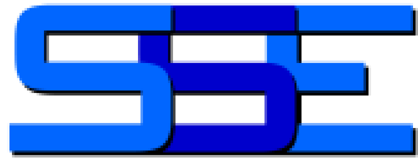


Praktikum/Teamprojekt WS07/08

Wintersemester 2007/2008

Software Systems Engineering
Fachbereich Mathematik / Informatik
Technische Universität Braunschweig

http://www.sse-tubs.de/

Übersicht

- (Folien werden ins Netz gestellt)

- Praktikum (4 SWS, 4 Leistungspunkte) – auch für Dipl.-Studiengang
- Teamprojekt (4 SWS, 6 Leistungspunkte)

- Teamprojekt erhält erweiterte Aufgabestellung

- 4 Aufgaben:
  - Workflowmodellierung mit Aktivitätsdiagrammen (Dirk Reiss)
  - Funktionsnetze: Konsistenzprüfung von Sichten mit dem Gesamtsystem (Hans Grönniger & Holger Krahn)
  - Funktionsnetze: Simulation des Funktionsnetzes (Hans Grönniger & Holger Krahn)
  - Sequenzdiagramme (Martin Schindler)

- Gesamtorganistation: Steven Völkel
- Erster Ansprechpartner: Holger Rendel h.rendel@tu-bs.de

Übersicht

- Teamgröße: 3+ Personen
- Zuordnung:
  - Nach Interesse
  - Es müssen sich 3 Personen finden
  - Abhängigkeiten müssen erfüllt sein
  - Bei > 4 Personen: Einigung, notfalls Losen
- Abhängigkeiten:
  - AD
  - FN Konsistenz
  - FN Simulation requires FN Konsistenz
  - SD requires FN Simulation

MontiCore

- Am Institut entwickeltes Framework zur Definition und Verarbeitung domänenspezifischer Sprachen
- Eingabe: Grammatik
- Ausgabe: Parser, Lexer, AST-Klassen, Editoren…
- Ziel dieses Praktikums: Codegenerierung
- Online-Service
- www.monticore.de (Papers on MontiCore, Getting started, Install Howto etc.)
- Wiki: http://s01.sse.cs.tu-bs.de/prak07
- Mailingliste: TOBEDONE
- Beispiel via CVS:
  - Host: cvsstud.sse.cs.tu-bs.de
  - Path: /cvs/Prak0708
  - User: TOBEDONE
  - Module: Automaton, Templates, 1 Modul je Gruppe

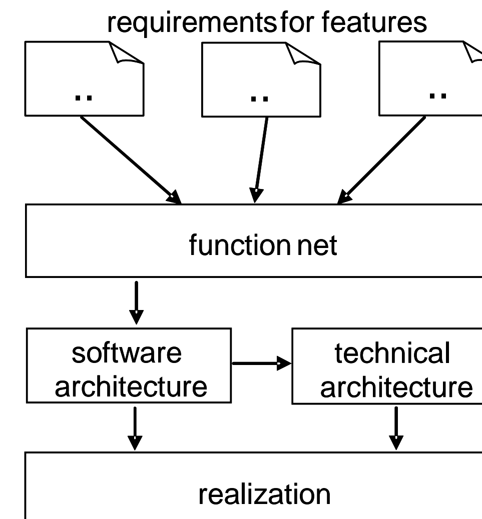
Hinweise

- Eclipse 3.2.0 (NICHT: Eclipse 3.3)
  - http://s01.sse.cs.tu-bs.de/eclipse/eclipse-SDK-3.2-win32.zip
- Java 5 (NICHT: 6)
- Teilweise Internetverbindung nötig
- Jedes Projekt hat eigenes Repository (Liste mit Mailadressen)
- Für Abgaben stehen Templates zur Verfügung (CVS)

