4.3. XSLT

Dieses Kapitel beschreibt zunächst das Übersetzungsschema, wie man mit Hilfe der XSL-Transformation (Extensible Stylesheet Language), kurz XSLT, von der FSM-Grafik zu dem FSM-Source-Code kommt. Zu Beginn wird die Struktur des Übersetzungsablaufs gezeigt. Darauf wird eine Einführung in die Transformationssprache (XSLT) gegeben. Zum Schluss wird mit Hilfe eines Beispiels erläutert, wie anhand von XSLT-Befehle, die FSM_Switch() aufgebaut bzw. generiert wird.

4.3.1. Übersetzungsablauf

Das dynamische Verhalten von den einzelnen Komponenten ist vorher mit dem FSM-Editor vom User zu definieren. Wie man in Abbildung 1 sieht ist der FSM-Editor ein Multi-Page-Editor der zwei verschiedene Seiten hat, die unten durch Tabs selektiert werden können. Die erste Seite „Design“ ist für die grafische Modellierung von State Machines zuständig. Auf der zweiten Seite „Source“ wird Source-Code passend zum Design generiert.

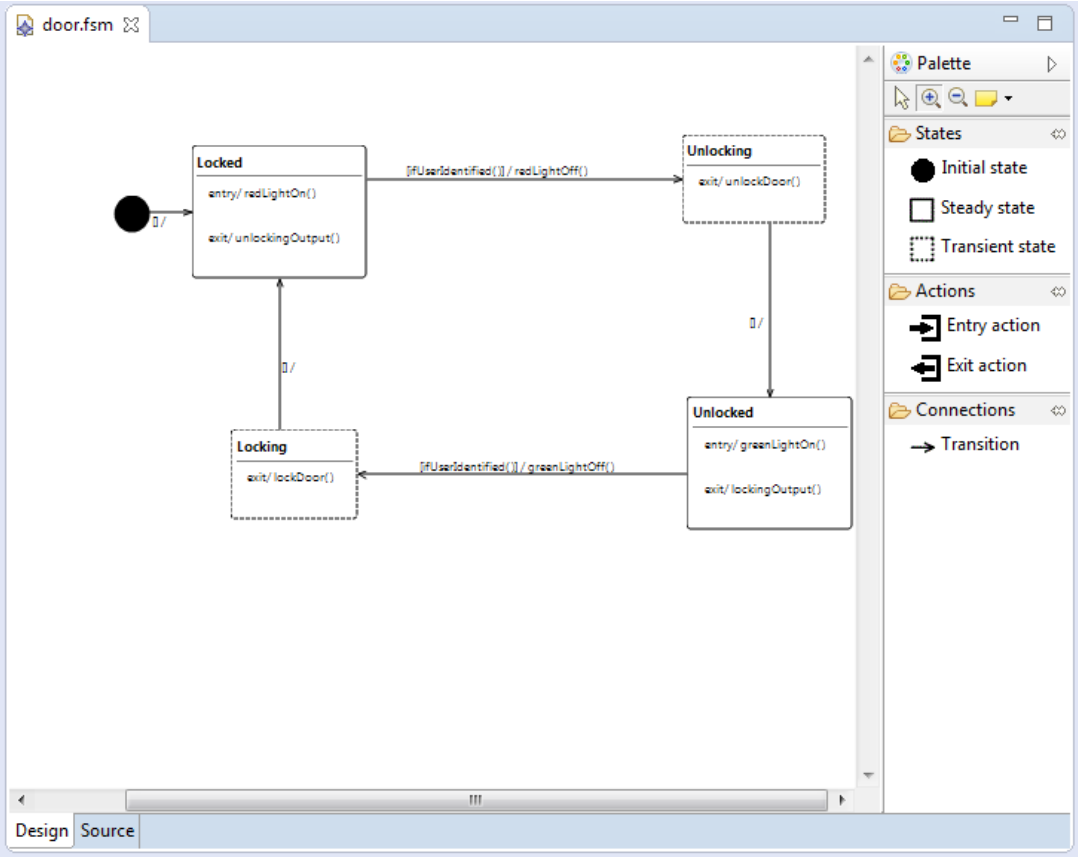
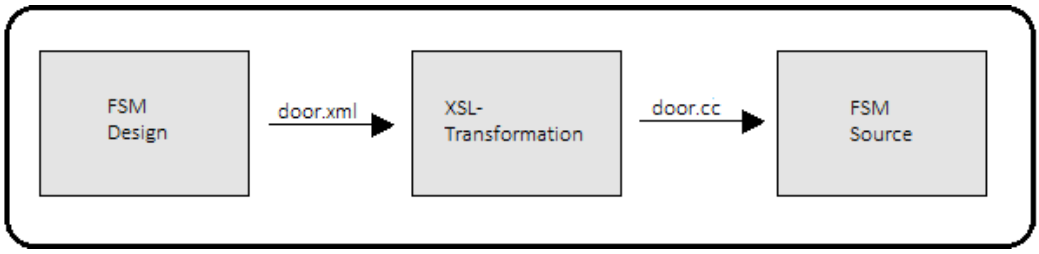


