

Aufgabe 1: Verhaltensmodellierung mittels Sequenzdiagramm

Wiederholen Sie das Kapitel aus der Vorlesung, das sich mit Sequenzdiagrammen beschäftigt.

- Welche 4 Arten von Interaktionsdiagrammen gibt es? Beschreiben Sie diese kurz. Wofür werden Interaktionsdiagramme eingesetzt?
- Wie ist ein Sequenzdiagramm prinzipiell aufgebaut? Welche Elemente kann es enthalten?
- Was ist eine Zustandsinvariante im Kontext des Sequenzdiagramms? Wie können Zeiteinschränkungen angegeben werden?
- Was ist ein aktives Objekt, was ist ein passives Objekt? Wie unterscheiden sich diese?

Aufgabe 2: Verhaltensmodellierung mittels Sequenzdiagramm

Wiederholen Sie das Kapitel aus der Vorlesung, das sich mit Sequenzdiagrammen beschäftigt.

- Beschreiben Sie die Unterschiede zwischen synchronen und asynchronen Nachrichten.
- Welche Arten von Verzweigungen und Schleifen können in Sequenzdiagrammen auftreten? Beschreiben Sie die entsprechenden Operatoren.
- Welche Operatoren stehen im Sequenzdiagramm zur Verfügung, um parallele Abläufe zu realisieren bzw. um Ordnungen im Ablauf festzulegen?
- Erklären Sie die kombinierten Fragmente aus der Gruppe "Filterungen und Zusicherungen".

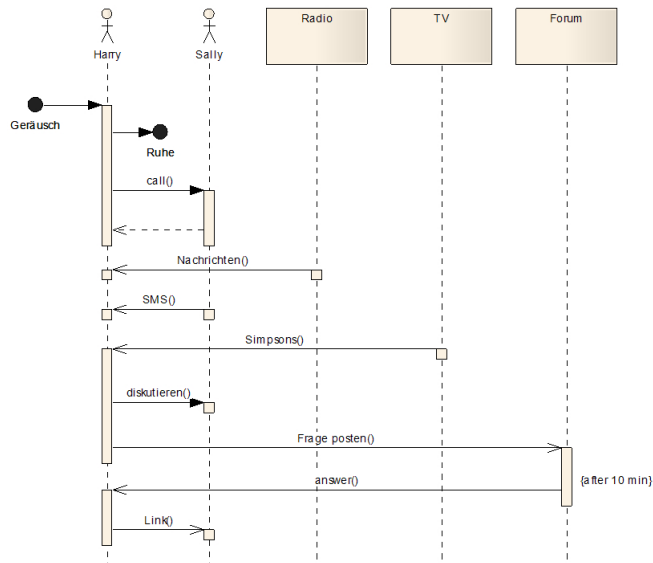
Aufgabe 3: Kommunikation im Sequenzdiagramm

- Synchrone/Asynchrone Kommunikation

Beschreiben Sie die im folgenden Diagramm vorkommenden Kommunikationsabläufe mittels Sequenzdiagramm.

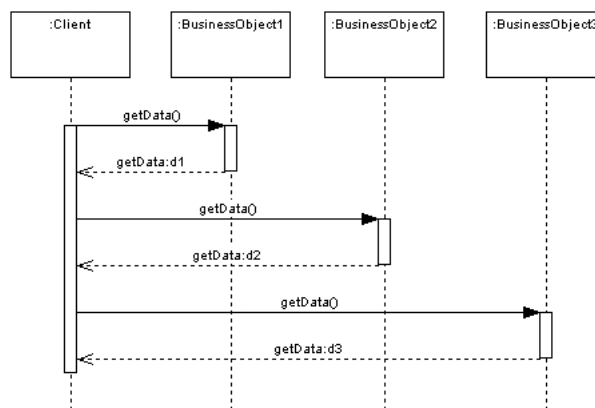
Harry wird am Morgen von einem Geräusch unbekannter Herkunft geweckt. Ein lauter "Ruhe"-Schrei seinerseits ist sinnlos, nun ist er wach. Da geteiltes Leid halbes Leid ist, ruft er seine Freundin Sally an und erzählt ihr davon. Sie bedauert ihn. Danach hört er sich im Radio die Nachrichten an. Als diese vorbei sind, sendet ihm Sally eine SMS, dass im Fernsehen eine neue Folge seiner Lieblingsserie "die Simpsons" läuft. Diese sieht sich Harry natürlich an. Nun möchte Harry seine Hausaufgaben machen. Er schaltet den Computer ein und sieht, dass Sally sich im Chat befindet. Natürlich muss er noch schnell die Geschehnisse aus den Simpsons mit Sally diskutieren. Bei der Hausübung steht Harry nun vor einem unüberwindbaren Problem. Er postet dieses im Forum. Nach 10 Minuten wird er per Mail informiert, dass bereits eine Antwort gepostet wurde. Er schickt den Link per Mail an Sally, weil auch sie die selbe Hausübung bearbeiten muss.

