

## Richtlinie

# **Modellierung mit Statemate MAGNUM und Rhapsody in Micro C**

Dok.-Nr.: BMS/QM/RL/STM

Version: 1.4

Anzahl Seiten: 39

Status: formal

Stand: 20.08.2001

Autor: Th. Kreppold

Berner & Mattner Systemtechnik GmbH  
Otto-Hahn-Str. 34  
85521 Ottobrunn

## HISTORIE

Version	Datum	Zweck	KM Status	Autor
0.1	17.08.98	Draft Release	in Bearbeitung	J. Weiss
1.0	01.03.99	Vollständig überarbeitet	in Bearbeitung	G. Dotzler
1.1	01.04.99	Übersetzung ins Deutsche	in Bearbeitung	G. Winderlich
1.2	15.01.01	Überarbeitung und Erweiterung	in Bearbeitung	Th. Kreppold
1.3	14.02.01	Überarbeitung zur Übernahme als offizielle Richtlinie	vorgelegt	R. Grimm
1.3	14.02.01	Formal nach Review	Formal	I. Bräuer
1.4	24.04.01	Überarbeitung und Erweiterung	vorgelegt	Th. Kreppold

## INHALTSVERZEICHNIS

<b>0</b>	<b>Allgemeine Informationen.....</b>	<b>6</b>
0.1	ZWECK UND ANWENDUNG .....	6
0.1.1	Verbindlichkeit .....	6
0.1.2	Zusätzliche Empfehlungen.....	6
0.2	VORSCHRIFTEN UND RANDBEDINGUNGEN.....	7
0.3	BEZUGSDOKUMENTE .....	7
<b>1</b>	<b>Grundeinstellungen.....</b>	<b>8</b>
<b>2</b>	<b>Setup eines neuen Projekts.....</b>	<b>9</b>
<b>3</b>	<b>Layout von Charts .....</b>	<b>10</b>
3.1	ACTIVITYCHARTS .....	10
3.1.1	Allgemeine Regeln.....	10
3.1.2	System Context Diagram .....	12
3.1.3	Subactivitycharts.....	13
3.2	STATECHARTS .....	14
3.2.1	Allgemeine Regeln.....	14
3.2.2	Control Activities .....	15
3.2.3	Statecharts, die eine Funktionalität beschreiben .....	16
3.2.4	Testbenches .....	16
3.2.5	Top Testbench.....	17
3.3	GENERIC CHARTS, STATEMATE KOMPONENTEN UND STATEMATE LIBRARIES.....	18
3.3.1	Generic Charts.....	18
3.3.2	Statemate Komponenten und Libraries .....	19
3.4	GLOBAL DEFINITION SETS (GDS).....	19
3.5	ACTION LANGUAGE .....	19
3.6	SUBROUTINEN .....	21
<b>4</b>	<b>Namensregeln.....</b>	<b>23</b>

---

4.1	MODELLIERUNGSSPRACHE .....	23
4.2	INFORMELLE DOKUMENTATION.....	23
4.2.1	Interfaces.....	23
4.2.2	Activities .....	23
4.3	OBJEKTBEZEICHNUNGEN .....	24
4.3.1	Namenskonsistenz.....	24
4.3.2	Hauptactivities und Hauptstate Namen .....	24
4.3.3	Condition Namen .....	24
4.3.4	Variablennamen Links-Rechts-Leserichtung.....	24
4.3.5	Memnotechnische Abkürzungen .....	25
4.3.6	Empfohlene Suffixe.....	25
4.3.7	Besondere Vereinbarungen in der Automobilindustrie .....	26
<b>5</b>	<b>Check Model .....</b>	<b>27</b>
<b>6</b>	<b>Panels und Simulation .....</b>	<b>28</b>
6.1	ENDBENUTZER INTERFACE .....	28
6.2	ENGINEERING INTERFACE.....	28
6.3	APPLIKATIONS INTERFACE .....	29
6.4	SIMULATIONS PROFILE .....	29
6.5	ALLGEMEINE HINWEISE ZUR SIMULATION.....	29
<b>7</b>	<b>Tests im STM und RiMC .....</b>	<b>30</b>
<b>8</b>	<b>Allgemeine Modellierungsgrundsätze .....</b>	<b>31</b>
<b>9</b>	<b>Konfigurationsmanagement .....</b>	<b>32</b>
9.1	STATEMATE INTERNES KONFIGURATION MANAGEMENT .....	32
9.1.1	Revision History.....	32
9.1.2	Versionsmanagement .....	32
9.1.3	Externes Konfigurationsmanagement .....	32
<b>10</b>	<b>Standard Attribute .....</b>	<b>33</b>

---

<b>11</b>	<b>Empfehlung für den Modellaustausch.....</b>	<b>35</b>
<b>12</b>	<b>Rhapsody in Micro C .....</b>	<b>36</b>
	<b>Anhang A: Überblick über wichtige Preferences.....</b>	<b>37</b>
	PREFERENCES MIT PFADANGABEN:.....	37
	• AUSSCHNITT AUS DEN GENERAL PREFERENCES .....	37
	• AUSSCHNITT AUS DEN DATA DICTIONARY EDITOR PREFERENCES .....	37
	AUSSCHNITT AUS DEN PREFERENCES FÜR ACTIVITYCHARTS .....	37
	AUSSCHNITT AUS DEN PREFERENCES FÜR STATECHARTS .....	38

## ABBILDUNGSVERZEICHNIS

Abb. 1	Activitychart.....	11
Abb. 2	System Context Diagramm .....	12
Abb. 3	Subactivitychart.....	13
Abb. 4	Statechart .....	14
Abb. 5	Control Activitychart .....	16
Abb. 6	Top Testbench .....	18

## **0 ALLGEMEINE INFORMATIONEN**

### **0.1 Zweck und Anwendung**

Diese Richtlinie definiert die Konventionen zur Verwendung von Statemate MAGNUM (STM) und Rhapsody in Micro C (RiMC) für die Modellierung. Folgende Ziele sollen damit erreicht werden:

- Die Lesbarkeit der Modelle soll erhöht und diese dadurch wartungsfreundlicher gemacht werden.
- Die Portabilität der Modelle zwischen verschiedenen Entwicklern soll verbessert werden.
- Namenskonventionen sollen Eindeutigkeit gewährleisten und Namenskonflikte verhindern.
- Die Richtlinie kann als Grundlage für Qualitätssicherungsmaßnahmen, z.B. für Codeinspektionen eingesetzt werden.
- Wiederholungsfehler sollen vermieden werden.
- Die Wiederverwendbarkeit soll verbessert werden.
- Neue Projektmitglieder sollen sich leichter einarbeiten können.
- Eine höhere Qualität der Modelle soll erzielt werden.
- Eine Verkürzung der Entwicklungszeit soll sich ergeben.
- Kostenreduzierung

#### **0.1.1 Verbindlichkeit**

Konventionen sind für den Entwickler von Modellen mit Statemate MAGNUM verbindlich und müssen unbedingt eingehalten werden. Die Einhaltung der Richtlinien sicherzustellen liegt in der Verantwortung des jeweiligen QS-Beauftragten des Projektes.

#### **0.1.2 Zusätzliche Empfehlungen**

Empfehlungen sind nach Möglichkeit einzuhalten. Aus projektspezifischen Gegebenheiten können in Ausnahmefällen Abweichungen von den Empfehlungen erlaubt sein. Diese müssen zu Projektbeginn schriftlich bekannt gemacht werden.

## 0.2 Vorschriften und Randbedingungen

Keine

## 0.3 Bezugsdokumente

- [MODEL] Modeling Reactive Systems with Statecharts: The STATEMATE approach  
I-Logix Inc., Three Riverside Drive, Andover, MA 01810
- [GUIDE] Methodology and Style Guidelines  
I-Logix Inc., Three Riverside Drive, Andover, MA 01810
- [CONVERT] Converting Models from Statemate Magnum to Rhapsody in MicroC  
I-Logix Inc., Three Riverside Drive, Andover, MA 01810
- [CPPRL] Richtlinie C/C++ Programmierung  
Berner & Mattner Systemtechnik  
Dok.-Nr.: BMS/QM/RL/Cpp

## 1 GRUNDEINSTELLUNGEN

Für die Grundeinstellungen sollen die Berner & Mattner Preferences (z.B. bms<vers>\_statemate\_defaults.pref) verwendet werden.

