Inhaltsverzeichnis

4 Konzeption und Implementierung

Dieses Kapitel beschreibt zunächst das Konzept des gesamten Aufbaus dieser Abschlussarbeit. Darauffolgend wird abschnittsweise die Umsetzung des Konzeptes erläutert.

4.1 Genereller Überblick

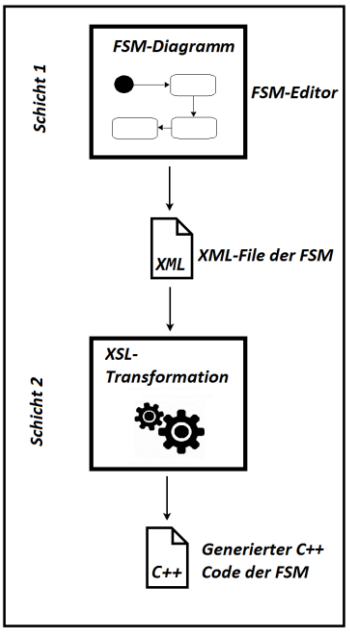

Abbildung 4.1: Ablauf des Übersetzungsschemas: Vom Zustandsdiagramm zum generierten C++-Code

Wie in Abbildung 4.1 zu sehen ist, besteht der Aufbau des Übersetzungsschemas aus mehreren Schichten die aufeinander aufbauen. Auf der ersten Schicht befindet sich der FSM-Editor, der mit Hilfe von GMF (Eclipse Foundatioin, 2011) aufgebaut wurde (siehe Abschnitt 3.3). Mit diesem steht dem Programmierer ein Werkzeug zum vereinfachten Entwurf eigener, grafischer Zustandsautomaten zur Verfügung. Das entstehende Zustandsdiagramm, was mit Hilfe vom FSM-Editor gezeichnet wurde, wird in einen XML File gespeichert. Dies enthält alle wichtigen Informationen für die grafische Darstellung der erstellten FSM. Dadurch ist der FSM-Editor von anderen Plug-ins oder Anwendungen leicht zu verwenden. In dieser Arbeit

werden diese XML-Informationen für die Codegenerierung gebraucht und somit an der zweiten Schicht übergeben (siehe Abbildung 4.1). Die zweite Schicht entspricht der XSL-Transformation (World Wide Web Consortium, 2007), die hier eingesetzt wird, um aus dem XML-File das C++-File zu generieren. Dies enthält den generierten C++-Code der FSM.

4.2 Konzept des GMF-Editors

4.3 Implementierung des GMF-Editors

4.4 Konzeption der XSL-Transformation

Wie in Abbildung 4.1 zu sehen ist, wird in der zweiten Schicht ein verfahren benötigt, um aus dem XML-File, den C++-Code zu generieren. Für diesen Zweck wird die XSL-Transformation, kurz XSLT, eingesetzt (World Wide Web Consortium, 2007).

XSLT

„Die Abkürzung für XSLT steht für eXtendable Stylesheet Language Transformation. XSLT gehört zu den deklarativen Sprachen und beschreibt und steuert die Umwandlung (Transformation) von XML-Dokumenten in andere Formate wie HTML, XHTML, Text oder andere XML-Strukturen.“ (Bongers, 2004 S. 26)