Achten Sie bei diesem Beispiel besonders darauf, ob die beschriebenen Kommunikationsabläufe synchron oder asynchron sind.



- Transfer-Object-Assembler-Pattern

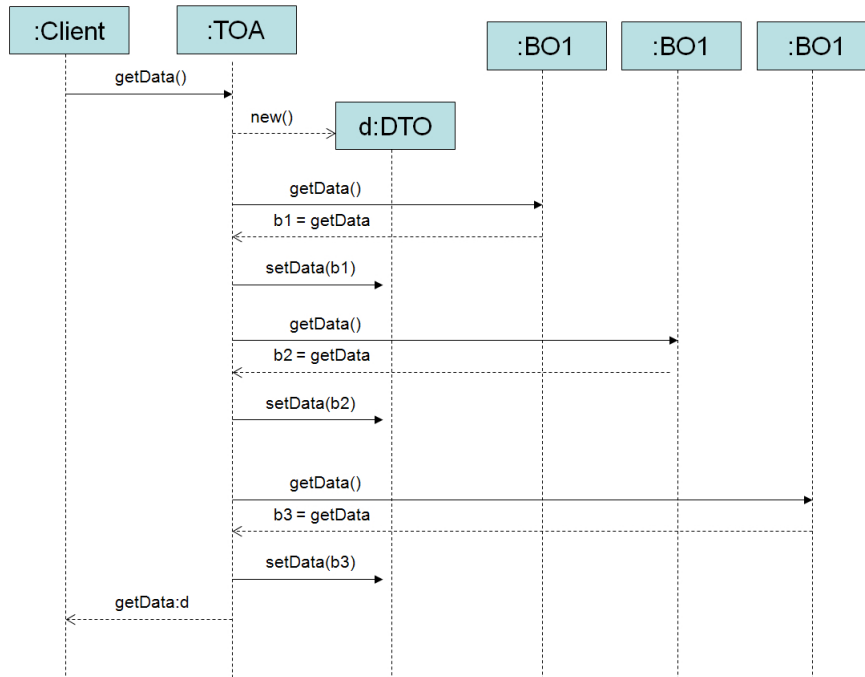
Ein Client in einer verteilten Umgebung benötigt diverse Informationen von 3 Geschäftsobjekten (BusinessObjects). Dies kann auf folgende Weise realisiert werden:



Bei diesem Lösungsansatz benötigt der Client Wissen über die Geschäftsobjekte, um an die notwendigen Daten zu gelangen. So ist der Client stark an die Geschäftsobjekte gekoppelt, was im Allgemeinen nicht wünschenswert ist. Um diese Abhängigkeiten aufzulösen, kann das Transfer-Object-Assembler-Pattern eingesetzt werden. Hier werden die Daten aus mehreren Geschäftsobjekten zu einem Transferobjekt durch einen Assembler zusammengesetzt, welches an den Client übergeben wird. So erhält dieser ausschließlich die von ihm benötigten Daten in einer gekapselten Form. Konkret wird das Pattern auf folgende Weise realisiert:

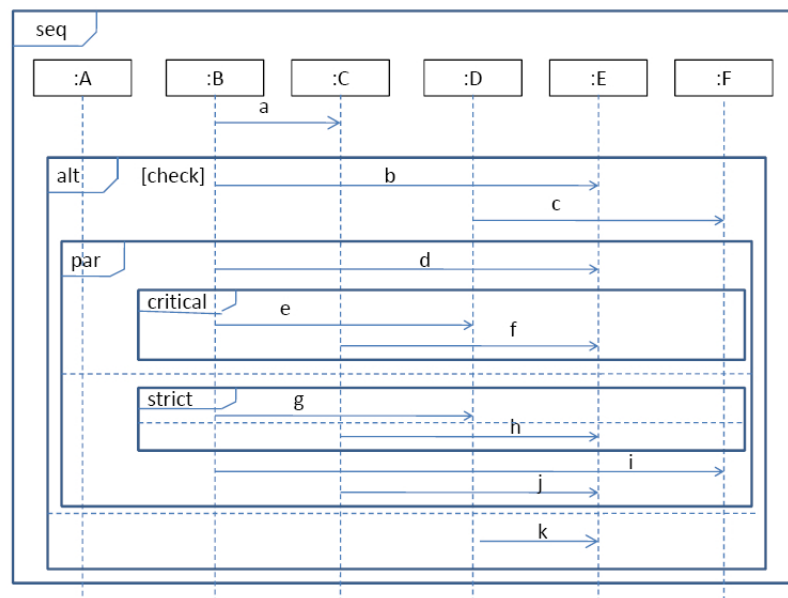
Der Client fordert mittels `getData()` die benötigten Informationen vom `TransferObjectAssembler` an. Dieser erzeugt zunächst ein `DataTransferObject` und befüllt dieses mit den Daten von 3 unterschiedlichen `BusinessObjects`. Die Daten eines `BusinessObjects` können ebenfalls mittels `getData()` abgefragt werden. Das `DataTransferObject` bietet (ev. überladene) `addData`-Methoden, die das Eintragen der Daten erlauben. Letztendlich gibt der `TransferObjectAssembler` das `DataTransferObject` an den Client zurück.

