



SEP: Implementierung eines Sequenzdiagramm-Editors für AutoFOCUS 2/AutoFLEX

Bearbeiter: Florian Hiesinger

Abgabedatum: 01.03.2004

Betreuer: Jan Romberg, Stefan Wagner

Aufgabensteller: Prof. Dr. Manfred Broy



Inhaltsverzeichnis

1	EINLEITUNG	3
1.1	Aufgabenstellung	4
1.2	Rahmenbedingungen	5
2	SYSTEMNUTZUNG	6
2.1	SSD-EET	7
2.2	DFD-EET	8
3	SYSTEMAUFBAU	10
3.1	Metamodell	10
3.2	Komponenten	13
4	AUSBLICK	16
	ANHANG	18
	Abbildungsverzeichnis	18
	Abkürzungsverzeichnis	18
	APIs (Java Doc)	18



1 Einleitung

In der Software-Entwicklung werden verstärkt grafische Modellierungssprachen wie die Unified Modelling Language (UML) eingesetzt, um Software auf einer abstrakten Ebene zu beschreiben. Durch die Verwendung von grafischen Symbolen anstatt von Text können die Übersichtlichkeit und Verständlichkeit der Software-Beschreibung stark verbessert werden. Der Entwickler zeichnet Diagramme mit Hilfe von CASE-Werkzeugen, wie z. B. Rational Rose oder Together. Diese Diagramme lassen sich idealerweise direkt in eine Programmiersprache übersetzen.

An Lehrstuhl Broy wurde das CASE-Werkzeug AutoFOCUS speziell für den Bereich der eingebetteten Systeme entwickelt. Eingebettete Systeme sind Rechnersysteme, die in andere technische Systeme eingebaut werden. So befinden sich heute in Automobilen, Flugzeugen und Waschmaschinen eine Vielzahl von Prozessoren. AutoFOCUS unterstützt verschiedene Diagrammartentypen zur Beschreibung der Systemstruktur, der Kommunikation zwischen Systemteilen und des Verhaltens von Systemteilen. Das Werkzeug ist Freeware und kann über die Web-Seite <http://autofocus.in.tum.de> geladen werden.

Im Sommersemester 2003 wurde das STP (Softwaretechnikpraktikum) AutoFLEX durchgeführt, das AutoFOCUS um die zwei Beschreibungsmittel Data Flow Diagram (DFD) und Mode Switching Diagram (MSD) erweiterte, um zusätzlichen Anforderungen gerecht zu werden.

Ein DFD besteht aus Blöcken, die Schnittstellen (Ports) besitzen, über welche sie mit den anderen Blöcken über so genannte Channels verbunden sein können (Abb. 1). Die Nachrichten zwischen den Blöcken eines DFDs werden ohne Verzögerung („immediate semantics“) ausgetauscht, d.h. sie werden gleichzeitig gesendet und empfangen. Man nutzt DFDs zur Beschreibung des Datenflusses einer Berechnung, die innerhalb eines Blocks oder einer Komponente geleistet wird. Sie können mit dem Modus eines MSDs oder mit einer Komponente/einem Block assoziiert sein.

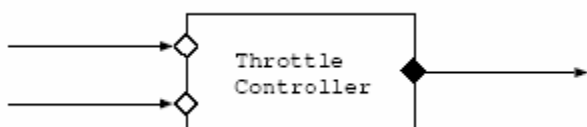


Abbildung 1: einfaches DFD

Ein MSD besteht aus Modi mit Verbindungspunkten für ausgehende/eingehende Übergänge, über die sie miteinander verknüpft sind. Mit seiner Hilfe beschreibt man den dynamischen Teil des während einer Berechnung auftretenden Datenflusses. Durch die Moduswechsel im MSD wird die Veränderung der Berechnungen, die eine Komponente/ein Block ausführt, wieder gespiegelt. Die Berechnungen sind dabei



mittels den Modi zugeordneten DFDs definiert. MSDs können mit Komponenten oder Blöcken assoziiert sein.

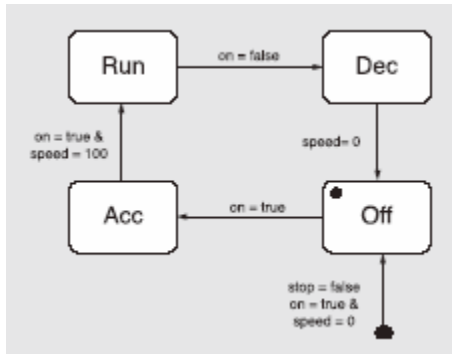


Abbildung 2: Beispiel-MSD

1.1 Aufgabenstellung

Im Rahmen des SEPs sollte ein drittes Beschreibungsmittel in AutoFOCUS 2/AutoFLEX integriert werden, um die Erweiterungen, die durch das STP AutoFLEX vorgenommen wurden, abzurunden. Neben dem DFD Editor, der Datenflussdiagramme durch Abhängigkeitsgraphen und dem MSD Editor, der Verhaltensbeschreibungen durch Transitionsgraphen darstellt, sollte ein EET (Extended Event Trace) Viewer entstehen, mit dem bestehende Ablaufsequenzen dargestellt werden können. Diese Sequenzen sind ähnlich zu MSCs (Message Sequence Chart) oder den UML Sequenzdiagrammen. Anstelle eines vollständigen EET Editors sollte ein EET Viewer, erweitert um die Konzepte aus dem STP AutoFLEX, entstehen, der aber bereits architektonisch auf einen Editoreinsatz vorbereitet ist.