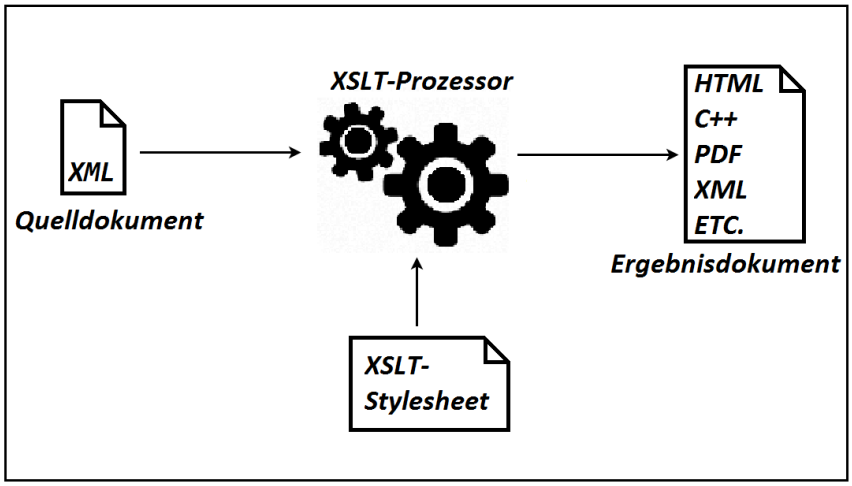

Abbildung 4.2: Schematische Darstellung einer Verarbeitung mit XSLT

Wie in Abbildung 4.2 zu sehen ist, beteiligen sich in jeder Transformation drei Dokumente.

1. **Quelldokument**

   Das Quelldokument ist ein XML-File, das alle Informationen des FSM-Diagramms umfasst und für die Ausführung einer XSL-Transformation notwendig ist (vgl. Bongers, 2004, S.26)

2. **XSLT-Stylesheet**

   Dieses Dokument definiert die Umwandlungsregeln und ist für die Ausführung einer Transformation auch erforderlich. Das Stylesheet wird von dem XSLT-Prozessor eingelesen (siehe Abbildung 4.2) und anhand der Umwandlungsregeln wird aus ein

oder mehrere XML-Dokumenten das gewünschte Ausgabeformat erzeugt (vgl. Bongers, 2004 S. 28).

3. **Ergebnisdokument**

   Das Ergebnisdokument lässt sich aus dem Quelldokument und XSLT-Stylesheet erzeugen (ebd., S. 28, 29).

Einleitung der XSLT Elemente

Wie im vorigen Abschnitt „**XSLT“** schon erwähnt, beinhaltet das XSLT-Stylesheet eine Menge von Umwandlungsregeln. Diese werden in der Stylesheet-Hierarchie nach drei unterschiedlichen Kategorien geordnet (ebd., S. 32):

- **Wurzelelemente:**

  Die einzigen beiden XSLT-Elemente, die als Wurzelelement eingesetzt werden können, sind xsl:stylesheet oder xsl:transform, in denen die XSLT-Version festgelegt werden muss.

- **Toplevel-Elemente:**

  Direkte Kinderelemente des Wurzelelementes werden Toplevel-Elemente genannt. Diese definieren den globalen Aufbau des Stylesheets (wie z.B. Funktionsköpfe, Schablonen (Templates), Formatierungsregeln usw.).

- **Instruktionen:**

  Diese Elemente sind den Toplevel-Elementen untergeordnet (Schleifen, Bedingungen, Templatesaufrufe usw.).

Die ausführliche Beschreibung der XSLT-Funktionsweise, wird im Abschnitt „**Die Transformation“** auf Seite 13 explizit erklärt. Es wird gezeigt, wie anhand von den Oben genannten XSLT-Elementen, der C++-Code generiert wird.

4.5 XSL-Transformation Implementierung

Nachdem die Implementierung des FSM-Editors beschrieben wurden ist (siehe Abschnitt 3.3), folgt die Beschreibung der Umsetzung der XSL-Transformation. Diese ist in der Lage einen XML-File einzulesen, um daraus den zugehörigen C++-Code zu generieren. Zuerst wird anhand eines FSM-Diagramms des FSM-Editors (siehe Abbildung 4.4) gezeigt wie ein zugehöriges XML-File aussieht (siehe Listing 4.1). Anschließend wird eine kleine Transformation durchgeführt und der dazugehörige generierte C++-Code gezeigt.