Die Attribute von Berner & Mattner können verwendet werden (ac\_attributes.def, di\_attributes.def, ach\_attributes.def, sch\_attributes.def).

Die Preferences und Attribute sind in Dateien definiert und in den folgenden Verzeichnissen abgelegt:

STM:

[\\nt1\daten\SOFTWARE-PRODUKTE\INTERN\stm\stmm<vers>\preferences](#)

RiMC:

[\\nt1\daten\SOFTWARE-PRODUKTE\INTERN\Rhapsody in MicroC\RiMC<vers>\preferences](#)

In den Dateien:

- Berner & Mattner Preferences: bms<vers>\_statemate\_defaults.pref
- Activity Attribute Definition: ac\_attributes.def
- Data Item Attribute Definition: di\_attributes.def
- Activitychart Attribute Definition: ach\_attributes.def
- Statechart Attribute Definition: sch\_attributes.def

Dabei ist <vers> durch die relevante STM oder RiMC Version zu ersetzen, also z.B. <vers> = 21 für Statemate MAGNUM Vers. 2.1.

Ein bevorzugter Text Editor wie z.B. der „Ultra Edit Text Editor“ ([www.ultraedit.com](http://www.ultraedit.com)) kann verwendet werden. Mit diesem Texteditor ist Syntaxhervorhebung möglich. Die Datei „Wordfile.txt“, die in den oben genannten Verzeichnissen zu finden ist enthält die Einstellungen, die der „Ultra Edit Text Editor“ benötigt um die Statemate Action Language hervorzuheben.

## 2 SETUP EINES NEUEN PROJEKTS

Wird ein neues Statemate Projekt erzeugt, sollen folgende Punkte beachtet werden.

- Definition der zu verwendenden Tools und Festlegung der Toolversionen. Werden externe Konfigmanagementtools verwendet oder wird das STM eigene Konfigmanagementtool verwendet? Auf welche Art und Weise soll das Konfigurationsmanagement erfolgen? Soll eine automatisierte Anforderungsverfolgung (mit speziellen Requirementstracing-Tools wie Doors) verwendet werden oder soll dies von Hand über die Statemate Attributes erfolgen? Welche Arten von Anforderungen (nur funktionale oder alle) sollen verfolgt werden. Wird neben den Statemate Panels noch ein spezielles HMI z.B. aus Altia benötigt. Welche Rapid Prototyping Plattform kommt zum Einsatz? Wird in späteren Entwicklungsphasen - nach der Modellierung mit STM - RiMC verwendet werden?
- Die Rechte der am Projekt beteiligten User müssen im Dateisystem vergeben werden. Jeder User benötigt volle Schreib- und Leserechte auf Betriebssystemebene.
- Der Projektmanager muß bestimmt werden. Dieser hat die Rechte die Projektmitglieder in STM oder RiMC einzutragen, d.h. die Zugriffsrechte der Mitglieder auf das STM oder RiMC Projekt festzulegen.
- Die STM Datenbank sollte zentral auf dem Server im Projektverzeichnis liegen. Dabei sollte eine sinnvolle Verzeichnisstruktur wie z.B. P:\Projekte\- Bei Projektstart sollte überprüft werden ob die Preferences Datei und die Attribut Definitions Dateien vorhanden und auf dem aktuellen Stand sind. Projektspezifische Einstellungen (z.B. Default Workareapfad) können bei Bedarf in den Project Preferences geändert werden.
- Tipp: Erstellt man ein neues Projekt in Statemate sollte zuerst der Datenbankpfad in das dazugehörige Textfeld eingetragen werden und dann der Projektname eingegeben werden.

## 3 LAYOUT VON CHARTS

Die Grundprinzipien der Nutzung der Statemate MAGNUM Methodik und Style Guidelines sind in [MODEL], [GUIDE] dargelegt. Sinn und Zweck dieser Darstellung ist es, Regeln für das Layout von Charts und Namensfestlegungen innerhalb eines Statemate MAGNUM Modells festzulegen.

Die Defaultgröße der Charteditoren sollte verwendet werden. Ein Reframe der Charts sollte vermieden werden. Das Grid sollte den Wert 0.25 haben.

Allgemein sollte man beachten, dass in Activities und States, die (grafisch) keinen Inhalt haben, der Name in der Mitte der Box steht. Activities und States, die mit Inhalt gefüllt sind, sollten ihren Name im linken oberen Eck der jeweiligen Box haben.

Das Layout umfasst verschiedene Arten von Activitycharts, Statecharts und Flowcharts, abhängig von ihrem speziellen Verwendungszweck.

### 3.1 Activitycharts

#### 3.1.1 Allgemeine Regeln

Folgende Statemate Elemente sollen nicht verwendet werden:

- Controlflows,
- Junction Connectors

Die Schachtelungstiefe von Activities sollte 6-8 Ebenen nicht übersteigen.

Der Name des Hauptactivities eines Charts ist gleich dem Chartnamen (A)<sup>1</sup>. Dieses Activity wird als großes Rechteck in der Mitte des Charts gezeichnet. Die darin liegenden Activities sollten wenn möglich vertikal in der Mitte des Hauptactivities angeordnet sein. Sind die Activities mit Dataflows verbunden, können sie etwas gegeneinander verschoben werden.

Alle Datenquellen und -senken werden als externe Activities modelliert (B, C, D).

---

<sup>1</sup> Die Hinweisbuchstaben beziehen sich auf Abb. 1 Activitychart

Dataflows werden als waagrechte und senkrechte Linien gezeichnet.

Dataflows von externen Activities zur Hauptactivity eines Charts sollen von der linken Seite kommen (*E*), ausgehende Dataflows zur rechten Seite weggehen (*F*). Da externe Activities auch mehrfach im selben Chart vorkommen können, ist es möglich, sie auf beide Seiten zu plazieren. Dies ist sinnvoll wenn sie Daten-Inputs und -Outputs haben (*C*). Dataflows, die so definiert sind, beschreiben die Schnittstelle zum jeweiligen Hauptactivity. Diese Schnittstelle soll klar und eindeutig sein.

Dataflows zwischen Subactivities sollen senkrecht gezeichnet werden (*G*). Wenn sich Dataflows kreuzen ist es nicht eindeutig ob sich ein Dataflow verzweigt oder sich zwei verschiedene Linien wirklich kreuzen. Es wird deshalb empfohlen Verzweigungen graphisch durch ein „T“ (*H*) darzustellen, und Kreuzungen durch ein „+“ (*I*) zu beschreiben.

Beschriftungen von Dataflows von bzw. zur Umgebung werden in die externen Activities plaziert (*E, F*). Beschriftungen bzw. Labels von Dataflows zwischen Subactivities sollen auf der linken Seite des Pfeils plaziert werden (in Datenflußrichtung) (*G*).

