Übungen zur Vorlesung
Spezifikation eingebetteter Systeme
WS 2010/2011

AG BS
Jan Peleska
Elena Vorobev

Übungsblatt 2

Revision: 1.0

In der Vorlesung wurde ein auf hierarchischen Timed Automata basierender Formalismus zur Modellierung von Echtzeitsystemen vorgestellt. Um auch bei einer großen Anzahl von zu modellierenden Automaten eine übersichtliche Darstellung zu erhalten, wurden hierarchisch aufgebaute Komponenten eingeführt. Die einzelnen, parallel laufenden Automaten sind dann Blätter in einem Hierarchiebaum von Komponenten.

Das Ziel dieses Übungsblattes ist es, im Meta-CASE-Tool `MetaEdit+` ein Metamodell entsprechend dem vorgestellten Modellierungsformalismus zu erstellen. Die dabei zu berücksichtigenden Anforderungen sind den beiden nachfolgenden Aufgaben zu entnehmen. Das Ergebnis der beiden Aufgaben sollte jeweils ein neuer Graphtyp sein, welcher es ermöglicht, Modelle von hierarchischen Timed Automata bzw. Komponentenhierarchien gemäß den Anforderungen zu erstellen.

Aufgabe 1: Hierarchische Timed Automata

Ein hierarchischer Timed Automaton besteht aus durch Transitionen miteinander verbundenen Locations, wobei jede Location ihrerseits einen Timed Automaton enthalten darf.

Jede **Location** muss einen *Namen* haben und darf mit einer *Invariante* sowie *Entry-*, *Do-* und/oder *Exit-*Actions versehen sein. Darüber hinaus soll eine Location als *urgent* oder *committed* markiert werden können - fehlen beide Markierungen, handelt es sich um eine "normale" Location[1]. Schließlich soll es die Möglichkeit geben, eine Location als *initial* zu markieren.

Das graphische Symbol für Locations soll ein farblich ausgefülltes Rechteck mit abgerundeten Ecken sein und stets mindestens den Namen der repräsentierten Location enthalten. Sofern vorhanden, sollen auch die Invariante sowie Entry-, Do- und Exit-Actions angezeigt werden, wobei die Darstellung der jeweiligen Action sowohl die Art der Action als auch die spezifische Action selbst beinhalten soll, bspw. **entry** / a. Initiale Locations sollen einen eingehenden Pfeil erhalten. Urgent bzw. committed Locations sollen durch ein zusätzliches **U** bzw. **C** gekennzeichnet werden.

Eine **Location-Hierarchie** entsteht dadurch, dass innerhalb einer Location andere Locations und Transitionen platziert werden können. Auf der Ebene des Metamodells müssen hierfür zunächst[2] keine weiteren Vorkehrungen getroffen werden, außer, dass dieser Aspekt bei der Definition der graphischen Repräsentation von Locations berücksichtigt werden muss (s.u.). Eine Location-Hierarchie wird beim Modellieren somit allein auf Basis der Positionen der Location-Symbole definiert, genauer, auf Basis der Enthaltensein-Beziehung zwischen den zugehörigen graphischen Elementen.

Die graphische Darstellung von hierarchischen Locations soll sich von der der nicht-hierarchischen lediglich dadurch unterscheiden, dass die Rechtecke der hierarchischen Locations transparent

[1] Eine Location ist *entweder* urgent *oder* committed *oder* normal - die Einhaltung dieser Bedingung muss vorerst jedoch nicht überprüft werden.

[2] Dies wird sich ändern, sobald nicht nur die Syntax, sondern auch die Semantik betrachtet werden muss.

sind, um die Sichtbarkeit der darin platzierten Elemente zu gewährleisten.

Eine **Transition** verbindet genau eine Quell- mit genau einer Ziel-Location, wobei Quelle gleich dem Ziel sein darf. Einer Transition müssen eine *Guard-Condition*, ein *Synchronisationsevent* sowie eine *Action* zugeordnet werden können, wobei alle drei Elemente optional sind.

Transition sollen graphisch als Pfeile dargestellt werden. Sofern vorhanden, sollen die zugehörige Guard-Condition, das Synchronisationsevent sowie die Action über und/oder unter dem Pfeil angezeigt werden. Die Guard-Condition soll dabei in eckigen Klammern stehen und die Action ist durch einen Slash von den beiden anderen Elementen zu trennen, bspw. `[g]c!/a` oder auch `/a`.

Aufgabe 2: Komponentenhierarchien

Ein Komponentenhierarchie besteht aus einer oder mehreren hierarchisch aufgebauten Komponenten, wobei zwischen den einzelnen Komponenten Kommunikationskanäle deklariert werden können.

Eine **Komponente** hat immer einen *Namen* und $0 \ldots N$ *Variablendeklarationen*, wobei eine Variablendeklaration aus einem *Variablennamen* und einem *Variablentyp* besteht. Als Variablentypen sollen nur `Integer` oder `Clock` erlaubt sein. Variablennamen müssen mit einem Buchstaben beginnen und können ansonsten beliebige Kombinationen aus Buchstaben, Zahlen und Unterstrichen ein. Jede Komponente kann entweder auf eine ihre untergeordnete Komponentenhierarchie oder auf einen hierarchischen Timed Automaton verweisen (*Dekomposition*).

Als graphisches Symbol für Komponenten soll ein Rechteck verwendet werden, welcher den Namen und optional (!) die darin deklarierten Variablen enthält.

Zur Kommunikation zwischen verschiedenen Automaten sollen **Binär-** sowie **Broadcast-Kanäle** verwendet werden können. Zu diesem Zweck müssen diese zwischen Komponenten gleicher Hierarchiestufe deklariert werden können. Eine Komponente kann dabei als Sender oder als Empfänger fungieren. Ein Binärkanal darf mit genau einer Sender- und genau einer Empfängerkomponente assoziiert werden. Im Gegensatz dazu dürfen Broadcast-Kanäle auch mit mehreren Empfängerkomponenten verknüpft sein.

Die Repräsentation beider Kanaltypen soll wieder durch Pfeile erfolgen, wobei diese sich zumindest farblich unterscheiden sollen. Über den Pfeilen soll der jeweilige Kanalname angezeigt werden.

Abgabe: 07.12.2010 bis 16:00 Uhr