Beispiel

Als Beispiel betrachten wir ein OMNeT++-Netzwerk, der aus zwei Knoten („Tic“ und „Toc“) besteht (siehe Abbildung 4.3). Diese sind mit einer bidirektionalen Kommunikations-verbindung miteinander verbunden. Einer der beiden Knoten „Tic“ erstellt ein Paket „transferredMsg“ und sendet dies an Toc weiter. Toc modifiziert das Paket und sendet es an Tic wieder zurück usw.

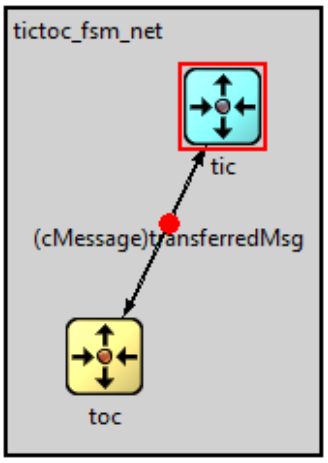

Abbildung 4.3: Beispiel:-TicToc-Netzwerk

FSM-Diagramm des FSM-Editors

Ab diesem Abschnitt wird nur noch der Knoten Tic betrachtet. Zuerst werden mit Hilfe vom FSM-Editor alle möglichen Zustände für diesen Knoten gezeichnet (siehe Abbildung 4.4).