Modellieren Sie das Transfer-Object-Assembler-Pattern mittels UML2-Sequenzdiagramm.



Aufgabe 4: Kombinierte Fragmente

- Geben Sie alle möglichen Ereignisfolgen des folgenden Diagramms an.

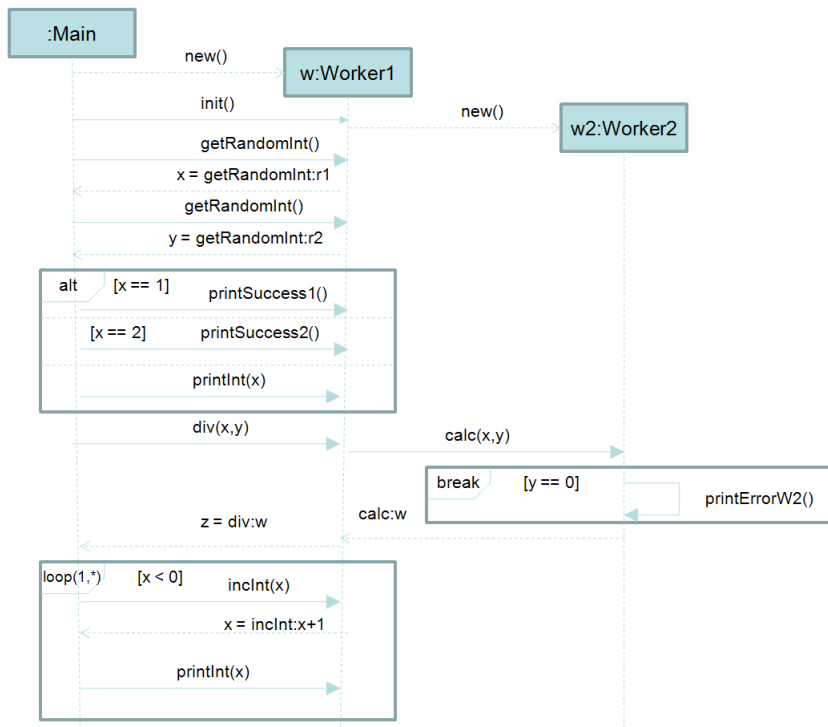


- minimaler Trace: $a \rightarrow k$ bzw. $k \rightarrow a$ vorausgesetzt check hat den Wert false. Ansonsten:
- durch den par-Operator ergeben sich folgende Ereignisfolgen:
 - Operand: $d \rightarrow e \rightarrow f$
 $d \rightarrow f \rightarrow e$
 - Operand: $g \rightarrow h \rightarrow i \rightarrow j$
 $g \rightarrow i \rightarrow h \rightarrow j$
 $g \rightarrow h \rightarrow j \rightarrow i$

- Die Traces dieser beiden Operanden können beliebig kombiniert werden, wobei zwischen e und f (egal ob $e \rightarrow f$ oder $f \rightarrow e$) keine andere Nachricht kommen darf.
- Weiters gilt: a muss vor b sein; c kann vor b sein, c kann auch vor a sein. Diese Traces werden den Traces, die aus dem par-Fragment resultieren, vorangestellt. In den Traces, in denen c unmittelbar auf d folgt, können c und d vertauscht werden.
- Darstellung von Programmabläufen mittels Sequenzdiagramm

Stellen Sie die Abläufe von folgendem Programm mittels Sequenzdiagramm dar. Modellieren Sie auch allfällige Antwortnachrichten. Die Deklaration der Variablen muss nicht angegeben werden.