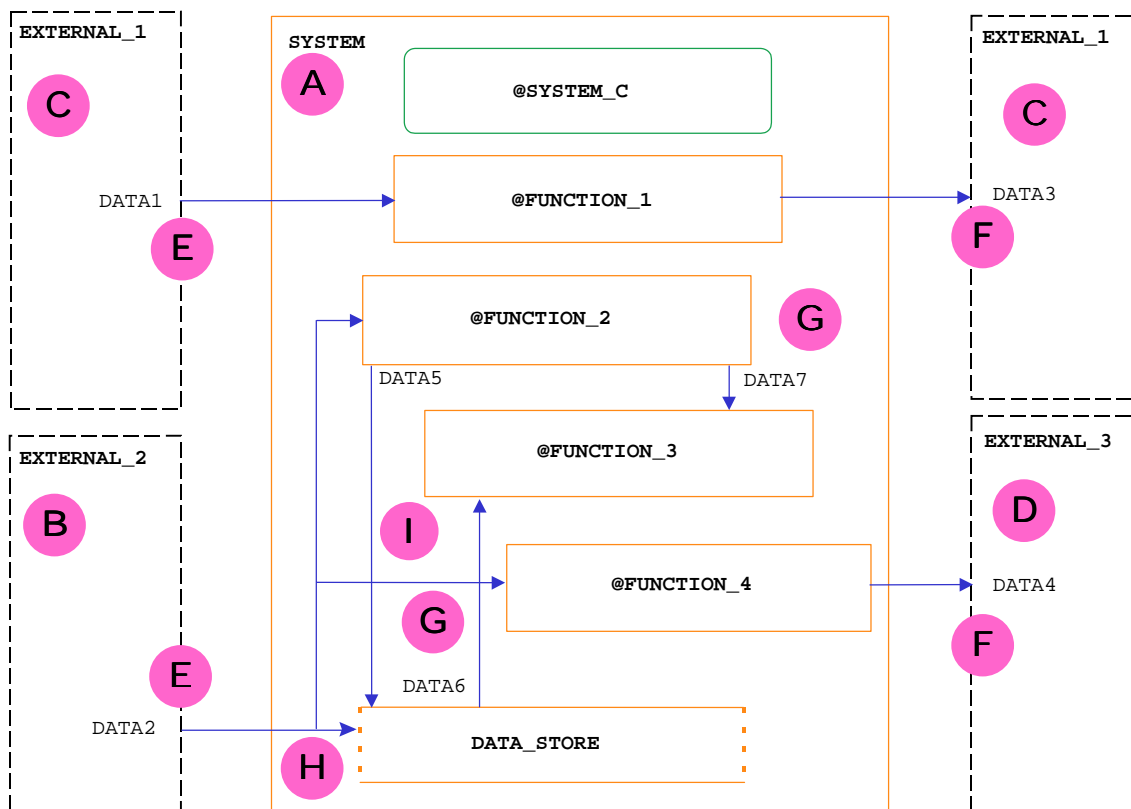
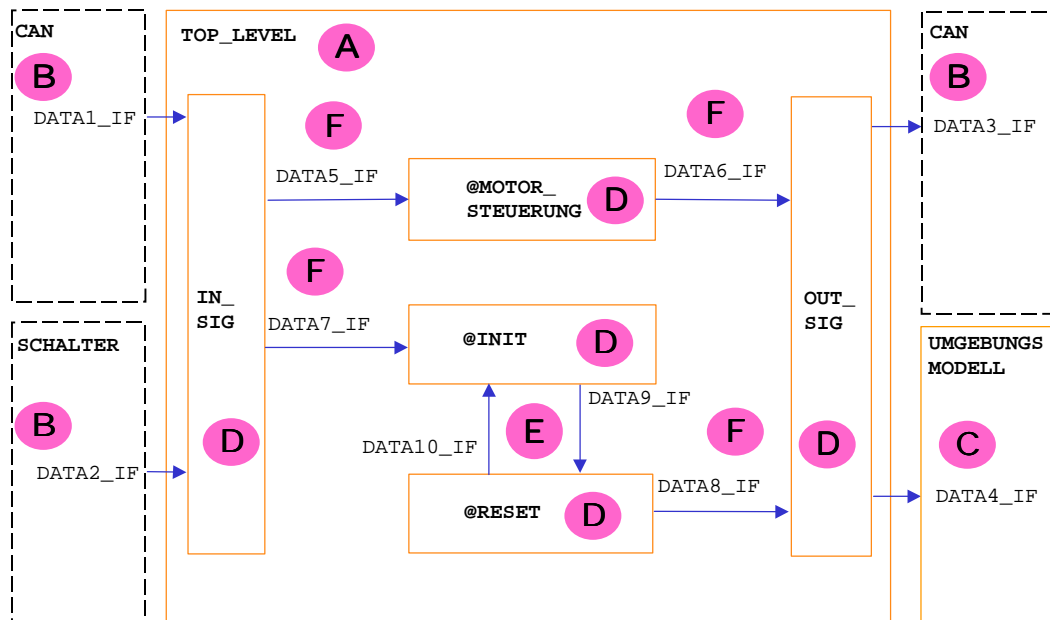


Abb. 1 Activitychart

### 3.1.2 System Context Diagram

Das Context Diagramm stellt das System in der höchsten Ebene dar (Abb. 2 System Context Diagramm). In der Mitte des Context Diagramms wird das System selbst durch eine Activity dargestellt (A)<sup>2</sup>. Externe Activities werden entweder durch die Verwendung von Environment Activity Symbolen (B) oder durch ein zusätzliches Activity dargestellt (C). Letztere werden benutzt, wenn es notwendig ist, umfangreiches Verhalten externer Teile (Umgebungsmodell) zu modellieren bzw. wenn vorhandene Komponenten eingesetzt werden sollen. Sofern vorhanden sollten solche Activities unten im Chart platziert werden.



**Abb. 2 System Context Diagramm**

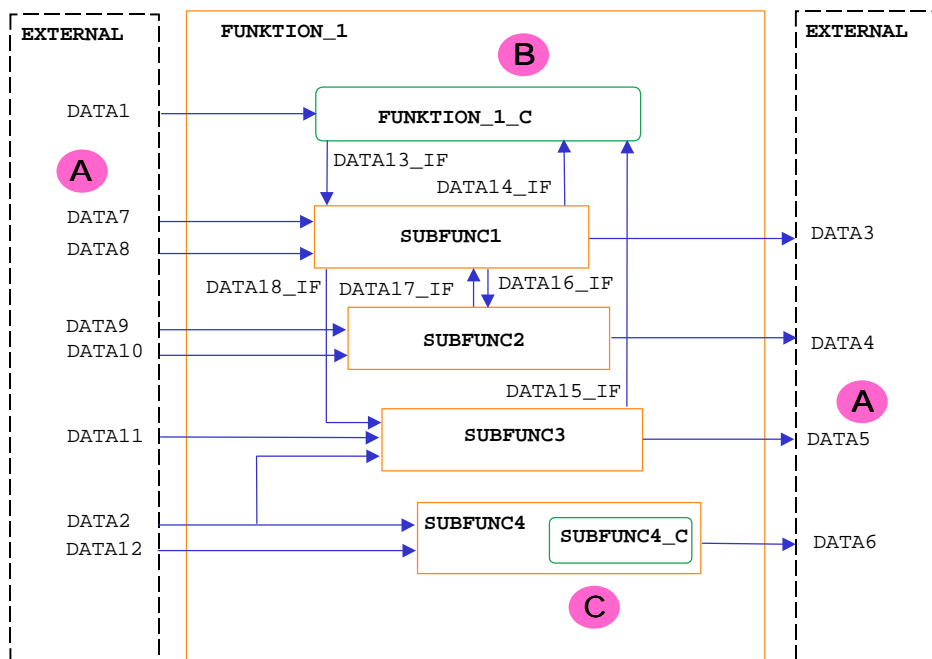
Das allgemeine Layout eines Context Diagramms bezieht sich auf ein System von unabhängigen Activities (Funktionen) (D), wobei jede ihre eigenen Input bzw. Output Information Flows besitzt (F).

Um die Lesbarkeit zu unterstützen wird empfohlen, zwischen zwei Activities jeweils nur zwei Information Flows zu benutzen, in denen die weiteren Informationen (Variablen), die zwischen den Activities ausgetauscht werden, gekapselt sind (E).

<sup>2</sup> Die Hinweisbuchstaben beziehen sich auf Abb. 2 System Context Diagramm

### 3.1.3 Subactivitycharts

Falls notwendig kann ein Activity in der nächsten Ebene weiter untergliedert werden (Abb. 3 Subactivitychart). In dieser Ebene kann das jeweilige Hauptactivity weitere Subactivities (Unterfunktionen) enthalten, die wiederum verschiedene Funktionalität haben. Diese Unterfunktionen können durch das Control Activity (B)<sup>3</sup> gesteuert werden.



**Abb. 3 Subactivitychart**

Für eine bessere Lesbarkeit werden alle hinein- und hinausgehenden Daten explizit als Dataflows dargestellt (A). Alle Information Flows sind in ihre einzelnen Elemente (Variablen) aufgelöst. Die Elemente der Schnittstelle werden dadurch verdeutlicht.

Auf unterster Ebene wird das Verhalten der einzelnen Funktionen modelliert. Dann kann die Funktion ein Control Activity (C) enthalten. Dies ist sinnvoll, wenn die Funktionalität nicht nur durch eine Mini-Spec beschrieben werden soll, sondern wirkliche Systemzustände vorhanden sind, die in Statecharts modelliert werden sollten.

Beschreibt man die Funktionalität mit Mini-Specs können oft komplexe „If then else“ Konstrukte auftreten. Diese können oft besser in einer Truth Table realisiert

<sup>3</sup> Die Hinweisbuchstaben beziehen sich auf Abb. 3 Subactivitychart

werden. Benützt man trotzdem Mini-Specs sollte man darauf achten, dass der Code nicht über die Fläche, die man auf dem Bildschirm zur Verfügung hat, hinausgeht.