Aufgabenbeschreibung Funktionsnetze

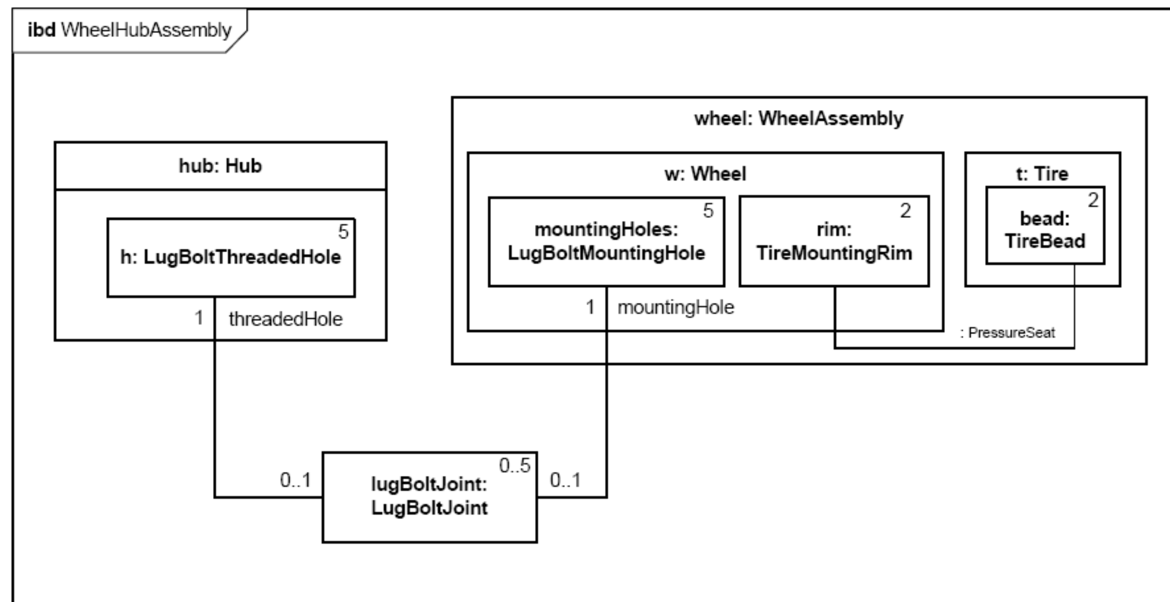
- Zwei eigenständige Aufgaben
  - Konsistenzprüfung von Sichten mit dem Gesamtsystem
  - Simulation des Funktionsnetzes

- für beide Aufgaben: Hintergrundinfos Funktionsnetzmodellierung
  - Entwicklung von automotive Systems extrem komplex
  - Aufteilung der Aufgaben auf verschiedene Entwicklungsphasen
  - Logische Architektur zeigt Funktionen und Kommunikationsverbindungen ohne Details einer technischen Realisierung



Aufgabenbeschreibung Funktionsnetze

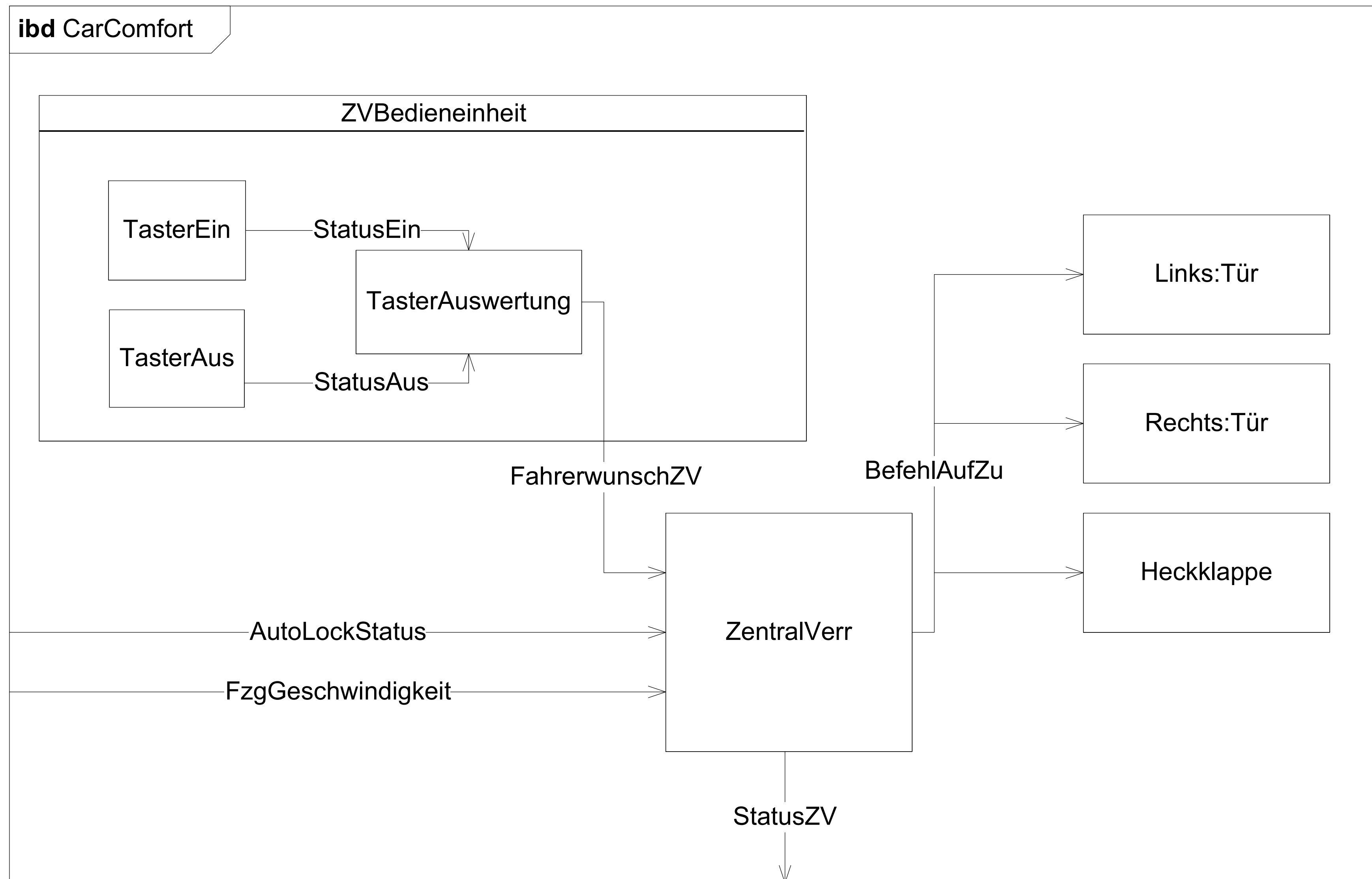
- Modellierung der Logischen Architektur durch „Funktionsnetze“
  - Funktionen, die über Konnektoren verbunden sind und zeitlich veränderliche Werte (Signale) austauschen
  - Lassen sich mittels angepassten SysML- (Internen) Blockdiagrammen gut beschreiben (Block = Funktion)
  - SysML: OMG Standard / UML-basiert