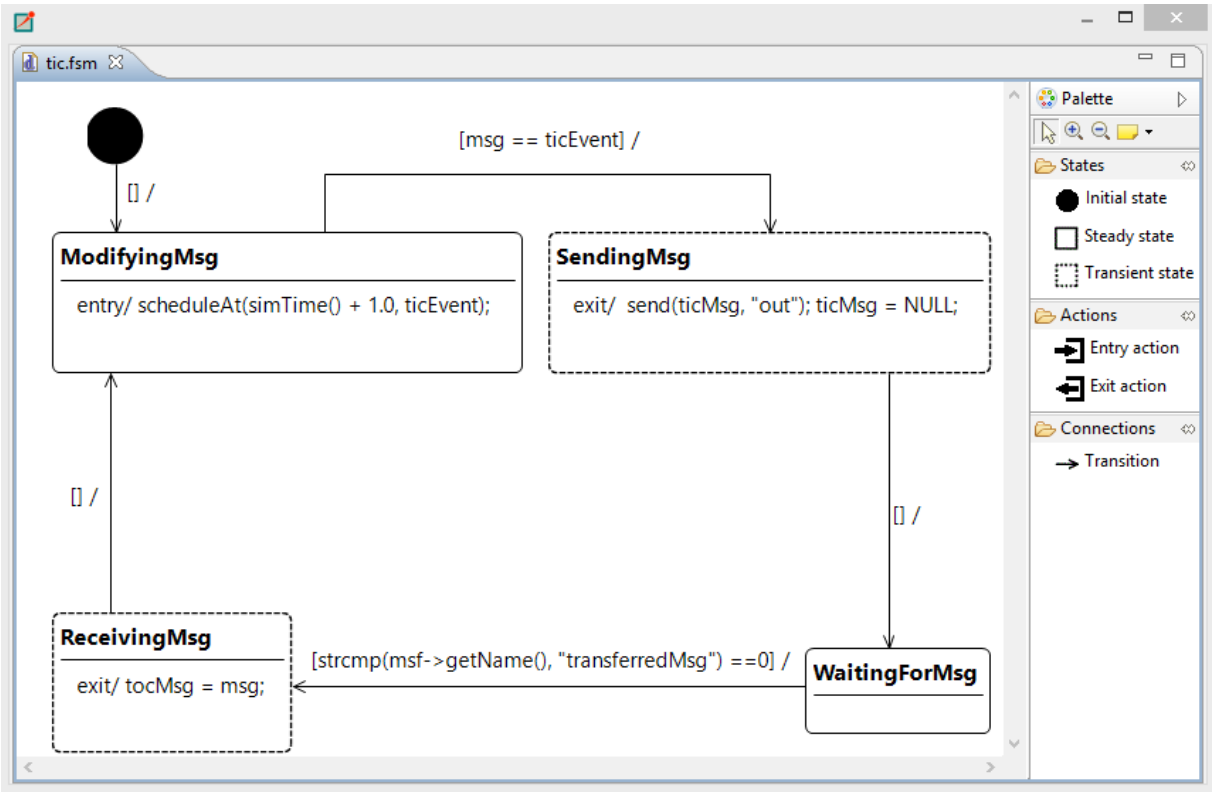

Abbildung 4.4: Tic-FSM mit dem FSM-Editor gezeichnet

Wie in Abbildung 4.4 zu sehen ist, kann sich Tic in fünf verschiedene Zustände befinden:

1. INIT (Initialstate)
2. ModifyingMsg (Steady State)
3. SendingMsg (Transient State)
4. WaitinForMsg (Steaty State)
5. ReceivingMsg (Transient State)

Beim Eintreten in dem Startzustand „ModifyingMsg“, soll ein Timer „ticEvent“ gesetzt werden. Beim Ablauf des Timers wechselt die FSM in dem „SendingMsg“ Zustand. Von dem „SendingMsg“ Zustand wechselt die FSM ohne eine Bedingung (Guard) in dem nächsten Zustand „WaitingForMsg“, dabei wird der Exit-Code ausgeführt, der das Paket

„transferredMsg“ an Toc versendet. Danach wartet die FSM in dem „WaitingForMsg“ Zustand so lange bis, das Paket „transferredMsg“ von Toc wieder ankommt. Erst dann wechselt die FSM in dem „ReceivingMsg“ Zustand. Nachdem das Paket erhalten wurde, wechselt die FSM wieder in dem Startzustand und wiederholt den vorherigen Ablauf.

XML-Code des FSM-Diagramms

Im vorigen Abschnitt wurde das FSM-Diagramm des Knotens Tic mit dem FSM-Editor gezeichnet. Dabei hinterlegt der FSM-Editor ein XML-File, dessen Aufbau in diesem Abschnitt erläutert wird.

```
<?xml version="1.0" encoding="UTF-8"?>

  <!-- Wurzelknoten. Enthaelt alle Zustaende (state, initalstate) als Kinderknoten -->
  <fsm:FSM xmi:id="_HXV">


     <!-- ModifyingMsg-Zustand vom Typ Steady -->
     <state xmi:type="fsm:SteadyState" xmi:id="_r0X" name="ModifyingMsg">
      <outTrans xmi:type="fsm:Transition" xmi:id="_MY4" Guard="msg == ticEvent"
                Effect="" target="_tKj" source="_r0X"/>
      <entry xmi:type="fsm:Action" xmi:id="_F8h"
             entryLabel="scheduleAt(simTime() + 1.0, ticEvent);"/>
    </state>

    <!-- ReceivingMsg-Zustand vom Typ Transient -->
    <state xmi:type="fsm:TransientState" xmi:id="_szm" name="ReceivingMsg">
      <outTrans xmi:type="fsm:Transition" xmi:id="_B4m" target="_r0X" source="_szm"/>
      <exit xmi:type="fsm:eAction" xmi:id="_NEf" exitLabel="tocMsg = msg;"/>
    </state>

    <!-- SendingMsg-Zustand vom Typ Transient -->
    <state xmi:type="fsm:TransientState" xmi:id="_tKj" name="SendingMsg">
      <outTrans xmi:type="fsm:Transition" xmi:id="_ipM" target="_f8R" source="_tKj"/>
      <exit xmi:type="fsm:eAction" xmi:id="_OiL"
            exitLabel=" send(ticMsg, &amp;quot;out&amp;quot;); ticMsg = NULL;"/>
    </state>

    <!-- WaitingForMsg-Zustand vom Typ Steady -->
    <state xmi:type="fsm:SteadyState" xmi:id="_f8R" name="WaitingForMsg">
      <outTrans xmi:type="fsm:Transition" xmi:id="_mYD"
                Guard="strcmp(msf->getName(), &amp;quot;transferredMsg&amp;quot;) ==0"
                Effect="" target="_szm" source="_f8R"/>
    </state>

    <!-- Initalzustand, zeigt auf dem Startzustand mit der ID "_r0X" (ModifyingMsg)-->
    <intialState xmi:type="fsm:InitialState" xmi:id="_rWa">
      <outTrans xmi:type="fsm:Transition" xmi:id="__Sb7" target="_r0X" source="_rWa"/>
    </intialState>

  </fsm:FSM>
```