## 3.2 Statecharts

### 3.2.1 Allgemeine Regeln

In jedem Statechart gibt es auf der obersten Ebene genau einen State. Der Name dieses States ist gleich dem Namen des Charts (A)<sup>4</sup>. Dieser State füllt das gesamte Chart aus.

States werden als Rechtecke mit abgerundeten Ecken gezeichnet.

Transitionen werden als Straight Lines gezeichnet. Sie sollten einander nicht kreuzen.

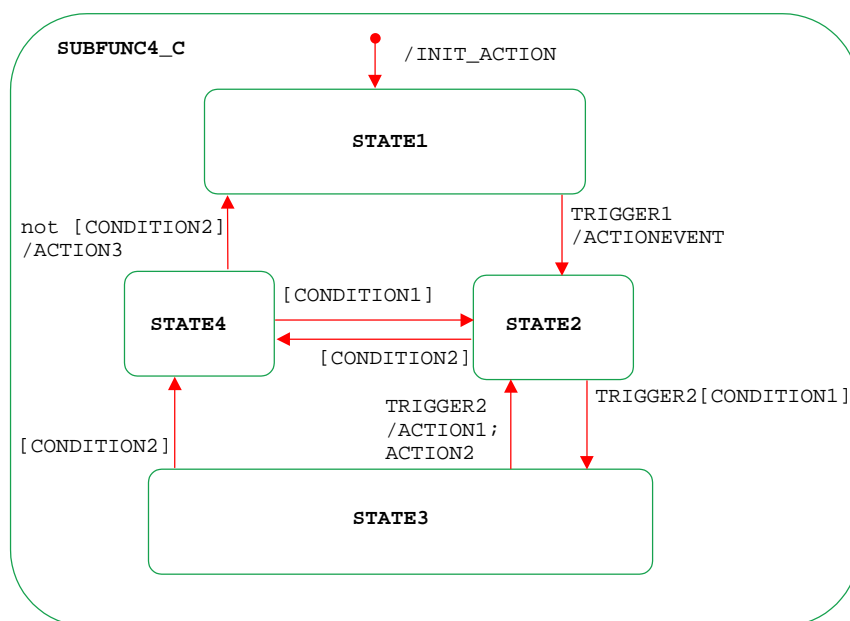


Abb. 4 Statechart

Actions, die viel Platz benötigen, sollten in den Labels als Makros definiert werden, anstatt sie ins Data Dictionary als Static Reaction zu schreiben. Die Labels sollten immer links am Anfang der Transition stehen (in Flußrichtung).

<sup>4</sup> Die Hinweisbuchstaben beziehen sich auf Abb. 4 Statechart

Darüber hinaus sollen folgende Regeln für die Platzierung der Labels beachtet werden:

Richtung der Transition	Plazierung des Labels
links nach rechts	über dem Pfeil
rechts nach links	unter dem Pfeil
oben nach unten	rechts vom Pfeil
unten nach oben	links vom Pfeil

Falls das Layout in bestimmten Fällen mit anderen Labelpositionen besser sein sollte, brauchen die oben genannte Richtlinien nicht eingehalten werden.

### 3.2.2 Control Activities

Control Activities zeigen die Steuerung der parallelen („sibling“) Activities. Ihre einzige Aufgabe ist es, diese Activities zu starten, stoppen, zu unterbrechen und fortzusetzen. Das kann durch interne (*A*)<sup>5</sup> oder externe (*B*) Events (Conditions etc.) getriggert werden (Abb. 5 Control Activitychart). In Control Activities sollten keine Algorithmen der untergeordneten Funktionen modelliert werden. Die Hauptaufgabe ist eine Art Scheduler. Berechnungen sollten nur ausgeführt werden, um den Status der zu steuernden Activities zu bestimmen. Control Activities werden nur in solchen Activities verwendet, in denen bestimmte Subactivities nur eine gewisse Zeit aktiv sein sollen. Sind alle Activities immer aktiv fällt das Control Activity weg.

Die Namen der verschiedenen States in einem Control Activity sollten folgendermaßen gewählt werden: Betritt man den State <ACTIVITYNAME>\_ACTIVE soll auch das Activity mit dem Namen <ACTIVITYNAME> gestartet werden. Betritt man den State <ACTIVITYNAME>\_PASSIVE soll auch das Activity <ACTIVITYNAME> gestoppt werden. Aus dieser Regel können verschiedene Namen für die jeweiligen States zusammengesetzt werden.

„Throughout“ und „Within“ Konstrukte sollen nicht verwendet werden, weil sie schwer verständlich und nicht in RiMC verfügbar sind.

---

<sup>5</sup> Die Hinweisbuchstaben beziehen sich auf Abb. 5 Control Activitychart

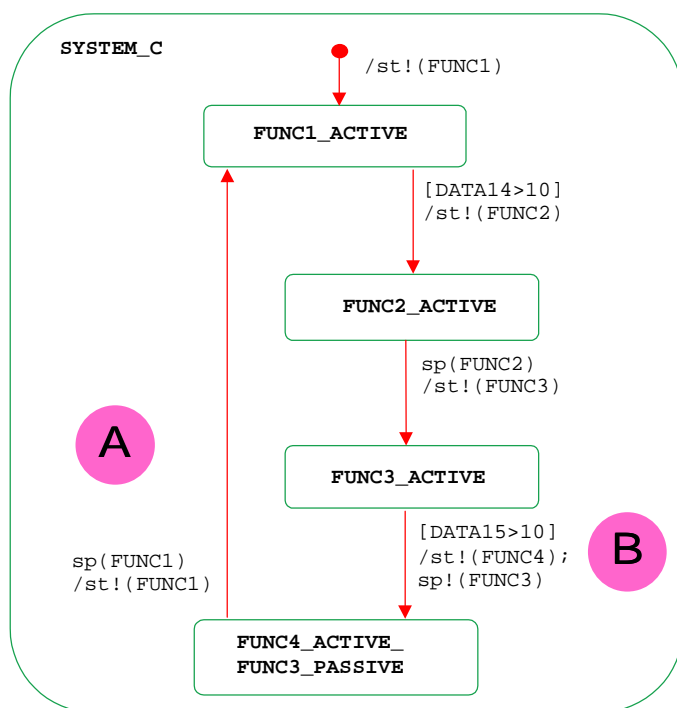


Abb. 5 Control Activitychart

### 3.2.3 Statecharts, die eine Funktionalität beschreiben

Solche Statecharts repräsentieren normalerweise das Verhalten von Embedded Systemen in der untersten Stufe, wo eine weitere funktionale Dekomposition nicht angebracht ist. Typischerweise besteht eine Activity in der untersten Stufe nur aus einem Statechart, das die gesamte Funktionalität des Activities beschreibt. Diese Statecharts können alle Features verwenden, die Statemate MAGNUM anbietet. Wie oben schon erwähnt, sollten Statecharts nur States enthalten, die auch wirklich im System vorkommen. Weiterhin sollte beachtet werden, dass ein State Zeit konsumieren kann. D.h. das System kann beliebig lange in einem State verweilen bis über eine Transition der Übergang in einen anderen State erfolgt. Der Name des States soll dabei den jeweiligen Systemzustand widerspiegeln (z.B. `POWER_ON`, `POWER_OFF`). Einfache Statennamen wie `WAIT` sollten vermieden werden. Namen wie `WAIT_FOR_XXX` sind zu bevorzugen. Quasi Flowcharts mit States zu modellieren soll unterlassen werden. Statemate bietet andere Möglichkeiten solche Konstrukte zu realisieren (z.B. Action Language).

### 3.2.4 Testbenches

Testbenches werden erstellt um

- die Simulation des Modells zu vereinfachen und zu erleichtern,
- das Systemmodell durch reproduzierbare, externe Signale zu stimulieren,

- Änderungen im System zu erkennen und Werte von Variablen zu beobachten,
- auf Systemausgaben zu reagieren, die in der realen Umgebung zu einer Reaktion von externen Modellteilen (Umgebung) führen würde. Wenn diese Reaktionen zu komplex sind, um in einer Testbench modelliert zu werden, können Activities eingesetzt werden. Dies ist besonders vorteilhaft, wenn bereits generische Activities vorhanden sind, die man als Umgebungsmodell verwenden kann. Man verwendet sozusagen eine „Luxus Testbenchactivity“.
- die Panel Animation zu steuern und Modelldatenstrukturen in Datenstrukturen für Paneldarstellung zu übersetzen,
- Variablen mit vorgegebenen Werten zu initialisieren,
- eine Systemuhr (Timerticks) zu bauen.