Aufgabenbeschreibung Funktionsnetze

- SysML Blockdiagramm geeignet, weil
  - benutzt Begriffe des Systems Engineering
  - keine strikte Zwei-Ebenen-Hierarchie vorgeschrieben
  - Kommunikation kann über Hierarchiegrenzen gehen
- Anpassungen für die Funktionsnetzmodellierung
  - keine Multiplizitäten (Funktionsnetz beschreibt genau und das statische System)
  - nur gerichtete Konnektoren (Signalflussrichtung erkennbar)
  - „Instanzierbarkeit“ und damit Wiederverwendbarkeit von Blöcken
  - spezielle Stereotypen, um Fahrzeugumgebung zu beschreiben
- Beispiel...

Aufgabenbeschreibung Funktionsnetze



1. Sprachbeispiel textuell

```
package a;
ibd CarComfort {
  ZVBedieneinheit {
    TasterEin;
    TasterAus;
    TasterAuswertung;

    TasterEin -> TasterAuswertung : StatusEin;
    TasterAus -> TasterAuswertung : StatusAus;
   }

  ZentralVerr;

  TasterAuswertung -> ZentralVerr : FahrerwunschZV;

  b.Tür links;
  b.Tür rechts;
  Heckklappe;

  extern -> ZentralVerr : AutoLockStatus;
  extern -> ZentralVerr : FzgGeschwindigkeit;

  ZentralVerr -> links   : BefehlAufZu;
  ZentralVerr -> rechts : BefehlAufZu;
  ZentralVerr -> Heckklappe : BefehlAufZu;

  ZentralVerr -> extern : StatusZV;
}
```

```
package b;
ibd Tür {

  Türkontakt;
  StrgSchloss;

  Türkontakt -> StrgSchloss : StatusTK;
  extern -> StrgSchloss : BefehlAufZu;
}
```

2. Sprachbeispiel textuell

```
package a;
<<view>> ibd CarComfortAutoLock {

 <<ext>> ZentraleBedieneinheit;
 <<ext>> FzgStatus;

 ZentralVerr;

 StrgSchloss links;
 <<env>> AktuatorSchlossLinks;
 StrgSchloss rechts;
 <<env>> AktuatorSchlossRechts;

 ZentraleBedieneinheit -> ZentralVerr : AutoLockStatus;
 FzgStatus -> ZentralVerr : FzgGeschwindigkeit;

 ZentralVerr -> links   : BefehlAufZu;
 ZentralVerr -> rechts : BefehlAufZu;

 links   -> AktuatorSchlossLinks   : <<E>> AufZu x;
 rechts -> AktuatorSchlossRechts : <<E>> AufZu y;
}
```