Abbildung 1: FSM Multi-Page-Editor

[1] **XSL** ist eine in XML notierte Familie von Transformationssprachen zur Definition von Layouts für XML-Dokumente.

Im Folgenden wird die Realisierung der Präsentationsschicht des FSM-Multi-Page-Editor, der als fundamentalen Technologien XML und XSLT nutzt.
Der FSM-Editor Plug-In erzeugt die zwei seitige(Design, Source) .fsm Datei. Diese Datei basiert auf XML und enthält alle domänenspezifischen Aspekte sowie alle Informationen für die graphische Darstellung der erstellten FSM. Dadurch leicht zu parsen und auch von anderen Plug-Ins oder Anwendungen zu verwenden. In diesem Fall werden diese XML-Informationen bei der XSL-Transformation gebraucht.
Die XSL-Transformation wird hier eingesetzt um die Umwandlung einer XML-Datei "door.xml", in einer C++ Datei „door.cc" auszuführen. Das resultierende Dokument „door.cc" entspricht die zweite Seite des Multi-Page-Editors „Source".



Abblidung xx2: Übersetzungsablauf

4.3.2. XSLT-Einführung

Die XSL-Transformation ist eine flexible, leistungsfähige Programmiersprache, um dynamisch aus XML-Datenbeständen Teile zu extrahieren und sie anderen Anwendungen zur Verfügung zu stellen und sie dabei auch zugleich in neue Formate zu überführen. Sie stellt eine vollständige Sprache dar und ist ein Teil der Extensible Stylesheet[2] Language (XSL). Diese XSL-Transformation dient zur Definition von Umwandlungsregeln, die auf der logischen Baumstruktur eines XML-Dokumentes aufbauen.
XSLT-Programme, sogenannte XSLT-Stylesheets, sind nach den XML-Standards aufgebaut. Diese Stylesheets werden von spezieller Software (XSLT-Prozessoren) eingelesen um ein oder mehrere XML-Dokumente in das gewünschte Ausgabeformat umzuwandeln.

[2]**Stylesheet-Sprachen** sind formale Sprachen in der Informationstechnik, um das Erscheinungsbild von Dokumenten bzw. Benutzeroberflächen festzulegen.

4.3.3. XSLT-Funktionsweise

Um die Transformation durchzuführen, werden die XML-Dokumente als logischer Baum betrachtet: Die Quellbäume des XML-Quelldokumentes, der transformiert wird und die durch die Transformation entstehenden Zielbäume des Zieldokumentes.