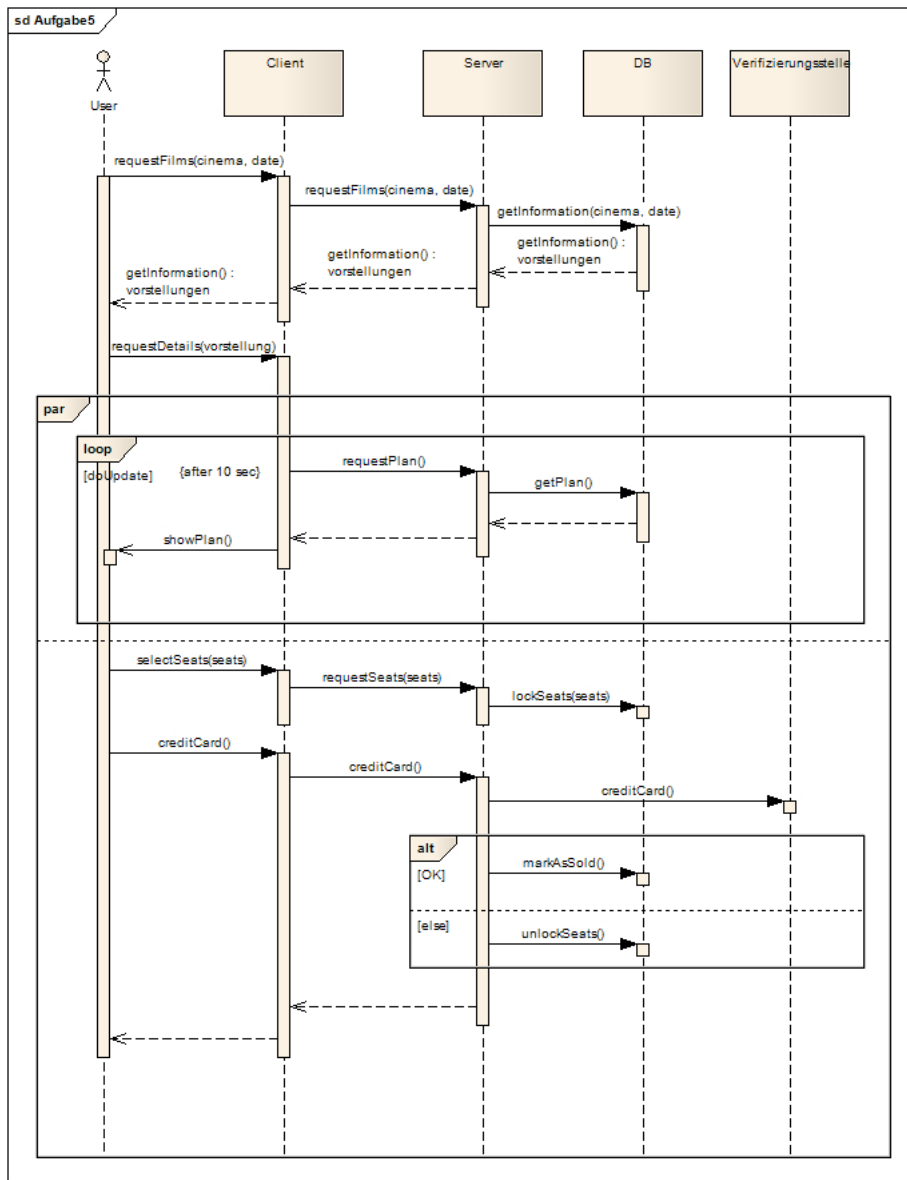
<pre> class Main { int x, y, z; Worker1 w = new Worker1(); w.init(); x = w.getRandomInt(); y = w.getRandomInt(); if (x == 1) { w.printSuccess1(); } elseif (x == 2) { w.printSuccess2(); } else { printInt(x); } z = w.div(x,y); do { x = w.incInt(x); printInt(x); } while (x < 0); } </pre>	<pre> class Worker1 { Worker2 w2; // ... void init() { w2 = new Worker2() } void printInt() { ... } int getRandomInt() { ... } void printError() { ... } void printSuccess1() { ... } void printSuccess2() { ... } int incInt(int i) { ... } int div(int a, int b) { return w2.calc(a, b); } } class Worker2 { // ... void printErrorW2 () { ... } int calc (int a, int b) { if (b == 0) { printErrorW2(); // exit beendet das // gesamte Programm exit; } return a / b; } } </pre>
---	--



Aufgabe 5: Kauf von Kinotickets

Erstellen Sie entsprechend folgender Spezifikation ein Sequenzdiagramm, welches das Kaufen eines Kinotickets in einem (sehr einfachen) Kinokartenreservierungssystem ermöglicht.