Konsistenzprüfung

- Sprachdefinition für Blockdiagramme anhand der gesehenen Beispiele
  - (in der schriftlichen Aufgabenstellungen stehen weitere Hinweise, wie z.B. imports, package-Definitionen unterstützt werden sollen)
- Zusätzlich zu den Blockdiagrammen:
  - Abwandlung der Sprache entwickeln, die es erlaubt, unvollständige Sichten auf ein Gesamtsystem zu beschreiben, die auch Elemente der Umwelt enthalten kann
  - Ziel ist jedoch eine integrierte Sprachdefinition
- Anschließend ist ein Softwarewerkzeug zu entwickeln, das es erlaubt,
  - die Konsistenz einer Sicht zu einem vollständigen Funktionsnetz zu prüfen.
  - Inkonsistenzen zu entdecken und Veränderungen nach Nutzerauswahl automatisch durchzuführen

Simulation

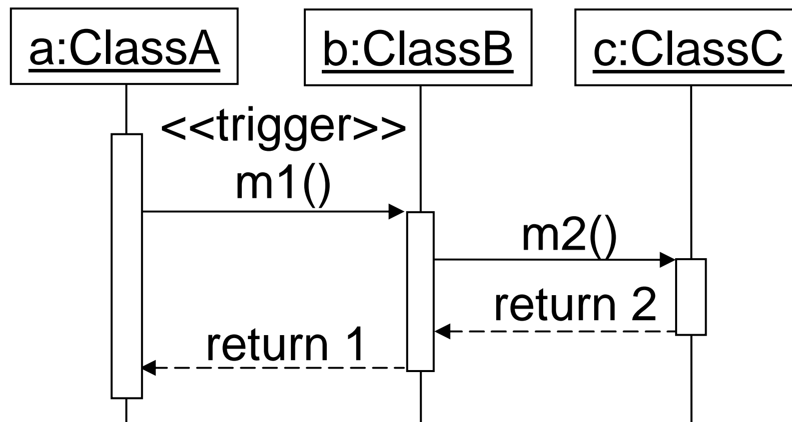
- Sprachdefinition für Blockdiagramme anhand der gesehenen Beispiele
  - (in der schriftlichen Aufgabenstellungen stehen weitere Hinweise, wie z.B. imports, package-Definitionen unterstützt werden sollen)
- Funktionsnetze lassen sich simulieren
  - einzelnen Funktionen in Java programmieren
  - wie im Funktionsnetz vorgegeben miteinander verschalten
  - Simulationsumgebung soll automatisch, schrittweise und durch den Nutzer unterbrechbar ausgeführt werden können
  - Zusätzlich soll die Simulationsumgebung „Traces“ des Systemablaufs erzeugen können, die nachträglich eine Beurteilung durch den Benutzer erlauben
  - Später Darstellung als Sequenzdiagramm (Absprachen erforderlich!)
  - Hinweise zur erwarteten Codegenerierung später anhand eines möglichen Generierungsergebnisses

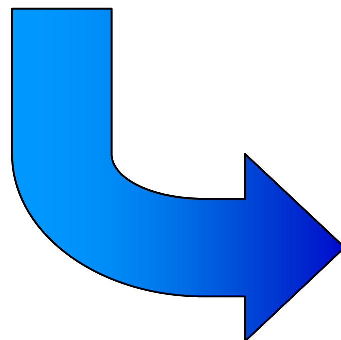
Sequenzdiagramme: Aufgabe

- Modellierung/Darstellung:
  - von exemplarischen Systemabläufen
  - anhand von Interaktionen zwischen Objekten
  - unter Berücksichtigung der zeitlichen Reihenfolge

- Ziele der Praktikumsaufgabe:
  - Entwicklung einer textuellen Darstellung
  - Focus: Modellierung von Traces innerhalb von Funktionsnetzen
  - Konsistenzprüfung von Traces
  - Transformation in einen ausführbaren Testfall
  - Demonstration des Ansatzes anhand eines Beispiels