Eine XSL-Transformation besteht aus eine Menge von Elemente. Das Element `<xsl:stylesheet>` ist das Wurzelelement eines XSL-Dokumentes. Zusätzlich ist es die Plattform für allgemein gültige Namensräume und Attribute. Die Kinderelemente des Wurzelelements werden Top-Level-Elemente genannt. Die wichtigsten dabei sind `<xsl:template>` und `<xsl:apply>`.
Mit dem Top-Level-Element `<xsl:template>` wird die Ausgabe von Daten gesteuert. Dies kann in zwei verschiedenen Wegen definiert bzw. aktiviert werden. Entweder durch Vergleich von Knoten mit Suchpattern.

```
<xsl:template   match=”KnotenName”>
      template-body
</xsl:template>
```

Oder durch expliziten Aufruf mit Hilfe eines Namens, den man selber an diesem Template zugewiesen hat. Auf diese Art und Weise werden Methoden definiert.

```
<xsl:template name=”zugewiesenerTemplateName”
      template-body
</xsl:template>
```

Der Methoden Aufruf würde dann so aussehen:

```
<xsl:call-template name=”zugewiesenerTemplateName”/>
```

Die Anweisung `<xsl:apply-templates>` wählt eine Menge von Knoten zur Bearbeitung aus. Sie veranlasst den XSLT-Prozess nach einem passenden Template für die Verarbeitung der Knoten zu suchen.

```
<xsl:apply-templates select=”KnotenName”/>
```

4.3.4. Beispiel

In diesem Abschnitt werden wir einige Funktionen von XSLT behandeln und sie anhand eines praktischen Beispiels besprechen. Um zu zeigen wie man von dem XML-Code auf die generierten FSMSwitch() in der CPP-Datei kommt.

Das Beispiel an dem wir die XSLT Funktionsweise besprechen, ist eine Tür „door“ die sich in fünf Zuständen befinden kann:

1. INIT (Initial State)
2. Locked (Steady State)
3. Unlocking (Transient State)
4. Unlocked (Steady State)
5. Locking (Transient State)

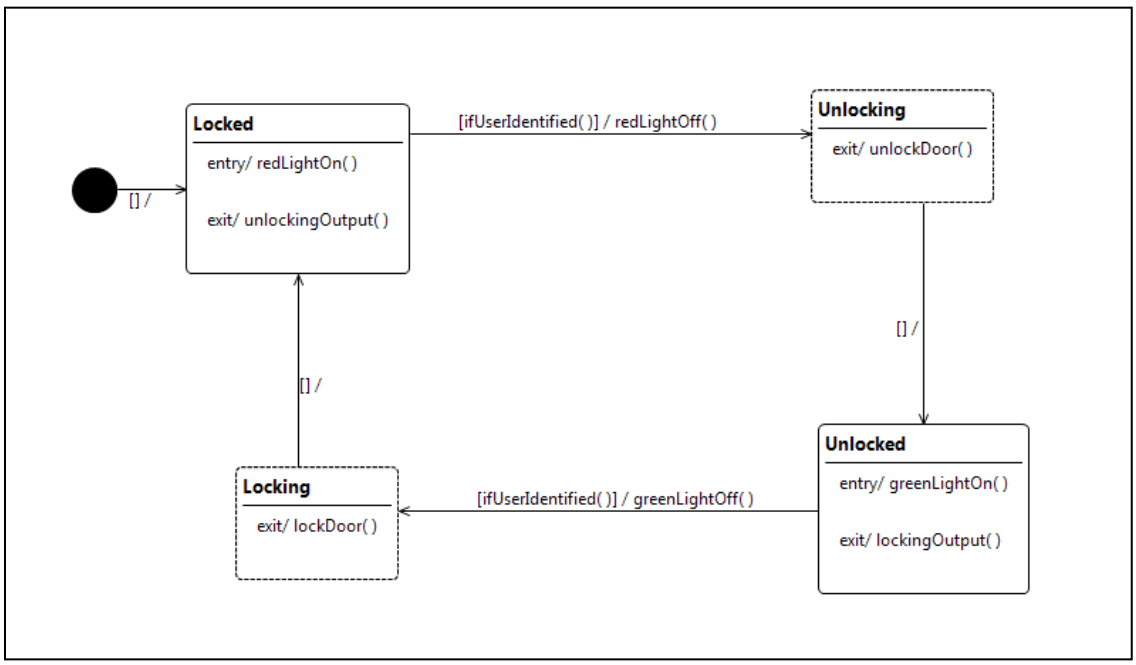


Abbildung 2: Die Datei door.fsm „Design Page“

Beim Eintreten in den Startzustand „Locked“, soll die LED-Lampe Rot Leuchten, um den User Bescheid zu geben, dass sich die Tür an diesem Zeitpunkt in einem geschlossenen Zustand befindet. Somit muss sich der User Identifizieren, was auch als Bedingung „Guard“ an der Transition steht, um die Tür zu entschließen. Der Wechselvorgang in dem „Unlocking“-Zustand findet erst bei einer erfolgreichen Identifizierung statt. Zusätzlich soll beim Entlassen von dem Zustand „Locked“, eine Ausgabe stattfinden, dass die Tür entschlossen wird und so weiter.

