder Abteilung (Anlieferung, Produktion, Qualitätskontrolle, 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 Lieferanten/-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 der die Lieferanten/-innen, die Bestandteile anliefern, erzeugt und aktiviert werden können. Für jede(n) Lieferanten/-in soll festgelegt werden, welcher Bestandteil und welche Menge er/sie anliefert. Außerdem soll das GUI eine Informationstafel für das Management enthalten.

Die GUI soll folgende Features bieten:

- Erstellung von Lieferanten/-innen (mit Angabe des Bestandteils und der Menge)
- Es sollen beliebig viele Lieferanten/-innen parallel gestartet werden können. (z.B. Implementierung als Threads)
- Anzeige der Anzahl aller verfügbaren Bestandteile im Space (aufgeschlüsselt je Typ, inkl. angebrochener Treibladungs-Packungen und deren Inhalt).
- Anzeige aller hergestellten Feuerwerksraketen. Dabei soll die gesamte Information über die Rakete ersichtlich sein: Seriennummer (ID), Bestandteile-IDs, IDs aller beteiligten Mitarbeiter/-innen, Füllmengen, Status der Effektladungen (ok/defekt), Ergebnis der Qualitätskontrolle (offen, ok, defekt), etc.
- Anzeige der Daten aller ausgelieferten Raketen (gruppiert nach Packung).
- Auch die Daten aller entsorgten Raketen sollen ersichtlich sein.

Das GUI soll nur die Lieferanten/-innen starten und die aktuellen Daten zur Fabrik auf der Anzeigetafel darstellen. Die eigentliche Produktion der Raketen wird durch eigenständige Programme (Produzent/-innen, Qualitätskontrolleur/-innen, Logistik-Mitarbeiter/-innen), 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). Asynchrone Kommunikation (z.B. über Notifications) und Transaktionen sollen sinnvoll eingesetzt werden.

Die in Task 1.1 durch die Lieferanten/-innen angelieferten Bestandteile sollen in den Space geschrieben werden. Anschließend soll die Lösung aus den einzelnen Komponenten fertige Feuerwerksraketen produzieren, die schließlich ebenfalls im Space gespeichert werden.

Jede Komponente und jede Rakete 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. Bei der

Implementierung soll auf eine möglichst hohe Concurrency geachtet werden, um den Ablauf in der Fabrik und die Zusammenarbeit der beteiligten Mitarbeiter/-innen nicht unnötig zu behindern.

Task 1.3 – Alternativ-Implementierung

Implementieren Sie dasselbe Problem mit einer nicht space-basierten Technologie ihrer Wahl. 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. Auch bei der Alternativimplementierung soll auf eine möglichst hohe Concurrency geachtet werden, um den Ablauf in der Fabrik und die Zusammenarbeit der beteiligten Mitarbeiter/-innen nicht unnötig zu behindern.

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 allerdings 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 Feuerwerksfabrik

Deadline:
11. Jänner 2015 23:55 in TUWEL

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

Dokumentation

Deadline:
11. Jänner 2015 23:55 in TUWEL

Task 3:

Erstellen Sie eine kurze Präsentation (ca. 10 Folien) 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.

Achten Sie darauf, dass beide Gruppenmitglieder sich in gleichen Teilen an beiden Implementierungen (Space-basiert, Alternativ) der Übungsaufgabe beteiligen. Die Kenntnis beider Teile der Übungsaufgabe wird bei den Abgabegesprächen überprüft werden.

Sie sollen ein „Build-Tool“ für die Aufgabe verwenden, um die Abgaben leichter starten und kontrollieren zu können. Hierfür kann bei Java 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. Für den nicht space-basierten Teil ist es Ihnen freigestellt, welche Sprache/Technologie sie verwenden.

Hilfreiche Dokumentation:	Bei der Abgabe:
• VO-Folien • MozartSpaces Tutorial • XcoSpaces Tutorial	• Abgabe des Beispiels zum angegebenen Termin im TUWEL • Präsentation des Programms auf ihrem Rechner (Laptop) • Ausfüllen des Feedbackformulars bei der 2. Abgabe