Sequenzdiagramme: Beispiel



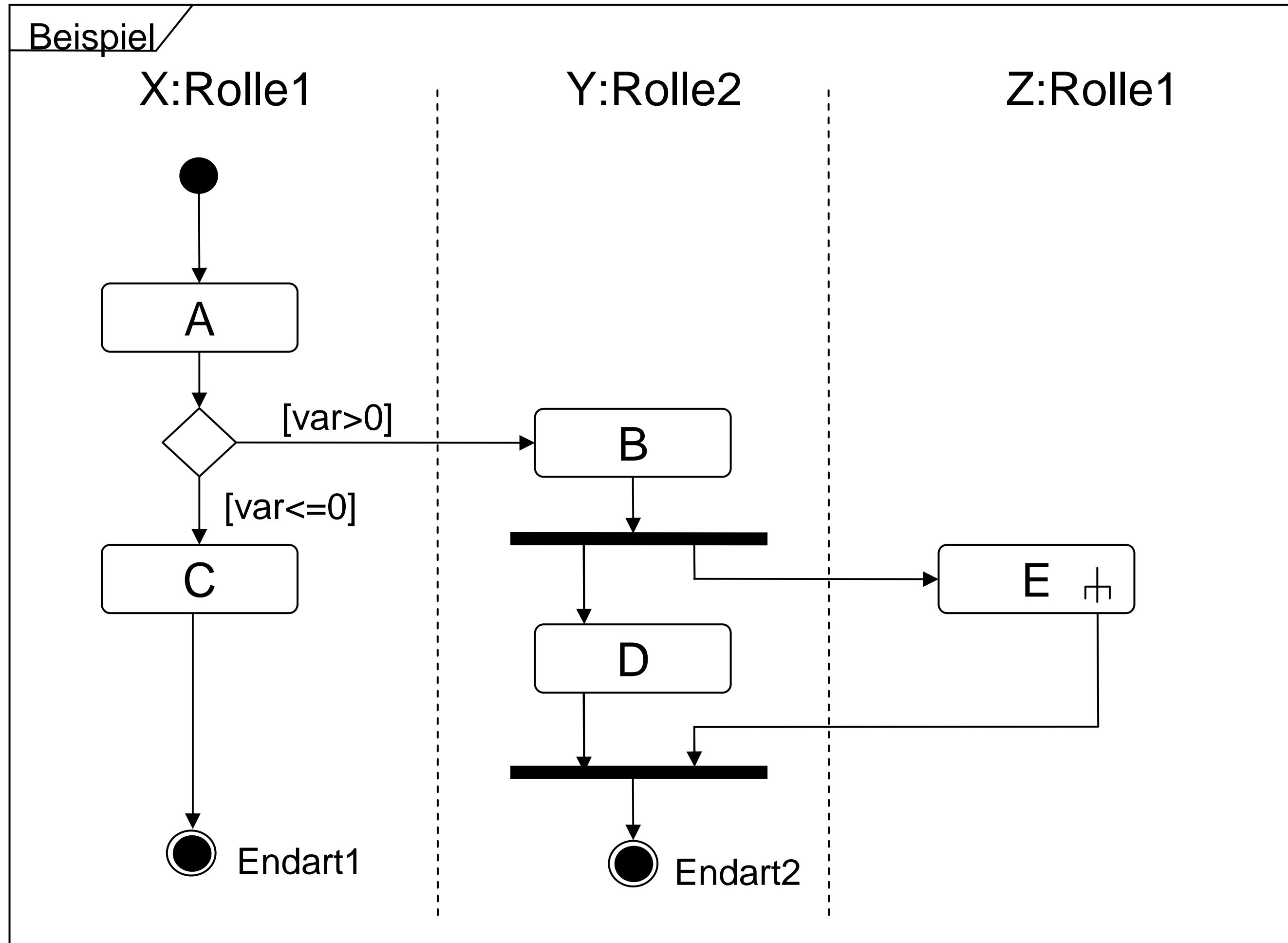


```
public void testExample(){
    Trace t1 = new Trace(a, "m1", b);
    Trace t2 = t1.getSubtrace(b, "m2", c));
    assertNotNull(t2);
    assertEquals(2, t2.getReturn());
    assertEquals(1, t1.getReturn());
}
```

Aufgabenbeschreibung Aktivitätsdiagramme

- Aktivitätsdiagramme zur Modellierung von Arbeitsabläufen
- Bestehend aus:
  - Start- und Endknoten
  - Aktivitäten
  - Verzweigungen
  - Parallelen Ausführungen
  - Partitionen
- Im Kontext dieser Aufgabe: Betrachtung von Aktivitäten als abzuarbeitende Aufgaben