### 3.2.5 Top Testbench

Pro Simulationsprofile sollte eine Top Testbench, die alle nötigen Datendefinitionen zur Verfügung stellt und alle weiter benötigten spezifischen Testbenches (Statecharts) als parallele Automaten enthält, vorhanden sein. Das ermöglicht den Datenaustausch zwischen den parallelen Automaten. Bei vielen einzelnen Testbench Charts ist kein Datenaustausch untereinander möglich.

Wenn die Simulation oder das Modell es erfordert, sollten in der Testbench die parallelen Automaten PANEL\_DRIVER> und/oder INIT\_VALUES> mit ihren Static Reactions definiert sein. INIT\_VALUES> soll die Initialisierung der Applikationsdaten im Modell durchführen. In PANEL\_DRIVER> sollen die Daten erzeugt oder verarbeitet werden, die zum Treiben der Panels in der Simulation nötig sind.

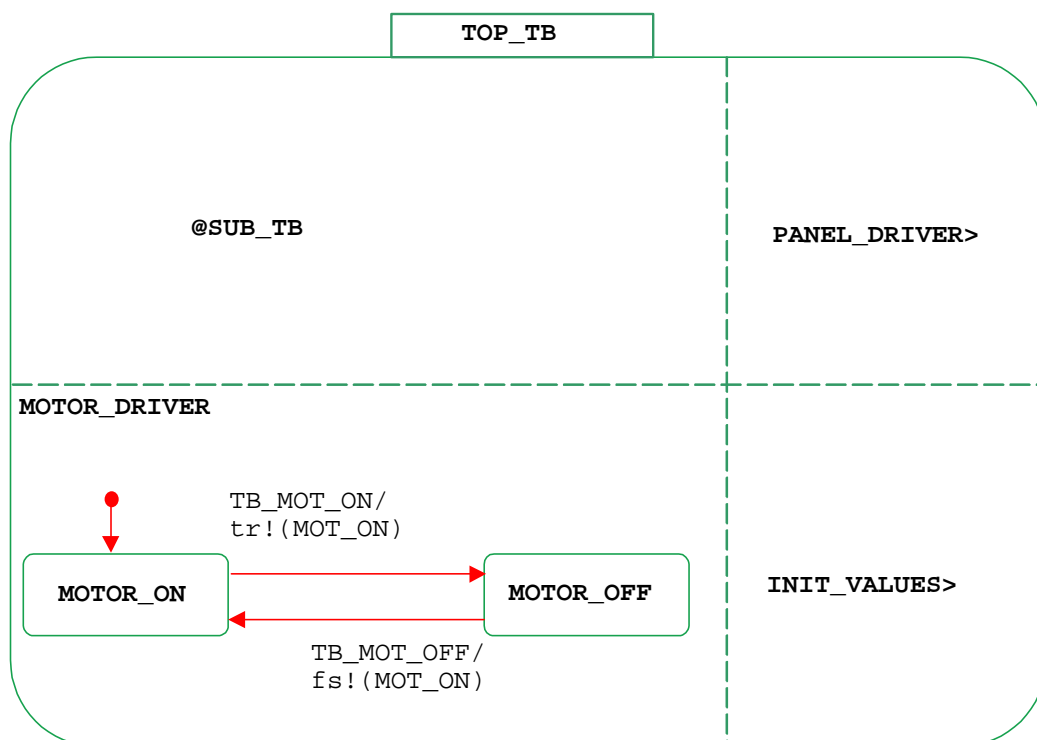


Abb. 6 Top Testbench

### 3.3 Generic Charts, StateMate Komponenten und StateMate Libraries

#### 3.3.1 Generic Charts

Ein Generic Chart dient als wiederverwendbares Modul.

Es sollen nur Activitycharts als Generic Charts verwendet werden. Generic Statecharts sind verboten, weil die Schnittstellen nicht sichtbar sind. Alle Regeln für Activitycharts kommen auch für Generic Activitycharts zur Anwendung.

Für alle Generic Activitycharts soll ein Context Diagramm gezeichnet werden, das eingehende und ausgehende Data Flows für formale Parameter und Environment Activities zeigt. Diese Vorgehensweise macht die Schnittstelle sichtbar. Environment Activities sind mit einem möglichst neutralen Namen zu versehen (z.B. INPUT oder OUTPUT).

Generic Charts sollen nicht auf globale Variablen verweisen. Die generische Schnittstelle soll klein (für die Wartung) und effektiv (für die Wiederverwendbarkeit) gestaltet werden. D.h. GDS sollten wenn möglich vermieden werden.

Daten, die zur Initialisierung oder Konfiguration des Generic Activities nötig sind, sollten nicht in der Schnittstelle bzw. als Parameter auftauchen.

Beim Test bzw. bei der Simulation von Generic Activitycharts sollte ein Top Activitychart erstellt werden, um eine Instanz des Generic Activitycharts erzeugen zu können. Hierbei sollten die Parameter und Bindings gleiche Namen haben.

Instanzen von generischen Charts, die wiederum in einem generischen Chart auftauchen, sollten die Modellierungstiefe von 3 Ebenen nicht überschreiten.

Generic Activities können nur die globale Zeitbasis verwenden. Deshalb wird die Einführung eines Inputs, der die Zeitkonstanten skaliert empfohlen:

Schnittstellenvariable: TM\_SCALE (Data Item, Real, Schnittstellenparameter der konstant ist)

Verwendung: VARIABLENBEZEICHNER (Data Item, Integer, Compound, Definition: TRUNC(ZAHLENWERT\*TM\_SCALE)).

### 3.3.2 StateMate Komponenten und Libraries

Wird ein Generic Activity als StateMate Komponente verwendet, sollte sie in einem neuen StateMate Projekt entwickelt werden. Dieses wird dann als Library definiert.

Für Komponenten von StateMate Libraries gelten die selben Regeln wie bei Generic Charts. Jede Komponente hat eine Long Description für eine detaillierte Interface Beschreibung. Die Short Description des Top Activities soll benützt werden, da sie im Komponenten Browser wieder zu sehen ist.

### 3.4 Global Definition Sets (GDS)

GDS sind für alle Charts in einem Projekt sichtbar. GDS sollen alle Datentypdefinitionen enthalten.

GDS sollen alle Konstantendefinitionen enthalten, die global zum gesamten Projekt sind und als Kodierwerte genutzt werden.

Es wird jeweils ein GDS für Typdefinitionen und eines für Konstantendefinitionen empfohlen.

### 3.5 Action Language

Die ausführliche Nutzung von Klammern wird empfohlen um Fehler zu vermeiden.

((!>=MAX) or (!<=MIN))

not EVENT Konstrukte sollen im Modell nicht vorkommen.

Der Slash soll immer am Ende einer Zeile stehen. Ausnahme: An der Default Transition beginnt man mit dem Slash, da dort kein Trigger vorkommen darf.

```
EVENT[CONDITION] /  
ACTION  
  
/ACTION
```

Der Operator soll bei Zeilenumbrüchen am Ende der Zeile stehen.

```
[CONDITION or  
CONDITION and  
CONDITION]
```

Der Condition Teil eines Triggers sollte immer nur aus einer eckigen Klammer bestehen z.B.: [(C1 or C2) and (C3 or C4)] anstatt [C1 or C2] and [C3 or C4].

Ablaufsteuerungselemente wie *if then else* Konstrukte oder *for* Schleifen können von der Textstruktur so aufgebaut werden, wie das z.B. in der „C“ Programmierung üblich ist.

```
if CONDITION then  
    ACTION1  
else  
    ACTION2  
end if
```

Die Events `rd()` und `wr()` sollten nicht verwendet werden.

Bei Platzproblemen in einem Chart sollen Compound Elemente (Makros) verwendet werden. Diese sind zu verwenden, wenn das Chart durch viel Text unübersichtlich wird. Beispielsweise kann der Ausdruck

```
FRONT_LEFT_DOOR_OPEN or  
FRONT_RIGHT_DOOR_OPEN or  
REAR_ LEFT _DOOR_OPEN or  
REAR_ RIGHT _DOOR_OPEN or  
BACK_DOOR_OPEN
```

durch ein Compound Element `ANY_DOOR_OPEN_M` ersetzt werden.