Der Benutzer fordert über seinen Client (Browser) eine Übersicht der Filme an, die in einem bestimmten Kino an einem bestimmten Tag gezeigt werden. Der Client leitet diese Anfrage an den Server des Kinobetreibers weiter, der sich wiederum die gewünschten Informationen aus der Datenbank holt. Der Benutzer kann nun Detailinformationen für eine bestimmte Vorstellung abrufen. Der Client fordert vom Server den Plan des Kinosaals an, in dem die verfügbaren Sitzplätze eingezeichnet sind. Die notwendigen Informationen stammen aus der Datenbank. Die graphische Darstellung für den Benutzer erfolgt durch den Client. Da sich die Belegung eines Saales jederzeit ändern kann, ruft der Client nun alle 10 Sekunden die aktuellen Belegungsinformationen ab und stellt diese für den Benutzer dar. Dies wird solange wiederholt, bis die Variable "doUpdate" auf "false" gesetzt wird. Gleichzeitig kann der Benutzer Sitzplätze für den angezeigten Kinosaal auswählen. Auch hier sendet der Client wieder eine Anfrage an den Server, der die entsprechenden Plätze in der Datenbank sperrt, bis der Kaufvorgang abgeschlossen ist (mögliche Fehlerfälle brauchen Sie nicht zu modellieren). Der Benutzer wird informiert, dass die Tickets für ihn reserviert sind, und dieser leitet nun den Zahlvorgang ein, indem er seine Kreditkartendaten an den Server übermittelt. Dieser übermittelt die Daten an die Verifizierungsstelle. Sind die Daten in Ordnung, werden die Tickets in der Datenbank als verkauft markiert und stehen an der Kinokassa zum Abholen bereit, ansonsten werden die Tickets wieder für den Verkauf freigegeben. In beiden Fällen wird der User informiert, dass der Kaufvorgang erfolgreich/nicht-erfolgreich war.



Aufgabe 6: Autocompletion mittels Ajax

Traditionell findet die Kommunikation zwischen Server und Client im Web auf synchrone Weise statt, d.h. der Client sendet eine Anfrage des Benutzers an den Server, wartet auf die Antwort, und stellt diese dann für den Benutzer dar, indem eine neue Webseite im Browserfenster aufgebaut wird. Dies hat den Nachteil, dass der Client blockiert, solange er keine Antwort vom Server erhalten hat. In dieser Zeit kann der Benutzer nicht weiterarbeiten.

Um dem Benutzer ein Weiterarbeiten zu ermöglichen, während der Client (Webbrowser) eine Anfrage an den Server stellt, wurde Ajax (Asynchronous JavaScript and XML) entwickelt. Ajax erlaubt das Nachladen von Daten bei Bedarf sowie ein kontextgesteuertes Update der angezeigten Webseite, ohne diese komplett neu aufzubauen. Die Realisierung auf der Client-Seite erfolgt mittels JavaScript.

Im Folgenden soll mittels Sequenzdiagramm die notwendige Kommunikation für die Realisierung einer (vereinfachten) Autocompletionfunktion im Browser dargestellt werden. Hierbei gibt der User einige Buchstaben in ein Textfeld ein und ihm werden Vorschläge zur Vervollständigung seines Textes geboten.

Durch die Eingabe eines Zeichens im Textfeld startet der User den Autocompletionprozess. Ein Timerobjekt wird erzeugt. Dieses wird benötigt, damit nur dann eine Anfrage an den Server geschickt wird, wenn der User 2 Sekunden keine Eingabe getätigt hat. Hierzu überprüft das Timerobjekt alle 2 Sekunden, ob die Bedingung "check" erfüllt ist. Ist dies der Fall wird über das XMLHttpRequest-Objekt, das für die

Kommunikation mit dem Server zuständig ist, eine Liste der möglichen Wörter mit einer asynchronen Nachricht angefordert, welche die eingegebenen Zeichen vervollständigen. Der Server schickt die Liste an das XMLHttpRequest-Objekt zurück, welches diese an das Textfeld weiterleitet, um die Vorschläge anzuzeigen. Gleichzeitig kann der Benutzer weitere Zeichen im Textfeld eingeben. Bei der Eingabe eines Zeichens wird das Timer-Objekt wieder zurückgesetzt, was sich dann natürlich auf die Auswertung der Bedingung “check” auswirkt.