Nachdem die XSL-Transformation ausgeführt wurde, wird auf der Source-Page des FSM-Multi-Page-Editors das folgende generiert:

- Die Initialisierung der verschieden Zustände mit den unterschiedlichen Typen (INIT, Transient, Steady)
- Die FSM_Switch-case, die die Semantik des Automaten auf der vorigen Seite beschreibt
- Weitere Code-Generierungen (Konstruktor, Deskruktor, usw.)

```
     // FSM and its states
    cFSM fsm;
    enum {
            INIT = 0,
            Locked = FSM_Steady(1),
            Unlocked = FSM_Steady(2),
            Unlocking = FSM_Transient(1),
            Locking = FSM_Transient(2),

          };
.
.
.

FSM_Switch(fsm){

            case FSM_Exit(INIT):
                                FSM_Goto(fsm, Locked);
                                break;

            case FSM_Enter(Locked):
                                //entry code
                                redLightOn();
                                break;

            case FSM_Exit(Locked):
                                //exit code
                                unlockingOutput();
                                if(ifUserIdentified()){
                                  //effect
                                  redLightOff();
                                  FSM_Goto(fsm, Unlocking);
                                }
                                break;

            case FSM_Enter(Unlocked):
                                //entry code
                                greenLightOn();
                                break;

            case FSM_Exit(Unlocked):
                                //exit code
                                lockingOutput();
                                if(ifUserIdentified()){
                                  //effect
                                  greenLightOff();
                                  FSM_Goto(fsm, Locking);
                                }
                                break;

            case FSM_Enter(Unlocking):

                                break;

            case FSM_Exit(Unlocking):
                                //exit code
                                unlockDoor();
                                FSM_Goto(fsm, Unlocked);
                                break;

            case FSM_Enter(Locking):

                                break;

            case FSM_Exit(Locking):
                                //exit code
                                lockDoor();
                                FSM_Goto(fsm, Locked);
                                break;


    }
}
```

Abbildung 3: Die Datei door.cc „Source Page“

Der XML-Code der zum FSM-Design gehört, sieht wie folgt aus:

```
<fsm:FSM xmi:id="_V_GRQGg2EeWRzPFirv0mnA">

   <state xmi:type="fsm:SteadyState" xmi:id="_ZBy" name="Locked">
     <outTrans xmi:id="_ioCK" Guard="ifUserIdentified( )"
        Effect="redLightOff( )" target="_aDs" source="_ZBy"/>
     <entry entryLabel="redLightOn( )"/>
     <exit exitLabel="unlockingOutput( )"/>
   </state>

   <state xmi:type="fsm:SteadyState" xmi:id="_Zpi" name="Unlocked">
     <outTrans xmi:id="_j0u" Guard="ifUserIdentified( )" Effect="greenLightOff( )"
        target="_ajj " source="_Zpi"/>
     <entry xmi:id="_iC9" entryLabel="greenLightOn( )"/>
     <exit xmi:id="_s-ZaY" exitLabel="lockingOutput( )"/>
   </state>

   <state xmi:type="fsm:TransientState" xmi:id="_aDs" name="Unlocking">
     <outTrans xmi:id="_jVu" Guard="" Effect="" target="_Zpi" source="_aDs"/>
     <exit xmi:id="_vWr" exitLabel="unlockDoor( )"/>
   </state>
   <state xmi:type="fsm:TransientState" xmi:id="_ajj " name="Locking">
     <outTrans xmi:id="_kWT" Guard="" Effect="" target="_ZBy" source="_ajj "/>
     <exit " xmi:id="_d6d" exitLabel="lockDoor( )"/>
   </state>

   <intialState xmi:type="fsm:InitialState" xmi:id="_X7N">
     <outTrans xmi:id="_hlE" target="_ZBy" source="_X7N"/>
   </intialState>

 </fsm:FSM>
```