Combinatorial Assignments sollen nicht verwendet werden.

Context Variablen (`$VARIABLE`) sollen nicht verwendet werden. Statt dessen können in Subroutinen Variablen verwendet werden, die mit der gleichen Funktionalität ausgestattet sind.

In Statemate können Enum Variablen definiert werden. Dazu müssen User Defined Types definiert werden. Es ist darauf zu achten, dass die Werte von verschiedenen Enum Typen keine gleichen Bezeichner haben. Bei Enum Werten wird deshalb empfohlen einen Suffix zu verwenden, der auch im Typnamen auftaucht (z.B. Typ: `DRV_STAT_T` Werte: `{DRV_ON,DRV_OFF,DRV_ERROR}`)

Mini-Specs mit mehreren Triggern (zu erkennen an den doppelten Semikolons `;;`) sollen immer mit einer Leerzeile getrennt werden.

```
fs(FL_DOOR_OPEN) /  
LIGHT_OFF ; ;
```

```
tr(ANY_DOOR_OPEN_M) /  
LIGHT_ON
```

### 3.6 Subroutinen

Subroutinen sollten nicht mehr als fünf Parameter enthalten.

Werden Subroutines mit C-Code Implementierung verwendet, ist es sinnvoll, die firmeninternen Codier Richtlinien einzuhalten. Bei Berner & Mattner ist die Richtlinie „C/C++ Programmierung“ [CPPRL] zu verwenden.

Mehrzeilige Kommentare sind immer an den Zeilenanfang zu stellen.

```
/*Kommentar Zeile 1  
*Kommentar Zeile 2  
*Kommentar Zeile n  
*/
```

Soll Code auskommentiert werden, ist das mit bedingter Compilierung zu realisieren.

```
#if 0  
Code Zeile 1  
Code Zeile 2
```

Code Zeile n  
#endif

## 4 NAMENSREGELN

### 4.1 Modellierungssprache

Soweit es sich nicht um internationale Projekte handelt, muss als erstes die verwendete Sprache (englisch/deutsch) festgelegt werden.

Es wird empfohlen Bezeichner zu wählen, die im Deutschen und Englischen ähnlich oder gleich sind. So kann das Modell später leicht an internationale Bedürfnisse angepasst werden.

In Statemate dürfen keine Umlaute oder Sonderzeichen verwendet werden (z.B. in der Long Description)

### 4.2 Informelle Dokumentation

Angaben zur Funktionalität werden in der Long Description eingetragen. Die Long Description im Chart Data Dictionary ist hiervon ausgeschlossen. Statt dessen soll die Long Description des Activities verwendet werden.

Die Information, die in einer Long Description des Activities steht könnte folgendermaßen aussehen: „Zweck des Activities, Eingangs- und Ausgangsparameter Testkriterien“

#### 4.2.1 Interfaces

Alle Einträge im Data Dictionary die Teil eines Interfaces (Parameter eines Generic Activities) sind, sollten eine textuelle Beschreibung besitzen (Long Description oder Short Description). Das Top Activity eines Generic Activities benötigt eine Short Description, da diese im Komponenten Browser auftaucht

#### 4.2.2 Activities

Die Teile des Anforderungsdokuments (z.B. das Lastenheft), die von einer Activity erfüllt werden, sollen in ihre Long Description übernommen werden. Auch die funktionelle Beschreibung des Activities soll in die Long Description. Es ist auch möglich auf externe Dokumente per „Link to External File“ zu verweisen. Dies ist besonders bei Grafiken empfehlenswert.

Externe bzw. Environment Activities stellen meistens ein Gerät, Bauteil o.ä. dar. Hier sollten die Namen dieser Komponenten als Bezeichner verwendet werden (CONTROLLER, MOTOR usw.). Dies gilt oft auch für System Context Diagramme. Subactivities beschreiben überwiegend die verschiedenen Funktionen in ei-

nem Modell. Es wird darum empfohlen, diese Funktionalität auch im Namen widerzuspiegeln (z.B. CALCULATE\_MOTORPOSITION ).

### 4.3 Objektbezeichnungen

Die Namen aller Objekte, die im Modell verwendet werden, werden dem Anforderungsdokument entnommen. Falls in der Anforderung etwas nicht klar benannt ist, sollte ein passender, bekannter, neuer Name eingeführt werden.

Prinzipiell bekommt jedes Element einen eindeutigen, aussagekräftige Namen. Es ist darauf zu achten, dass gewisse Schemata eingehalten werden. Beispielsweise ist es bei CAN Bezeichnern üblich die Namen aus der CAN Matrix zu entnehmen. Dies wird sogar von einigen Firmen gefordert.

#### 4.3.1 Namenskonsistenz

Unterschiedliche Variablen mit dem selben Namen sollten vermieden werden. Dies gilt besonders für Schnittstellenvariablen.

#### 4.3.2 Hauptactivities und Hauptstate Namen

Hauptactivities bzw. Hauptstates haben den selben Namen wie das jeweilige Chart.

#### 4.3.3 Condition Namen

Conditionnamen geben immer den true Zustand wieder. Heißt die Variable z.B. POWER\_ON, ist im true Zustand das modellierte Gerät in Betrieb. Heißt die Variable POWER\_OFF ist im true Zustand das modellierte Gerät außer Betrieb.

#### 4.3.4 Variablennamen Links-Rechts-Leserichtung

Der Hauptteil eines Variablenbezeichners sollte sich aus Abkürzungen von mehreren Namensteilen in einer festen Reihenfolge zusammensetzen. Am einfachsten so wie sie auch ausgesprochen werden.

Also Wischermotorposition heißt WI\_MOT\_POS und nicht POS\_MOT\_WI.

Allgemein setzt man vorher ein Kürzel für ein spezifisches Objekt. z.B. für vorne links FL\_ gefolgt von der Eigenschaft z.B. Geschwindigkeit \_V\_ oder Drehzahl \_N\_.

Weitere allgemein übliche Beispiele:

Timer	_TM_
Counter	_CNT_
Status	_STAT_
Positionen	_POS_
Richtungen	_DIR_
Anweisung	_CMD_

Als letztes kann noch ein Qualifizierer angehängt werden, um die Eindeutigkeit zu gewährleisten.

Einige Beispiele, wie sie im Context Diagramm auftreten können:

Eingänge	_I
Ausgänge	_O
Schalter	_SW
Busanforderung	_ANF
LED	_LED

#### 4.3.5 Memnotechnische Abkürzungen

Um zu lange Namen zu vermeiden, können memnotechnische Abkürzungen eingeführt werden. Ist das der Fall, muss die Erklärung der Abkürzung in der Short Description des Elements angegeben werden. Um Objekttypen einfach zu unterscheiden, sollten Suffixe benutzt werden.

#### 4.3.6 Empfohlene Suffixe

##### Charts und Files

Objekt Typ	Suffix
Control Activitychart <N> = dazugehöriger hierarchischer Activity Name	<N>_C
Testbench	_TB
Procedural Statechart	_PS
Global Definition Set	_G
Panel	_P

##### Variablen und Konstanten

Information Flow	_IF
User Defined Type	_T
Compound (Makro)	_M
Variablen, die in einer Testbench definiert sind	TB_
Subroutines	_SB

### Verzeichnisstruktur

Project Directory Name	<i>Projectname.db</i>
Workarea Directory Name	<i>Projectname_Erstellerkürzel.wa</i>

### Statemate Profiles

Check Model Profile	CHK_
Simulation Profile	SIM_
C Code Generation Profile	C_

Für den eigentlichen Bezeichner erscheint die Angabe des betroffenen Modellteils bzw. die besondere Einstellung (z.B. MODULX\_TB, ALL\_GBA, ALL\_VXWORKS) als hilfreich.

#### 4.3.7 Besondere Vereinbarungen in der Automobilindustrie

In der Automobilindustrie nehmen die einstellbaren Parameter (Applikationsdaten) und die verschiedene Ausbaustufen codierenden Konstanten eine Sonderstellung ein.

Diese Daten werden durch besondere Prefixe gekennzeichnet (z.B. A\_REST\_MOT\_MAX\_POS).

Daten, die im nichtflüchtigen Speicher (EEPROM) abgelegt werden sollen, werden durch den Prefix EE\_ gekennzeichnet.