Als erster Schritt sollten verschiedene Möglichkeiten der Realisierung (z.B. mit Hilfe von Frameworks) betrachtet werden und die Einarbeitung in das bestehende AutoFOCUS 2/AutoFLEX erfolgen. Als nächstes sollte für die neuen Konzepte aus dem STP AutoFLEX eine geeignete Umsetzung im AutoFOCUS 2/AutoFLEX Metamodell gefunden werden. Vor allem die neuen Konzepte MSCs für Blockdiagramme, inaktive Sequenzen in MSCs auf Blockebene und Moduswechsel in MSCs auf Komponentenebene bedingten eine Modifizierung. Außerdem war zu beachten, dass nun auch die Navigation von einem SSD-EET zu seinem zugeordneten DFD-EET unterstützt werden sollte.

Daran anschließend wurden das Design und die Implementierung des EET Viewers durchgeführt und in AutoFOCUS 2/AutoFLEX integriert. Dies umfasste insbesondere die Funktionalität der Darstellung von MSCs auf Blockebene und von Moduswechseln auf Komponentenebene, d.h. Anzeigen eines Sechsecks auf der Lebenslinie des betroffenen MSC-Threads. Ebenso die Einführung eines Aufrufmechanismus zwischen Super-MSCs und ihren Verfeinerungen (Sub-MSCs). Dafür war es erforderlich, eine Möglichkeit zu finden, um aus dem Modell die passenden Sub-MSCs zu ermitteln.



1.2 Rahmenbedingungen

Die gesamte Implementierung sollte in Java erfolgen, wobei ein Teil (Metamodell) aus Ant-Skripten generiert wurde. Der neu zu entwickelnde EET Viewer sollte sich natürlich nahtlos in die bestehende AutoFOCUS 2/AutoFLEX Softwareumgebung einfügen. Durch die langjährige Entwicklungsgeschichte des aktuellen AutoFOCUS 2/AutoFLEX gab es einige ältere Java-Quellcodebausteine, auf die zur Wiederverwendung zurückgegriffen werden konnte. Innerhalb des Projektes Quest gab es bereits einen MSC Editor, dessen Schnittstellen allerdings nicht zum AutoFOCUS 2/AutoFLEX passten. Außerdem fehlten die Funktionen zur Darstellung von Moduswechsel einer Komponente und die Verknüpfung zwischen Super-MSCs und Sub-MSCs. Die entsprechenden Anpassungen und Erweiterungen waren somit die Hauptaufgaben in der Implementierung.

Die AutoFOCUS 2/AutoFLEX Bedienoberfläche gliedert sich grob in zwei Teile:

1. Links der Projektbaum mit allen offenen Projekten und deren Modellen.
2. Rechts der Fensterbereich für die entsprechenden Editoren und Viewer der Bauelemente.

Diese beiden Teile sind relativ unabhängig voneinander behandelbar. Durch diese Gliederung ergeben sich auch die zwei wichtigsten Anforderungen, nämlich die korrekte Darstellung der im Modell abgelegten MSCs in der Baumstruktur, sowie die korrekte Darstellung und Funktion des MSC Viewers. Das Format, wie ein solches Modell aufgebaut ist, wird durch oben bereits erwähntes Metamodell definiert. Dazu wird ein UML-Klassendiagramm mit dem Werkzeug MMGen in die entsprechenden Java-Klassen umgesetzt. Automatisch werden dabei auch die nötigen Persistenzmechanismen erzeugt.

Um das Design und die Implementierung einheitlich zu belassen, war das Model-View-Controller Architekturmuster zu verwenden, welches auch den bereits vorhandenen AutoFOCUS 2/AutoFLEX Editoren zu Grunde lag. Eine weitere Anforderung war ja, Design und Realisierung des EET Viewers mit Blick auf eine spätere Erweiterung um Editor-Funktionalität zu gestalten.

Die Entwicklungsumgebung war Eclipse mit Java- und Ant- Modulen. Das Versionsmanagement erfolgte über das lehrstuhleigene CVS-Repository.



2 Systemnutzung

EETs dienen dazu das Kommunikationsverhalten zu beschreiben. Im alten AutoFOCUS gab es zur Strukturierung nur Komponenten, dementsprechend wurde nur die Interaktion zwischen den Subkomponenten einer Komponente dargestellt. Ähnlich wie bei Message Sequence Charts (MSCs) oder den Sequenz-Diagrammen in UML werden Komponenten als Boxen beschrieben, die eine senkrechte Achse als „Lebenslinie“ haben. Von diesen Achsen aus werden Nachrichten als Pfeile dargestellt. Zusätzlich gibt es den globalen Systemtakt („Tick“), der als gestrichelte, waagrechte Linie gezeichnet wird.

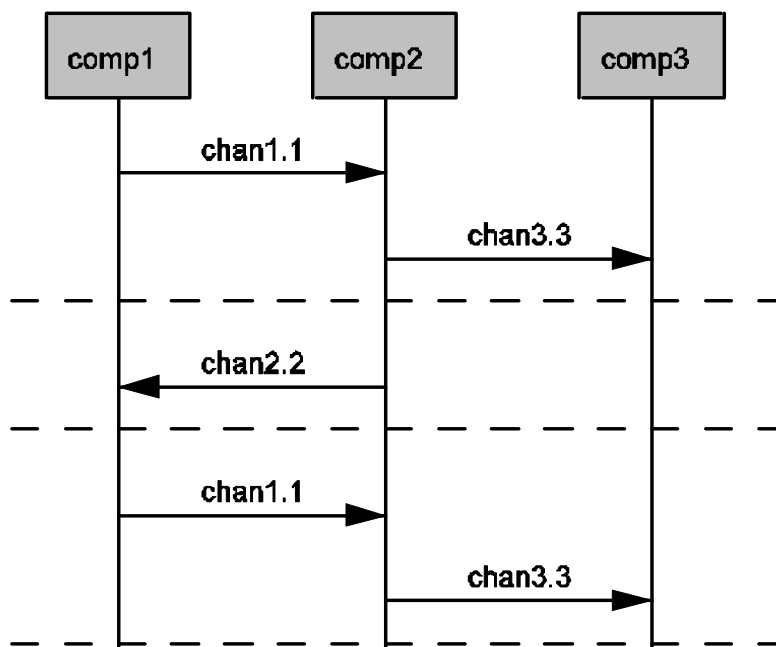


Abbildung 3: Komponenten-EET in AutoFOCUS

Die Nachrichten zwischen zwei Ticks werden, soweit keine kausalen Abhängigkeiten innerhalb des Systemschritts bestehen, als ungeordnet angesehen. Zusätzlich gab es das Konzept von „immediate Ports“, die nicht auf den nächsten Tick warten, sondern sofort durchschalten.