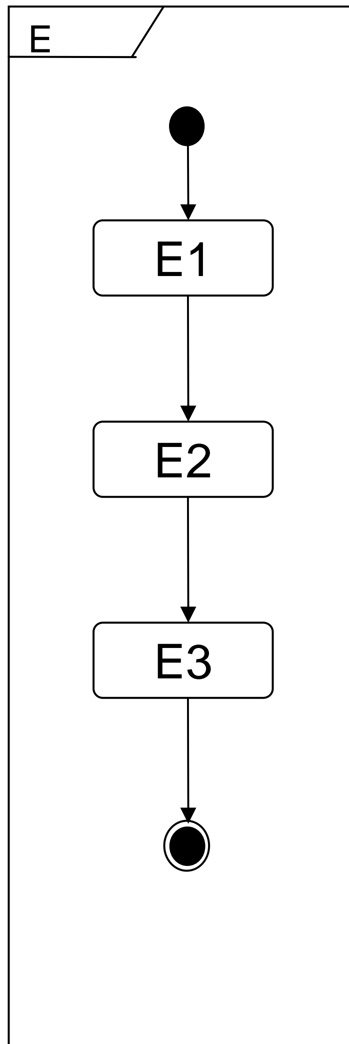
Beispiel



Textuelle Beschreibung

```
activitydiagram Beispiel {

   X:Rolle1 {
         activity A;
         activity C;
   }

   Y:Rolle2 {
         activity B;
         activity D;
   }

   Z:Rolle1 {
         activity E;
   }

   flow {
         initial -> A;
         A -> B : [var>0] || C : [var<=0];
         B -> E && D;
         E && D -> final<Endart2>
         C -> final<Endart1>
   }
}
```

Beispiel und textuelle Beschreibung




```
activitydiagram E {
        activity E1;
        activity E2;
        activity E3;

        flow {
                initial -> E1;
                E1 -> E2;
                E2 -> E3;
                E3 -> final<Endart3>;
        }
}
```

Erster Teil der Aufgabe

- Vertraut machen mit Monticore (!!!)
- Erstellen einer Grammatik, die die Beispielsprache akzeptiert
- Durchlaufen von Akzeptanztests
- Erstellen eines PrettyPrinters

- Weiterhin:
  - Aufstellen und Implementieren von Kontextbedingungen
  - Codegenerierung in ein Web-System zur Abarbeitung von Aufgaben

Zeitplan

- 14 Wochen Semesterlänge, erstes Treffen: 24.10., Ende LVA 09.02.

- initiale Grammatik im MC-Format zum Review (07.11) (kein Dokument) - Woche 2
- MC-Grammatik zum Review + Akzeptanztests + PrettyPrinter (21.11) - Woche 4
- Liste Kontextbedingungen + erste Implementierung (28.11.) - Woche 5
- Implementierung Kontextbedingungen + Testfälle (05.12.) - Woche 6

- Codegenerierung (16.01.) - Woche 11
- Beispiel (Zusatzabgabe bei Teamprojekt) (30.01.) - Woche 13
- Vorstellung: 06.02. - Woche 14

Weiteres

- Genaue Aufgabenstellungen im CVS
- Abgaben via Mail & CVS
  - Email an Betreuer am Abgabetag
    - Wo ist der Code/Dokumente im CVS
- Treffen:
  - Jeweils 1 Woche nach offiziellen Abgaben (vorige Folie, schwarz)
  - Ca. 30 Minuten pro Gruppe mit SV, Betreuer, HR & alle Gruppenteilnehmern
  - Aber auch zwischenzeitliche Treffen möglich
    - HR bzw. Betreuer kontaktieren
  - Mittwochs-Block vorzugsweise (immer freihalten!)

Voraussetzungen für einen Schein

- Es muss ersichtlich sein, dass jeder am Projekt mitgearbeitet hat
- Beispiele
  - Akzeptanztest enthalten Autor
  - Testfälle enthalten Autor
  - Beteiligung an Treffen mit Betreuern (nicht nur physisch)
  - CVS-Commits
  - …
- Benoteter Schein

Weiteres

- Gut kommentierter Code
- Gut kommentierte Grammatik
- Code, Kommentare auf Englisch
- KEIN generierter code im CVS !!!
- Selbständig arbeiten: Gruppentreffen auch ohne Betreuer…

Fragen?