CAN Bezeichner werden üblicherweise aus der CAN Matrix übernommen. Die Signale werden dabei als kontinuierlich anliegend angenommen. D.h. sie sind als einfache Integer Variablen anzusehen.

Applikationsdaten	A_
Codierdaten	C_
Nichtflüchtiger Speicher (EEPROM)	EE_

## 5 CHECK MODEL

Grundsätzlich sind alle Charts mit dem Check Model Feature zu testen. Es sollte ein Check Model Profile angelegt werden um komplexe Modelle zu checken. Im Charteditor können die jeweiligen Charts einem schnellen Test ohne Check Model Profile unterzogen werden.

Correctness Fehler sind im Modell nicht erlaubt.

Completeness Fehler können unterdrückt werden. In einem Review sollte aber dann geklärt werden ob es sinnvoll ist, diese Fehler wirklich nicht mehr anzuzeigen.

In Hinblick auf die spätere Verwendung von Rhapsody in Micro C kann das Modell dahin getestet werden, ob Probleme auftreten, wenn Charts aus der Statemate Datenbank in Rhapsody in Micro C verwendet werden. Statemate kann diese Kompatibilität untersuchen. Weiterhin sollte man [CONVERT] beachten, wenn ein Statemate Modell später in Rhapsody in Micro C verwendet werden soll.

## 6 PANELS UND SIMULATION

Panels werden für verschiedene Zwecke eingesetzt:

- Endbenutzer Interface
- Engineering Interface
- Applikations Interface

### 6.1 Endbenutzer Interface

Es wird empfohlen, das Endbenutzer Interface Panel von anderen Panels zu trennen. Es soll ein Top Activity geben (System Context Diagramm), das die Schnittstelle zeigt und das Verhalten komplett abbildet. Im Endbenutzer Interface soll sich nur auf diese Schnittstelle bezogen werden. Referenzen zu States im Modell oder zu lokalen Variablen sollen nicht verwendet werden, da sonst interne Abhängigkeiten entstehen.

Die Paneloberfläche sollte möglichst nahe an der späteren Geräteoberfläche, Schaltplan o.ä. sein. Es müssen Elemente für alle Eingaben vorgesehen werden, damit der Endbenutzer mit dem System kommunizieren kann. Alle Ausgaben, die für ihn sichtbar sind, müssen in diesem Panel angezeigt werden. Andererseits sollen aber keine zusätzlichen Informationen in diesem Panel eingegeben oder ausgegeben werden.

### 6.2 Engineering Interface

Panels dieser Art bieten im allgemeinen dem Entwickler die Möglichkeit das Modell zu testen in dem Daten eingegeben werden können, die von externen Quellen kommen (Nachbarkomponenten, Umgebung, etc.).

Hier können auch die verschiedensten Outputs dazu benutzt werden, interessante interne Zustände des Modells darzustellen, so dass es leichter fällt Systemänderungen zu verfolgen.

Im Engineering Panel sollten die im Modell verwendeten Variablennamen verwendet werden.

Engineering Panels können in der Regel mit dem Panelbuilder automatisch erzeugt werden.

### 6.3 Applikations Interface

In Modellen in denen Applikationsdaten vorkommen soll ein separates Panel vorhanden sein, in dem diese eingestellt werden können.

### 6.4 Simulations Profile

Dies wird benutzt, um das Modell im Ganzen zu starten bzw. Restarts durchzuführen, Simulationsparameter wie die Zeit (Auflösung, Autorun Time Factor) zu setzen und automatisierte Simulationsprozesse zu organisieren.

### 6.5 Allgemeine Hinweise zur Simulation

Eine automatische Simulation mit einer Testbench oder einem Playbackfile sollte zur Verfügung gestellt werden, wobei pro Test für ein Modul ein extra Simulationsprofile erstellt werden sollte. Dokumentationen zur Simulation sollen als Kommentare im Playbackfile stehen (mögliche Formate: */\* Dokumentation \*/* oder *-- Dokumentation*), falls eines vorhanden ist. In der Dokumentation muss der Verweis auf das verwendete Simulationsprofile stehen. Ein Tracefile sollte erstellt werden. Mit dem B&M Coverage Tool kann dann die Testabdeckung untersucht werden.

Bei einer automatischen Simulation sind die Inputvariablen im Panel auf die Option -Input und Output- zu stellen, da sonst die Veränderungen an den Eingängen nicht verfolgt werden können.

Als minimale globale Zeitbasis sollte im Profile 10ms gewählt werden, da die Simulation sonst sehr langsam werden kann. Simulationszeit ist nicht gleich Echtzeit!

In Statemate modellierte digitale Filter und ähnliche Bausteine aus der „kontinuierlichen Welt“ müssen besonders intensiv betrachtet werden. Bei hohen Frequenzen muss die Zeitbasis sehr klein sein. Dies kann unter Umständen die gesamte Simulation verlangsamen. Hier gilt es ggf. abzuwägen, ob die Performanceinbussen hingenommen werden, die Filter vereinfacht werden oder der Test auf der Basis des generierten C Codes vorgenommen werden soll.

## 7 TESTS IM STM UND RIMC

Bei der Modellierung in STM sollte man immer die Testbarkeit des Modells einplanen und auch dokumentieren. Ein wichtiges Feature in STM und RiMC sind z.B. die Testvektoren und die Playbackfiles, die man bei Tests unbedingt benutzen sollte. Durch die Benutzung der Testvektoren und Playbackfiles lässt sich die Reproduzierbarkeit der Tests gewährleisten.

Regressionstests können mit Hilfe des Command Line Interface durchgeführt werden. Hier können in einer Batch Datei alle Tests, die in einem STM Projekt definiert sind, aufgerufen werden. Ein benutzerdefinierter Durchlauf aller Tests ist dadurch möglich.

Beispiel einer Batch Datei.:

```
call D:\Programme\I-Logix\stmm\21\bin\run_stmm.bat
-wa E:\users\wa\test.wa\
-simulation SIM_TEST_PROFILE
-chart TEST_CHART
-input E:\users\wa\test.wa\ana\input.txt
-terminate
```

Testfunktionen, (z.B. Diagnosefunktionen) wie sie typisch in der späteren Software implementiert werden, sollten nicht modelliert werden. Diese blähen das Modell nur unnötig auf. Wichtige Bestandteile des Modells wie z.B. ein Fehlerzähler sollen natürlich implementiert werden.

Wichtig bei den Tests sind die Untersuchung des Modells im Grenzbereich. Overflows und Plausibilitätstests sollten dabei durchgeführt werden.

Berner & Mattner stellt für Testausgaben ein zusätzliches Debug Modul für C-Subroutinen zur Verfügung. Hier können Daten entweder in eine Datei oder auf die Standardausgabe geleitet werden. Weiterhin können verschiedene Debug-Level definiert werden, die die Datenausgabe ab einem bestimmten Level vornimmt oder auch nicht. Die Headerdatei, die für die Benutzung eingebunden werden muß ist unter folgendem Pfad zu finden: [\\Nt1\daten\SOFTWARE-PRODUKTE\INTERN\stm\bms\\_debug.h](\\Nt1\daten\SOFTWARE-PRODUKTE\INTERN\stm\bms_debug.h). Das Debug Modul ist im Source Code dokumentiert.

RiMC Modelle sollten, vor dem Test auf dem Target, immer unter Windows und zusätzlicher Verwendung der GBA (Graphical Back Animation) getestet werden. Insbesondere stehen hier auch die im Vergleich zur Targetentwicklungsumgebung üblicherweise komfortablen Debugmöglichkeiten von Windows (z.B. Microsoft Visual C++ Debugger) zur Verfügung.

## 8 ALLGEMEINE MODELLIERUNGSGRUNDSÄTZE

Jedes Datum sollte nur von einer Quelle (Activity) beschrieben werden.

Jedes Datum sollte initialisiert werden.

In STM sind keine Hardwareabhängigkeiten zu modellieren (z.B. CAN Treiber). Treten Hardwareabhängigkeiten in Zusammenhang mit dem Rapid Prototyping auf sind diese über das Embedded Rapid Prototyper Modul im I/O-Mapping bzw. den Card Files zu kapseln.

In dem Konstrukt `tm()`, `sc!()` und `dly()` sind immer nur Integer Werte als Zeiten zu definieren.

Arrays sollen bevorzugt mit dem Index 0 beginnen und aufwärts bis n laufen.

Bei Strings soll die Länge immer explizit angegeben werden.