Listing 4.1: XML-Code des FSM-Diagramms "Tic"

Jedes XML-File beginnt mit einem Wurzelknoten. Wie in Listing 4.1 zu sehen ist, entspricht es in diesem Fall der `<fsm:FSM>` Knoten. Aus der Sicht des Knoten `<state>` bzw. `<initialState>` ist der Knoten `<fsm:FSM>` der Elternknoten. Diese können wiederum untergeordnete Knoten (Attribute bzw. Elemente) haben. Da das XML-File auch Daten hält, die für die Transformation nicht relevant sind, werden die transformationsrelevanten Daten in Abbildung 4.5 als eine Baumstruktur verständlicher dargestellt.

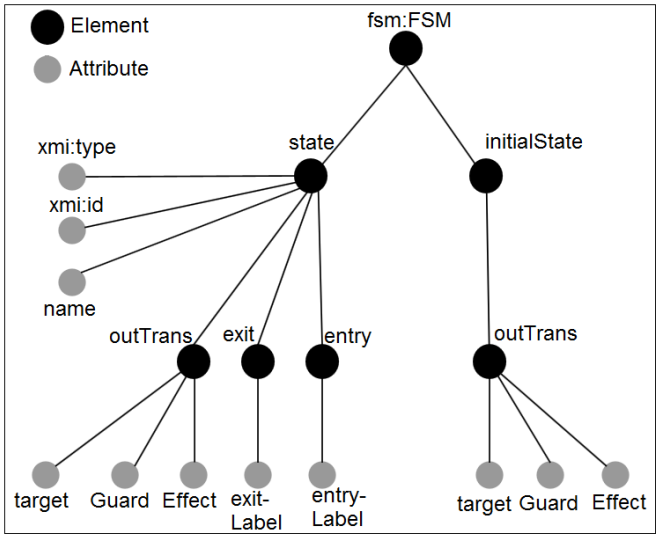

Abbildung 4.5: "Tic" XML-File als Baumstruktur

Der Knoten `<state>` hat sechs untergeordnete, transformationsrelevante Knoten (Attribute bzw. Elemente), nämlich:

- Attribut `<xmi:type>`: Um welchen Zustandstyp es sich handelt (Steady, Transient).
- Attribut `<xmi:id>`: Eine eindeutige generierte ID für jeden Zustand.
- Attribut `<name>`: Zustandsname.
- Element `<outTrans>` das wiederum drei relevante Attribute hat:
  - Attribut `<target>`: ID des Ziel-Zustandes.
  - Attribut `<Effect>`: Übergangseffekt.
  - Attribut `<Guard>`: Eine Transition kann nur durchlaufen werden, wenn der Wächterausdruck „Guard“ wahr ist.
- Element `<entry>` der wiederum ein relevantes Attribut hat:
  - Attribut `<entryLabel>`: Aktionen die beim Eintreten eines Zustandes stattfinden.
- Element `<exit>` der auch ein relevantes Attribut hat:
  - Attribut `<exitLabel>`: Aktionen die beim Verlassen eines Zustandes stattfinden.

Bei einem `<intialState>` Knoten sind die ausgehenden Transitionen und deren Attribute das einzig relevante für die XSL-Transformation (siehe Abbildung 4.5).