In AutoFOCUS 2/AutoFLEX werden nun zusätzlich Data Flow Diagrams (DFDs) eingeführt, um sofortige Berechnungen zu beschreiben. Ein DFD besteht dabei aus einem Super-Block und mehreren miteinander verbundenen Sub-Blöcken. Diese zusätzliche Strukturierung muss natürlich auch in die Beschreibungstechnik der EETs einfließen um auch Abläufe auf Blockebene darstellen zu können. Das in AutoFLEX erweiterte Konzept von AutoFOCUS, insbesondere die



hinzugekommenen Datenflussdiagramme, bedingte auch eine Erweiterung des ursprünglichen Metamodells um die neuen Konzepte *MSCMode* und *Sequence* mit einem Attribut *isInactive*. Außerdem besteht nun auch eine Assoziation zwischen Blöcken und MSC, sodass nun auch einem Block beliebig viele EETs zugeordnet sein können.

2.1 SSD-EET

Es ist möglich einen Moduswechsel der Komponente anzulegen, der als sechseckige Box im Komponenten-EET dargestellt wird. Die Semantik ist dabei so definiert, dass sich die Komponente nach dem Empfang der Eingangsnachrichten im neuen Modus befindet. Dieser Moduswechsel entspricht einer Referenz auf ein DFD-EET. Ein Klick auf die Lebenslinie der Komponente unterhalb des Moduswechsels oder ein Klick auf den Wechsel selbst startet automatisch ein neues Viewer-Fenster mit dem zugeordneten DFD-EET. Das für die Komponente noch weitere DFD-EETs hinterlegt sind, erkennt man an der dickeren Lebenslinie.

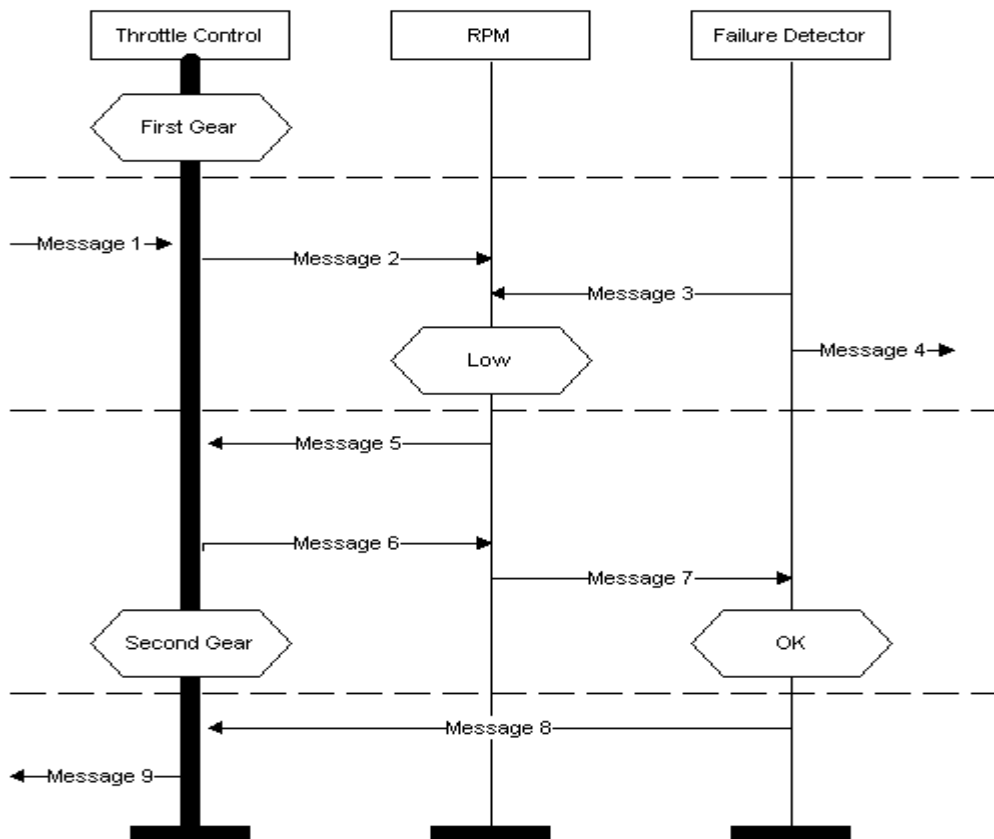


Abbildung 4: Komponenten-EET in AutoFOCUS 2/AutoFLEX



Ein Problem dabei ist, dass eigentlich in einem Systemschritt Nachrichten an die Referenz geschickt werden und im nächsten Systemschritt davon Nachrichten ausgehen. Unsere Lösung umgeht das Problem, dass in einem Systemschritt Nachrichten zur Referenz geschickt werden und im nächsten von der Referenz, indem Nachrichten auf Komponenten-Ebene nicht zu und nicht von Referenzen ausgehend geschickt werden. Stattdessen wird in Übereinstimmung mit dem ursprünglichen EET-Konzept davon ausgegangen, dass eingehende Nachrichten im aktuellen Systemschritt empfangen werden und die ausgehenden Nachrichten im nächsten Systemschritt gesendet werden. Diese gehen dann einfach von der Lebenslinie unterhalb des nächsten Ticks aus. Dabei geht die Abhängigkeit zwischen der DFD-Referenz und Nachrichten von anderen und an andere Komponenten verloren, aber sowohl sich überschneidende Pfeile, als auch eine Änderung der Darstellung der Ticks werden vermieden.