Konstanten sollen nicht als Zahlenwert sondern als Konstantendefinition (Data Item mit Usage Constant) und einem definierten Wert im Modell auftauchen.

In einem Projekt sollen so wenig Global Definition Sets wie möglich definiert sein.

## 9 KONFIGURATIONSMANAGEMENT

### 9.1 Statemate internes Konfiguration Management

#### 9.1.1 Revision History

Wenn das Statemate eigene Konfiguration Management verwendet wird, wird eine Revision History zu jedem Chart in den Chart Attributen durchgeführt.

Dieses Attribut enthält eine Beispielzeile der Form: Version / Datum / Bearbeiterkürzel / Änderungstext. Diese Vorlage dient dazu die Revision History einheitlich zu gestalten.

#### 9.1.2 Versionsmanagement

Die einfachste Form einen Bearbeitungsstand in der Workarea zu sichern ist die Erstellung eines Config Files. Es wird dabei ein reproduzierbarer Stand der Workarea in der Datenbank archiviert.

Config Files Namen beginnen mit dem Prefix CFG\_ und enden mit dem Erstellungsdatum im Format JJMMTT.

Für temporäre Konfigurationen wird zusätzlich ein Erstellerkürzel verwendet (z.B. CFG\_TK\_011201), für durch den Project Manager erzeugt Stände wird eine Versionsnummer vergeben (z.B. CFG\_01\_011201).

Config Files sind nach der Erstellung der Konfiguration aus der Workarea zu löschen. D.h. sie sollen in der Datenbank immer nur die Version 1 haben.

Im Config File wird die Erläuterung zur jeweiligen Konfiguration als Statemate Kommentar (*--Dokumentation*) extra eingegeben. Zur effizienten Nutzung muss man im Fenster „Create Configuration“ die Option „New Configuration File“ deaktivieren. Nach Erzeugung der Konfiguration wird dann der erläuternde Text eingegeben und schliesslich das die Config File mit „Check In & Delete“ in der Datenbank archiviert.

#### 9.1.3 Externes Konfigurationsmanagement

Wird ein externes Konfigurationsmanagement Tool (PVCS, CVS, Source Integrity etc.) verwendet kann das Versionsmanagement und die Revision History extern durchgeführt werden. Dabei kann sowohl die Konfigurationsmanagement API Schnittstelle genutzt oder auf der Backup und Restore Funktion aufgesetzt werden.

## 10 STANDARD ATTRIBUTE

Es sind die für Activities, Data Items, Statecharts und Activitycharts definierten Berner & Mattner Standard Attribute zu verwenden (können per „LOAD ATTRIBUTES“ mit default Werten belegt werden). Im folgenden Text sind die Standardattribute von Berner & Mattner aufgeführt. Als weiteres Attribut könnte man z.B. die Datenrate von externen Signalen definieren. Der Name des Attributs könnte hier beispielsweise MAX RATE OF CHANGE lauten.

Activity Attribute:

STATUS	Definiert den Bearbeitungsstand der Funktion (Open, Untested, Tested, Released)
PRIORITY	Definiert die Wichtigkeit im Gesamtsystem der Funktion (Optional, Neccessary, Mandatory)
CRITICALITY	Definiert die Sicherheitsrelevanz und somit die nötige Testtiefe der Funktion (No Effect, Minor, Major, Hazardous, Catastrophic)
REQ_NUM	Dient als Verbindung zur Anforderungsverfolgung bei manueller Verwaltung der Anforderungen (z.B. 3.2.1)

Data Item Attribute:

UNIT	Definiert die Einheit (z.B. ms)
RANGE	Definiert den Wertebereich (z.B. 0..210)
RESOLUTION	Definiert die Auflösung (z.B. 10)

Activitychart Attribute:

REVISION	1.0/2001.01.01/TK/Text
----------	------------------------

Statechart Attribute:

REVISION	1.0/2001.01.01/TK/Text
----------	------------------------

Allgemeine Attribute (Sind per Hand einzutragen):

EXT_LINK	Definiert für das B&M Advanced Documentor Template die Einbindung der externen Files. (Mögliche Werte: Link, Include, Reference)
----------	--

## 11 EMPFEHLUNG FÜR DEN MODELLAUSTAUSCH

Für einen Modellaustausch wird empfohlen eine Konfiguration des Modells zu exportieren und dieses in ein neues, leeres Projekt zu importieren

Folgende Vorgehensweise wird empfohlen:

- Ein Config File anlegen.
- Im Datenbank Browser im Menü „File“ die Option „Export Configuration“ wählen.
- Das Output Directory definieren, „Invoke script to pack“ wählen und bestätigen.
- Die erzeugte gepackte Datei kann nun z.B. per E-Mail ausgetauscht werden.
- Der Empfänger legt nun in Statemate ein neues Projekt an
- Im neuen Projekt den Workareabrowser öffnen. Im Menü „File“ die Option „Import from a packed file“ wählen. Die empfangene, komprimierte Datei auswählen und bestätigen.
- Die im Config File definierten Elemente erscheinen im Workareabrowser.

## 12 RHAPSODY IN MICRO C

Charakteristische Datentypen, die immer wieder im Modell bzw. in der Software auftauchen (z.B. ein „unsigned int“), sollten als „User Defined Type“ (z.B. als u-int16) definiert werden.

Im Code Generation Profile sollten immer so viele Module definiert sein, wie Tasks im Top Level Activitychart vorhanden sind. Die Namen der Module sind dabei gleich den Namen der Tasks zu wählen.

Subroutinen Namen sollen mit Großbuchstaben beginnen.

Namen von Konstanten sollen immer groß geschrieben sein.

Dateinamen sollen immer klein geschrieben sein.

In For-Schleifen soll, wenn möglich, die Zählvariable von n aus abwärts laufen.

Falls ein preemptives Target Betriebssystem verwendet wird, sollten aus Gründen der Datenkonsistenz immer lokale Kopien von globalen Variablen gemacht werden und mit diesen gearbeitet werden. Die lokalen Variablen sollten dabei immer den jeweiligen Tasknamen mit im Bezeichner tragen.

## **ANHANG A: ÜBERBLICK ÜBER WICHTIGE PREFERENCES**

### **Preferences mit Pfadangaben:**

- **Ausschnitt aus den General Preferences**

Editor Command Line enthält Pfad zum gewünschten Editor  
z.B. C:/Programme/Ultraedt/uedit32.exe

Pager Command Line enthält Pfad zum gewünschten Editor  
z.B. C:/Programme/Ultraedt/uedit32.exe

Postscript Viewer Command Line enthält Pfad zum gewünschten Viewer  
z.B. C:/Programme/Ghost/gsview32.exe

Data Bank Default Directory enthält Pfad zum gewünschten Verzeichnis  
z.B. P:/Projekte

Workarea Default Directory enthält Pfad zum gewünschten Verzeichnis  
z.B. C:/Users

- **Ausschnitt aus den Data Dictionary Editor Preferences**

Long Description Editor Command Line enthält Pfad zum gewünschten Editor  
z.B. C:/Programme/Ultraedt/uedit32.exe

Action Language Editor Command Line enthält Pfad zum gewünschten Editor  
z.B. C:/Programme/Ultraedt/uedit32.exe

User Code Editor Command Line enthält Pfad zum gewünschten Editor  
z.B. C:/Programme/Ultraedt/uedit32.exe

Attribute Definitions Directory enthält Pfad zum gewünschten Verzeichnis  
z.B. C:/Programme/I-Logix/stmm/21/preferences

### **Ausschnitt aus den Preferences für Activitycharts**

Die Farbe von:

- Activities ist DARKORANGE,
- externen Activities ist BLACK,

- Control Activities ist FORESTGREEN,
- Data Stores ist DARKORANGE,
- Dataflows ist BLUE,
- Namen ist BLACK.

Die Schriftart von:

- Namen ist 12 Punkte Courier fett,
- Labels ist 10 Punkte Courier.

Linienstärken sind:

- 1 für alle Activities, Data Stores, Dataflows.

### **Ausschnitt aus den Preferences für Statecharts**

Die Farbe von:

- States ist FORESTGREEN,
- Transitionen ist RED,
- Namen ist BLACK,
- Labels ist BLACK.

Die Schriftart von:

- Namen ist 12 Punkte Courier fett,
- Labels ist 10 Punkte Courier.

Linienstärken sind

- 1 für alle States und Transitionen.