Wie man sieht, beinhaltet die XML-Datei Informationen, die für die XSL-Transformation wichtig sind und sich als Baumstruktur begreifen lassen. Jedes XML-Dokument beginnt mit einem Wurzelknoten. In diesem Fall ist <fsm:FSM> der Wurzelknoten. Aus Sicht des Knotens <State> bzw. <initialState> ist der Knoten <fsm:FSM> der Elternknoten. Der Knoten <State> kann sechs untergeordnete Knoten(Attribute bzw. Elemente) haben, 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 fünf untergeordnete Knoten hat:
    - Attribut <xmi:id>: Eine eindeutige generierte ID für jede Transition.
    - Attribut <Guard>: Eine Transition kann nur durchlaufen werden, wenn der Wächterausdruck „Guard“ wahr ist.
    - Attribut <Effect>: Übergangseffekt.
    - Attribut <source>: ID des Quell-Zustandes.
    - Attribut <target>: ID des Ziel-Zustandes.
- Element <entry> der wiederum zwei untergeordnete Knoten hat:
    - Attribut <xmi:id>: Eindeutige ID.
    - Attribut <entryLabel>: Aktionen die beim Eintreten eines Zustandes stattfinden.
- Element <exit> der wiederum zwei untergeordnete Knoten hat:
    - Attribut <xmi:id>: ID des Quell-Zustandes.
    - Attribut <exitLabel>: Aktionen die beim Verlassen eines Zustandes stattfinden.

Jetzt wird erläutert, wie man vom XML-Code zum generierten CPP-Code kommt.

```
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>
```

XSL-Dokument „transform.xsl“:

```
A.  cFSM fsm;
B.  enum {
C.      INIT = 0,
D.      Locked = FSM_Steady(1),
E.      Unlocked = FSM_Steady(2),
F.      Unlocking = FSM_Transient(1),
G.      Locking = FSM_Transient(2),
H.   };
```

Zieldokument „door.cc“:

Das formulierte match-Attribut <fsm:FSM> (Zeile 1), definiert ein Muster (Pattern), mit dem die Knoten des XML-Dokumentes verglichen werden. Bei Übereinstimmung von Muster und Knoten, wird der Knoten (oder die Menge der übereinstimmenden Knoten) durch dieses Template verarbeitet. D.h. der <fsm:FSM>-Knoten wird in diesem Fall verarbeitet.

Text innerhalb von Templates, wird vom XSLT-Prozessor in das Zieldokument übernommen, das heißt innerhalb des Templates <xsl:template match="fsm:FSM"> werden Zeile 2, 3, 4 und 11 in das Zieldokument übernommen (Zeile A, B und H).

Mit <xsl:for-each> werden die Verarbeitungsangaben des Anweisungsrumpfes auf eine Menge von ausgewählten Knoten angewendet. Die Menge bestimmt der select-Ausdruck in Zeile 5 und zwar Alle <state>-Knoten, die vom Typ Steady sind. Wobei die Anweisungen im Anweisungsrumpf (Zeile 6) einmal für jeden Knoten ausgeführt werden.

Die <xsl:value-of>-Anweisung schreibt eine Zeichenkette in den Ausgabedatenstrom. Die Zeichenkette, die von der Anweisung zurückgegeben wird, ist die Auswertung des select-Ausdrucks, somit wird in Zeile 6 der Name des Zustandes zurückgeliefert.

Mit dem Ausdruck `<position()>` liefert die Knotenposition innerhalb der gefundenen Menge zurück.
Der Text zwischen den zwei Anweisungen in Zeile 6 steht, wird einfach in das Zieldokument übernommen und somit entstehen Zeile D und E.

Für die Zeilen 8 und 9 gilt dasselbe wie bei Zeile 5 und 6, nur dass es dieses Mal für die Menge aller Transient Zustände ausgeführt wird.

TO BE CONTINUED…