Diese Lösung ist übersichtlich, da keine Pfeile über Tick-Grenzen hinweg gezeichnet werden. Außerdem muss die Darstellung des Ticks in keiner Weise verändert werden, was bei einer Darstellung der DFD-EET-Referenz auf dem Tick nötig wäre.

Negativ ist dabei sicherlich, dass keine offensichtliche Verbindung mehr zwischen dem Versenden eines Ergebnisses einer Berechnung in einem DFD und der Referenz auf den DFD-EET vorhanden ist. Dieser Nachteil wird aber als relativ klein angesehen, da auch in den herkömmlichen EETs die Beziehung von Eingangs- und Ausgangsnachrichten nicht dargestellt wird. Um die Übersichtlichkeit noch zu erhöhen, wäre es sinnvoll zusätzlich die Nachrichten innerhalb eines Systemschritts zu ordnen, vor allem die Ausgangsnachrichten alle am Anfang zu zeichnen und danach die Eingangsnachrichten.

2.2 DFD-EET

Die Darstellung von Abläufen innerhalb von DFDs erfolgt analog zu den normalen Komponenten-EETs. Jeder Block in einem DFD wird oben als Viereck dargestellt und besitzt eine Lebenslinie, die senkrecht nach unten verläuft. Jeder Austausch von Variablen zwischen den Blöcken wird als waagrechter Pfeil zwischen den Lebenslinien dargestellt. Die Ticks der DFD-EETs sind synchronisiert mit den Ticks der Komponenten-EETs, da die Ticks ja den globalen Systemtakt angeben. Auf den Pfeilen der DFD-EETs wird der Wert der übergebenen Variablen angegeben. Am linken und rechten Rand des EETs werden bei den hereinkommenden und ausgehenden Pfeilen die entsprechenden Ports mit annotiert.

Da die DFD-EETs einem Modus der Komponente zugeordnet sind, sind die Abschnitte des DFD-EETs, in denen sich die Komponente im Super-MSC in einem anderen Modus befindet, inaktiv. Dies ist im Modell als *Sequence* mit Attribut *isInactive* gespeichert und wird grafisch durch graue Bereiche dargestellt.

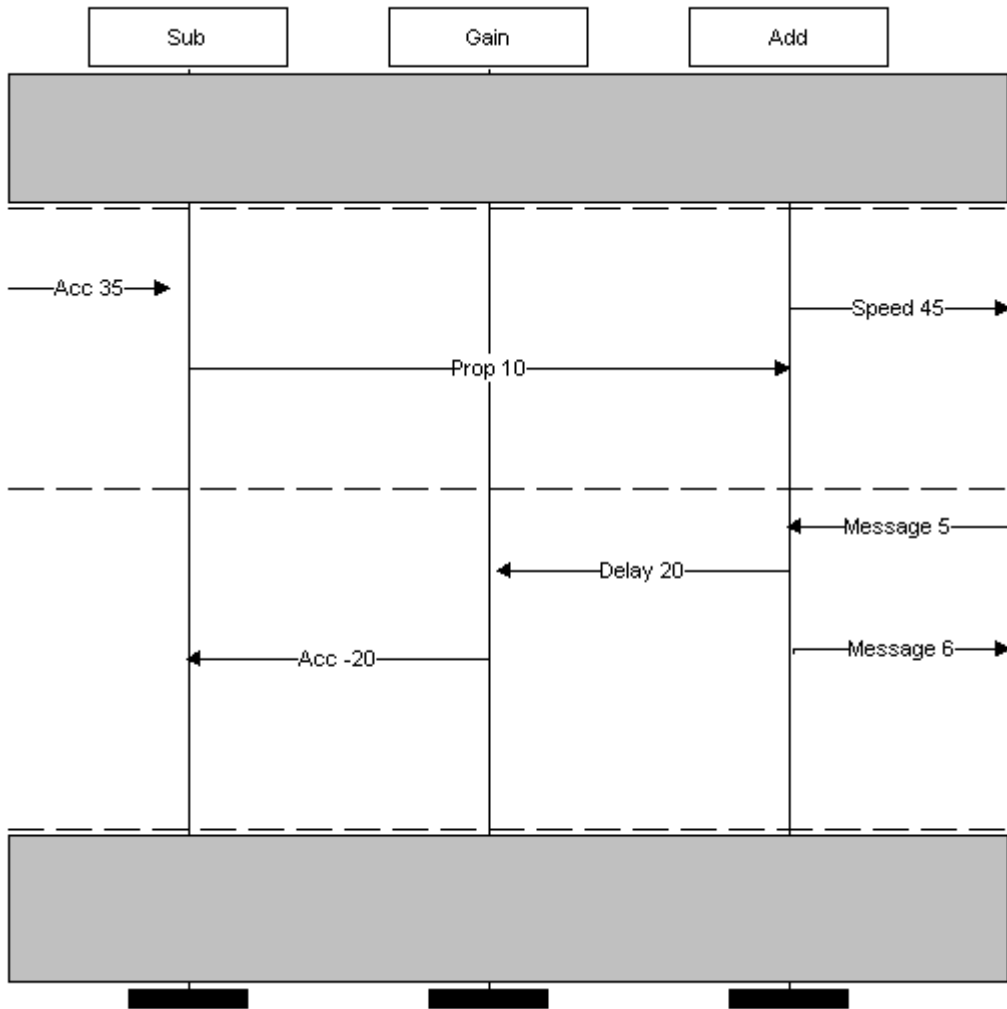


Abbildung 5: DFD-EET zum Modus First Gear des Throttle Controllers

Dabei kann eine Komponente oder ein Block nicht nur ein MSC besitzen, sondern auch eine ganze Liste von MSCs. Ein MSC beschreibt ja immer die zeitliche Abfolge von Interaktionen zwischen einer Menge von Subkomponenten oder Subblöcken innerhalb eines zeitlich begrenzten Kontextes. Zu beachten ist, dass die Zuordnung einer DFD-EET-Referenz zu einem MSC des entsprechenden DFDs eindeutig ist. D.h. für jedes SSD-EET existiert höchstens ein DFD-EET pro Komponentenmodus.