Die Transformation

Nachdem der Aufbau eines XML-Files im vorigen Abschnitt erläutert wurde, wird jetzt eine kleine Transformation durchgeführt. Diese zeigen wie der C++-Code im Zieldokument anhand der Umwandlungsregeln des XSLT-Stylesheets generiert wird. Für die Transformation wird ein Codeausschnitt aus dem XSLT-Stylesheet verwendet (siehe Listing 4.2). Dieser generiert den C++-Code, der für die Initialisierung der FSM zuständig ist. Dafür muss aus dem XML–File folgendes ausgelesen werden:

- Welche Zustände vorhanden sind
- Welche Typen diese Zustände haben
- Welche Position diese Zustände haben

Wie im Abschnitt „**Einleitung in der XSLT Elemente“** schon beschrieben wurde, enthält das XSLT-Stylesheet verschiedene Elemente. In Listing 4.2 Zeile 1 wird das Element <xsl:template> verwendet. Nach *Bongers* wird das wie folgt beschrieben: *„Die Deklaration xsl.template dient zur Definition einer Templateregel bzw. eines benannten Templates. Mit Hilfe des in diesem Template enthaltenen Sequenzkonstrukutors kann in Zusammenhang mit einem verarbeiteten Node ein beliebiger Output in einen Ergebnisbaum erzeugt werden.“* (Bongers, 2004 S. 863)

In diesem Fall wird mit der Anweisung `<xsl:template match="fsm:FSM">` der Würzelknoten `<fsm:FSM>` durch das definierte Template verarbeitet.

Text innerhalb von Templates wird vom XSLT-Prozessor in das Zieldokument übernommen, das heißt innerhalb des Templates `<xsl:template match="fsm:FSM">` wird der schwarzfarbiger Code in das Zieldokument geschrieben, wie z.B. in Zeile 2.

Mit der Anweisung `<xsl:for-each select="state[@xmi:type='fsm:SteadyState']">` (siehe Listing 4.2, Zeile 5) wird auf jeden „state“ Knoten, der vom Typ Steady ist, der Rumpf dieser for-each Schleife (Zeile 6) angewendet. Der Rumpf Schreibt den Namen und Position des Zustandes in das Zieldokument rein (siehe Listing 4.3, Zeile E und F). Für die „state“ Knoten vom Typ Transient werden die Zeilen G und H auf derselben Art und Weise in das Zieldokument geschrieben.

```
1.  <xsl:template match="fsm:FSM">
2.  cFSM fsm;
3.  enum {
4.     INIT = 0,
5.     <xsl:for-each  select="state[@xmi:type='fsm:SteadyState']">
6.     <xsl:value-of select="@name"/> = FSM_Steady(<xsl:value-of select="position()"/>),
7.     </xsl:for-each>
8.     <xsl:for-each  select="state[@xmi:type='fsm:TransientState']">
9.     <xsl:value-of select="@name"/> = FSM_Transient(<xsl:value-of select="position()"/>),
10.    </xsl:for-each>
11. };
12. </xsl:template>
```

Listing 4.2: Codeausschnitt aus dem XSLT-Stylesheet

```
A.  cFSM fsm;
B.  enum {
C.      INIT = 0,
D.      //Zustandsname = FSM_Typ(Position)
E.      ModifyingMsg = FSM_Steady(1),
F.      WaitingForMsg = FSM_Steady(2),
G.      ReceivingMsg = FSM_Transient(1),
H.      SendingMsg = FSM_Transient(2),
I.  };
```

Listing 4.3: Generierter Codeausschnitt aus dem Zieldokument (C++-File)

Literaturverzeichnis

Bongers, Frank. 2004. *XSLT 2.0.* Bonn : Galileo Press GmbH, 2004. ISBN 3-89842-361-1.

Eclipse Foundatioin. 2011. Graphical Modeling Framework. [Online] 2011. https://www.eclipse.org/gmf-tooling/.

World Wide Web Consortium. 2007. *XSL Transformations (XSLT) Version 2.0.* [Online] W3C Recommendation, 2007. [Zitat vom: 23. 10 2015.] http://www.w3.org/TR/xslt20/.

Abbildungsverzeichnis

Quellcodeverzeichnis