3 Systemaufbau

In diesem Kapitel beschreiben wir die Architektur und das Zusammenwirken der verschiedenen Softwarebestandteile des MSC Viewers. Dies geschieht abstrahiert von der konkreten Implementierung, um die zu Grunde liegenden Ideen und Konzepte in den Vordergrund zu rücken.

3.1 Metamodell

Ein Metamodell ist hier eine Beschreibung von Klassen und ihren Relationen untereinander, welche die möglichen Objekte und Objektbeziehungen (Modelle) zur Laufzeit des Programms charakterisiert.

AutoFOCUS 2/AutoFLEX basiert auf einem Metamodell, das die Konzepte der AutoFOCUS 2/AutoFLEX Beschreibungstechniken definiert. Das Metamodell ist als Klassendiagramm gegeben. Eine textliche Repräsentation des AutoFOCUS 2/AutoFLEX Metamodells ist im DBI Dateiformat verfügbar. Aus dieser DBI Datei lässt sich die Kernmodellkomponente von AutoFOCUS 2/AutoFLEX unter Verwendung des MMGen-Übersetzers automatisch erzeugen.

Eine detaillierte Diskussion des AutoFOCUS 2/AutoFLEX Metamodells kann man im „Quest Developers Guide“ finden. Wir beschränken uns hier auf den MSC-Teil des Metamodells.

Neu hinzugekommen ist die Klasse *MSCMode*, die für die Moduswechsel einer Komponente geschaffen wurde. Eine Komponente kann mehrere Modi haben, für die jeweils ein DFD hinterlegt sein kann. Möchte man nun so einen Moduswechsel in einem SSD-EET darstellen, benötigt man eine *MSCMode*-Instanz, der der passende Modus und die Komponente zugeordnet sind. Die Instanz kann gleichzeitig eine Referenz auf ein möglicherweise vorhandenes DFD-EET sein, das sich über den Modus und das SSD-EET ermitteln lässt. Auf diese Weise kann man zwischen den MSCs navigieren.

Die Klasse *Sequence* hat ein zusätzliches Attribut *isInactive* erhalten, um inaktive Phasen in einem DFD-EET zu kennzeichnen. Das ist immer dann der Fall, wenn wir einen Abschnitt eines DFD-EETs betrachten, dessen zugehörige Komponente sich in diesem Zeitabschnitt in einem anderen Modus befindet (siehe Abbildung 6: erweitertes MSC-Metamodell).

Da ursprünglich nur Komponenten MSCs besaßen, bestand keine Assoziation zwischen Blöcken und MSCs. Diese wurde nun eingefügt, damit auch den Blöcken MSCs zugeordnet werden können. Über diese Assoziation ist dem *MSCMode* sein Rootblock zuordenbar (Abbildung 7: MSC-Metamodell mit Block und Component).

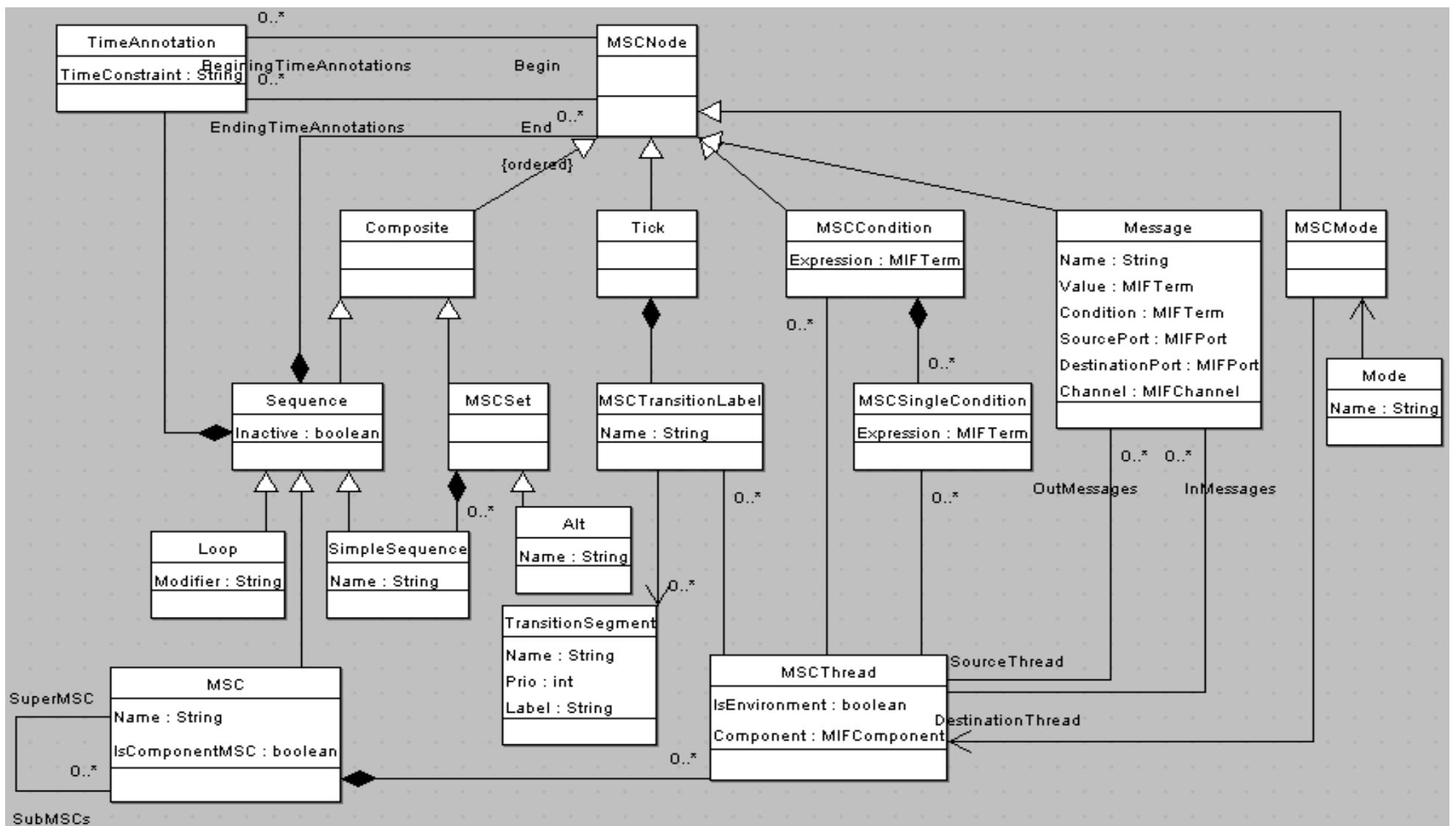


Abbildung 6: erweitertes MSC-Metamodell

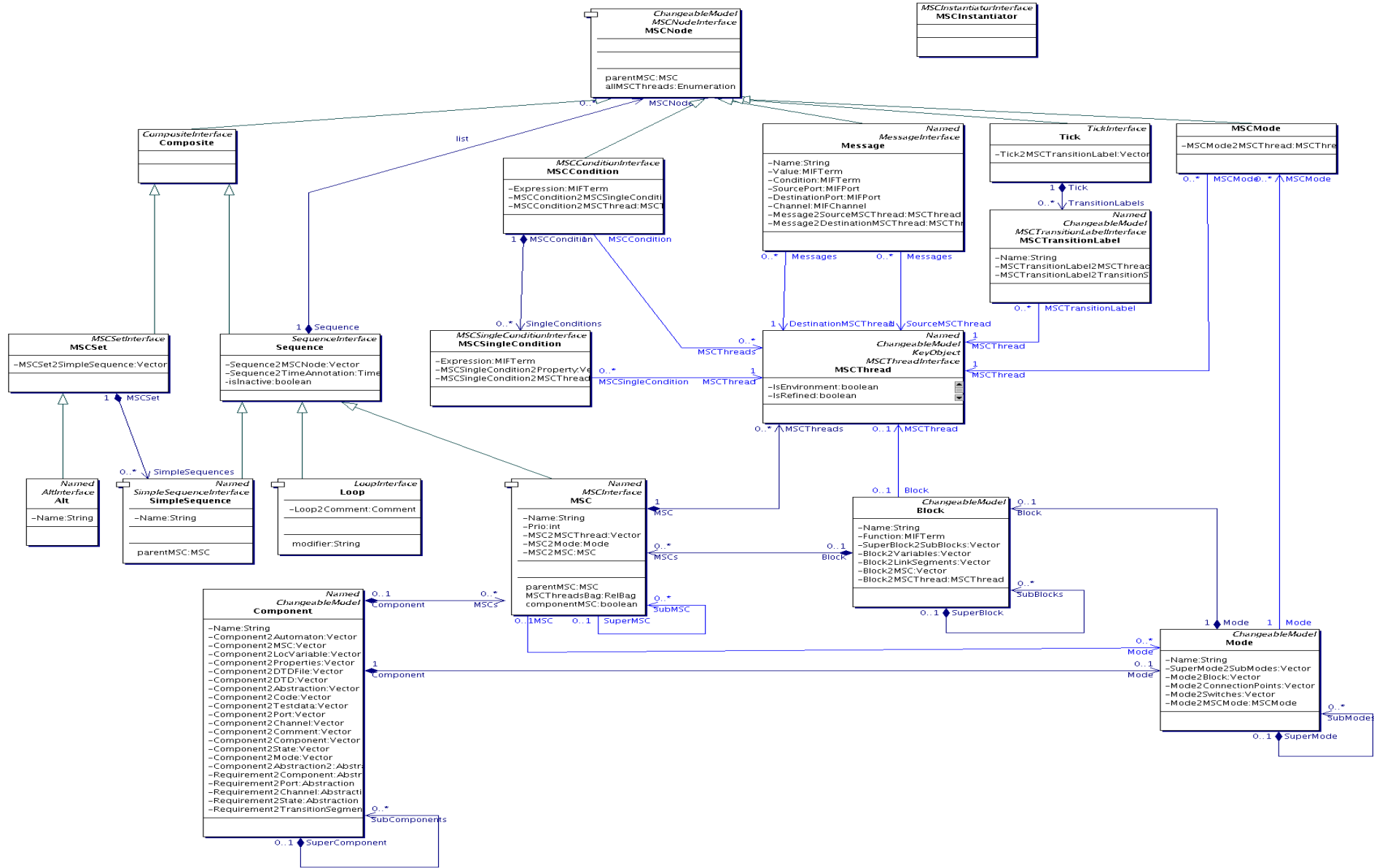


Abbildung 7: MSC-Metamodell mit Block und Component



3.2 Komponenten

AutoFOCUS 2/AutoFLEX ist durch verschiedene Java-Packages strukturiert. Wir betrachten im Folgenden auf konzeptioneller Ebene die für den MSC Viewer modifizierten oder neu hinzugekommenen Packages und Klassen.

3.2.1 de.tum.autoflex.base

Dies ist das Basis-Package, das auch die Main-Methode enthält. Sie gibt der Applikation ihren Rahmen. Hier erfolgen auch globale Konfigurationseinstellungen.

Die für uns wichtige Klasse ist der *WindowManager*. Er entspricht dem Singleton-Pattern, d.h. zur Laufzeit existiert nur eine einzige Instanz der Klasse. Beim Starten von AutoFOCUS 2/AutoFLEX wird diese Instanz erzeugt. Bei seiner Konstruktion legt er ein *MainFrame* an, in das er die ebenfalls von ihm instanziierten *MainFrame*-Buttonleiste, das *MainFrame*-Dateimenü und den *QuestBrowser*, der den Projektbaum enthält, einfügt. Den größten Platz nimmt aber der *JDesktopPane* ein, der als Container für die einzelnen Editorfenster dient.

Eine wichtige Funktionalität des *WindowManagers* ist das Anzeigen der Bearbeitungsleiste, die zum momentan aktiven Editorfenster gehört. Dazu hält er für alle möglichen Editoren die passende Leisten bereit, die er dann kontextabhängig austauscht.

Der *WindowManager* kennt alle geöffneten Editorfenster, kann sie bei Bedarf aktivieren oder neue öffnen. Außerdem bietet er über das *MainFrame*-Dateimenü die Persistenzmechanismen an.

3.2.2 de.tum.quest.vertical.ssdeditor

Dieses Package wurde aus dem AutoFOCUS 2/AutoFLEX Vorgängerprojekt Quest übernommen. Für uns ist nur die *QuestBrowser*-Klasse von Interesse. Die wichtigste Aufgabe des von *JFrame* abgeleiteten *QuestBrowsers* ist die Erzeugung und Aufbewahrung des *Browser*-Objekts.

3.2.3 quest.testing.application.browser

Die in den *MainFrame* eingebettete *Browser*-Komponente stellt die Projekte als Baumstruktur dar. Dazu nutzt sie die abgespeicherten Modellinformationen, die sie auf diese grafische Darstellung abbildet. Einen Doppelklick auf solch ein Bauelement für das ein Editor existiert, leitet sie mitsamt dem entsprechenden *TreeNode*-Objekt an den *WindowManager* weiter, der dann die weitere Verarbeitung übernimmt. So wird beispielsweise beim Anklicken eines MSC-Knotens unser MSC Viewer geöffnet.



Die hinzugekommenen Klassen *MSCTreeNode*, *InactivePeriodTreeNode* und *MSCModeTreeNode* sind von *TreeNode* abgeleitet und sorgen für Bereitstellung einer grafischen Repräsentation innerhalb des Baumes. Darüber hinaus sind die *TreeNode*-Instanzen mit ihren Modellelement-Objekten verknüpft, die ja die semantische Information tragen. So ist eine Verbindung zwischen der grafischen Baumdarstellung und dem Modell geschaffen.

3.2.4 de.tum.autoflex.editor.gui.msc

Stellt den größten Anteil des MSC Viewers. Es enthält alle Klassen zur Darstellung der DFD- und SSD-EETs im Editorfenster. Die Grundidee ist, die grafische Repräsentation eines Modells durch eine Tabellenstruktur zu bewerkstelligen. Dazu wird das als Liste abgespeicherte MSC ausgelesen und für die am Anfang der Liste stehenden *MSCThreads* Spalten angelegt. Die *MSCThreads* bilden dabei die Spaltenköpfe. Für jede der darauf nachfolgenden *MSCNodes*, wird eine Zeile angelegt und die einzelnen Zellen der Zeile den Modellinformationen entsprechend erzeugt.

Zum Beispiel wird eine Nachricht vom *MSCThread* der Spalte 1 zum *MSCThread* der Spalte 4 dazuführen, dass in der Zelle der Spalte 1 der Nachrichtenpfeil beginnt, in den Zellen der Spalten 2 und 3 der Mittelteil des Pfeils dargestellt wird und in der Zelle der Spalte 4 der Pfeil mit seiner Spitze endet. Alle anderen Zellen bleiben in dieser Zeile leer. Eine *Sequence* mit Attribut *isInactive* bewirkt dagegen in allen Zellen der aktuellen Zeile die Darstellung einer grauen Fläche.

MscEditorFrame ist der Container für die *MscTable* und veranlasst beim *WindowManager* den Austausch der Editor-Bearbeitungsleiste. Die Struktur der MSCs ist nicht flach, sondern kann beliebig stark hierarchisch strukturiert sein. Der Grund ist die Vererbung von *MSCNode* an *Sequence* und die 1-zu-n Assoziation zwischen ihnen (Abbildung 7: MSC-Metamodell mit Block und Component). Die *MscTable* ermöglicht das abflachen dieser hierarchischen MSCs. Sie ruft für das MSC auch den *MscAdapter* auf. Der *MscAdapter* ist das Bindeglied zwischen der Modellinformation und der daraus erzeugten Tabelle. Dafür gibt es für jede *MSCNode* und für jeden *MSCThread* des Metamodells eine Adapterklasse, die einen Teil der Modellinformationen des jeweiligen Elements speichert und Methoden zur Informationsabfrage bereithält. Auch wird eine *RenderFactory* erschaffen, die für jedes Modellelement einen Renderer liefert. *MscAdapter* und *RenderFactory* werden von *MscTable* genutzt, um die Spalten der Tabelle zu generieren.

Nach der Erzeugung der Spalten, werden diese zu einer Tabelle zusammengefügt und im *MscEditorFrame* angezeigt. Eine Spalte besteht dabei aus Spaltenkopf und Spalteninhalt. Die Köpfe sind, wie oben erwähnt, die *MSCThreads*. Die zugehörige grafische Darstellung wird von der *MscThreadRenderer*-Klasse implementiert. Im Anschluss daran werden die *MSCNodes* durchlaufen und so die Spalteninhalte aus den einzeln gewonnenen Zellen zusammengesetzt. Dazu wird je nach Knotentyp vom *MscNodeRenderer* die grafische Darstellung gezeichnet. Dabei nutzt er die Informationen aus den Adapterklassen.



Die Konfiguration der grafischen Umsetzung erfolgt durch die Klasse *MscEditorConfig*. Sie enthält für jedes mögliche Element eines MSCs die Festlegungen zur Darstellung dieses Elements in der Tabelle. So lassen sich beispielsweise die Höhe eines *MSCModes* und die Dicke der Lebenslinie eines *MSCThreads* verändern.

3.2.5 de.tum.autoflex.editor.controller.msc, de.tum.autoflex.editor.mode.msc

Die beiden Packages tragen momentan keine Funktionalität und dienen nur als Platzhalter. Sie wurden angelegt um der Architektur der anderen Editoren zu folgen, sind aber für den MSC Viewer nicht notwendig. In diese Packages gehören die Controller-Anteile des MVC-Patterns.



4 Ausblick

In diesem Kapitel beschreiben wir die wichtigsten Anpassungen und Erweiterungen des MSC Viewers, damit die Funktionalität eines MSC Editors erreicht wird.

Der Editor soll ja dem Model-View-Controller-Pattern folgen. Die Komponenten „Model“ und „View“ sind bereits durch unser Metamodell und den MSC Viewer abgedeckt. Es fehlt also noch die Komponente Controller, die Änderungen am Modell vornimmt und die grafische Darstellung analog verändert.

Da wir Codebausteine aus dem Questprojekt, in dem bereits ein MSC Editor implementiert war, übernahmen, existieren schon Methoden, die einen Teil der Controller-Funktionalität implementieren. Diese müssen in die Packages `de.tum.autoflex.editor.controller.msc` bzw. `de.tum.autoflex.editor.mode.msc` ausgelagert und angepasst werden. Zusätzlich ist es noch nötig, die durch das STP AutoFLEX hinzugekommenen Konzepte in der Controller Komponente zu berücksichtigen und entsprechend umzusetzen. Die dafür notwendigen Schritte beschreiben wir im Folgenden.

In der Klasse `MscTable` befindet sich Code (Methode `createActionComponent()`) zur Erzeugung von `ActionListener`, die dazu dienen Funktionalität für die Veränderung des MSCs anzubieten. So gibt es z. B. Methoden zum Einfügen von neuen Nachrichten oder Ticks. Diese Listener reagieren auf `ActionEvents` und sind verknüpft mit einer `ToolBar` und einem `EditMenu`, die solche Ereignisse auslösen können. Dazu werden die einzelnen Knöpfe der `ToolBar` und die Einträge des `EditMenus` mit den `ActionListener` verbunden. Durch die Methode `setActionEnabledness()` werden, abhängig von der angeklickten Zelle in der Tabelle, die möglichen Funktionen frei geschaltet. `ToolBar` sollte deshalb in die Klasse `MscToolBar` und `EditMenu` sollte in die Klasse `MscEditorFrame` ausgelagert werden, erweitert um die Möglichkeit zum Anlegen von `MSCModes` und inaktiven `Sequences`. Die von den `ActionListenem` aufgerufenen Methoden gehören in das Package `de.tum.autoflex.editor.controller.msc`.

Jede Methode der Klasse `MscTable` zur Behandlung eines `ActionEvents` ruft eine Partner-Methode der Klasse `MscAdapter` auf. Diese sorgt dafür, dass das Modell und die grafische Darstellung passend modifiziert werden. Dazu nutzt sie auch das Adapterobjekt des jeweiligen Modellelements. Diese Partner-Methoden müssen auch in das Controller-Package migriert werden. Der `MscAdapter` meldet sich außerdem als `ChangeListener` beim MSC Modellelement an. Veränderungen müssen aber vom Controller propagiert werden. Deswegen ist hier die Implementierung ebenfalls zu ändern.

Bei den anderen AutoFOCUS 2/AutoFLEX Editoren ist die zugehörige `ToolBar` so implementiert, dass das Anklicken eines Knopfes zum Aktivieren des passenden Modus aus dem Package `de.tum.autoflex.editor.mode.xyz` führt. Zusammen mit dem beim Auswählen einer Zelle der MSC Tabelle ausgelösten Event wird dann die geeignete Methode in der Controller Komponente aufgerufen. Dies sollte bei



MscTable genauso funktionieren. Dafür ist für jede Modifizierungsmöglichkeit des MSCs eine Modusklasse zu schreiben, die dann die richtige Methode im Package `de.tum.autoflex.editor.controller.msc` aufruft.

Zum Abschluss sind noch kleine Ergänzungen in der Klasse *Browser* und im *WindowManager* nötig, damit das Anlegen von MSCs für SSDs und DFDs möglich wird.



Anhang

Abbildungsverzeichnis

Abbildung 1: einfaches DFD	3
Abbildung 2: Beispiel-MSD	4
Abbildung 3: Komponenten-EET in AutoFOCUS	6
Abbildung 4: Komponenten-EET in AutoFOCUS 2/AutoFLEX.....	7
Abbildung 5: DFD-EET zum Modus First Gear des Throttle Controllers.....	9
Abbildung 6: erweitertes MSC-Metamodell.....	11
Abbildung 7: MSC-Metamodell mit Block und Component	12

Abkürzungsverzeichnis

CASE	Computer Aided Software Engineering
CVS	Concurrent Version System
DFD	Data Flow Diagram
EET	Extended Event Trace
MSC	Message Sequence Chart
MSD	Mode Switching Diagram
MVC	Model-View-Controller
SEP	Systementwicklungsprojekt
SSD	System Structure Diagram
STD	State Transition Diagram
STP	Softwaretechnikpraktikum
UML	Unified Modelling Language

APIs (Java Doc)

[HTML-Version](#)