Übersetzung von XSEL-Ausdrücken in andere XML-Updatesprachen

Projektarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

vorgelegt von: Hannes Grunert
Matrikelnummer: 7200076
geboren am: 30.08.1987 in Ribnitz-Damgarten
Gutachter: Dr.-Ing. habil. Meike Klettke
Betreuer: Dipl.-Inf. Thomas Nösinger
Abgabedatum: 28.03.2012

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 27. März 2012

Inhaltsverzeichnis

Kapitel 1

Aufgabenstellung

Ziel dieser Projektarbeit ist es, die von Tobias Tiedt im Rahmen des Forschungsprototyps CodeX (Conceptual Design and Evolution for XML-Schema) entwickelte XML-Updatesprache XSEL in eine andere Updatesprache zu übersetzen. Als mögliche Zielsprachen kommen XUpdate und XSLT in Frage. Eine Übersetzung in XQuery-Update wäre ebenfalls denkbar, ist aber nicht Bestandteil der Aufgabenstellung.

Die weitere Ausarbeitung ist wie folgt gegliedert:

- Kapitel 2 vergleicht XUpdate und XSLT anhand mehrerer Kriterien, um die geeignetere Zielsprache zu ermitteln. Außerdem wird die Quellsprache XSEL kurz vorgestellt.
- Kapitel 3 und 4 beschäftigen sich mit der Übersetzung in die gewählte Zielsprache(n).
- In Kapitel 5 wird die Integration in CodeX vorgestellt und die Übersetzung an einem Beispiel vorgeführt.
- Eine Zusammenfassung erfolgt in Kapitel 6.

Kapitel 2

Vergleich von XUpdate und XSLT

In diesem Kapitel erfolgt eine kurze Vorstellung von XUpdate und XSLT. Beide Sprachen werden hinsichtlich ihrer Funktionalität und der Verfügbarkeit von Prozessoren miteinander verglichen.

2.1 XSEL

Ausgangspunkt der Übersetzung ist die von Tobias Tiedt (siehe [Tie05]) entworfene Update-Sprache „XML Schema Evolution Language“ (XSEL). Dieser Sprachvorschlag wurde speziell für die Evolution von XML-Schemata entwickelt, eignet sich allerdings auch für das Update von XML-Dokumenten. Die Syntax der Sprache orientiert sich dabei an XUpdate, unterscheidet sich aber im Funktionsumfang und in der Darstellung neuer XML-Fragmente. Zur Selektion der zu verändernden Dokumentteile wird wie in XUpdate und XSLT XPath verwendet. Eine einzelne XSEL-Operation besteht aus einem XML-Element, durch dessen Namen die Funktion der Anweisung spezifiziert wird. Jedes Element kann zudem über ein oder mehrere Attribute verfügen. Folgende Operatoren stehen in XSEL zur Verfügung:

- *add, insert_before, insert_after*: Diese Operationen dienen dazu, neue Attribute und Elemente in das Schema einzufügen.
- *delete*: Durch diese Operation können Elemente und Attribute aus einem Schema wieder entfernt werden.
- *rename*: Soll ein Element oder Attribut umbenannt werden, wird diese Operation verwendet.
- *replace, change*: Durch diese Operatoren können bestehende Elemente und Attribute ausgetauscht werden bzw. ihr Wert ersetzt werden.
- *move, move_before, move_after*: Diese Operationen verschieben bestehende XML-Fragmente an eine andere Position im Schema.

Um die betreffenden Schemabestandteile auszuwählen, verfügt jede Operation über ein select-Attribut, welches als Attributwert einen XPath-Ausdruck besitzt. Gleiches gilt für die after- und before-Attribute, welche in den entsprechenden insert- und move-Befehlen verwendet werden. Die Definition neuer XML-Fragmente und Attributwerte erfolgt in den content- bzw. value-Attributen des entsprechenden XSEL-Ausdruckes.

Eine einzelne XSEL-Anweisung sieht wie folgt aus:

```
<add select="//xs:complexType[@name='kontakt']/xs:sequence"
    content="&lt;xs:element name=&quot;Nachname&quot;
      type=&quot;xs:string&quot;/&gt;"/>
```

Diese Anweisung fügt unter dem Kontextknoten, der im select-Attribut ausgewählt wurde, das neue XML-Fragment „*<xs:element name='Nachname' type='xs:string'/>*“ ein. Das Ergebnis dieser Anweisung sieht nach dem Evolutionsschritt wie folgt aus (die Sequenz war zuvor leer):

```
<xs:element name='kontakt'>
    <xs:sequence>
        <xs:element name='Nachname' type='xs:string'/>
    </xs:sequence>
</xs:element>
```

Eine vollständige XSEL-Datei besteht aus einer Abfolge von mehreren Evolutionsschritten. Damit die Anweisungen über einen XML-Prozessor, wie z.B. einen DOM-Prozessor, ausgelesen werden können, wurde im Vorfeld der Projektarbeit ein Root-Element XSELCommands eingefügt, welches die einzelnen Evolutionsschritte als Kind-Knoten beinhaltet. Eine XSEL-Datei hat somit folgenden Aufbau:

```
<XSELCommands>
    <Command 1/>
    ...
    <Command n/>
</XSELCommands>
```

Durch die Funktionalitäten des Document Object Model (DOM) ist ein effizienter Zugriff auf die XSEL-Elemente und deren Attribute möglich. Dies erleichtert die spätere Übersetzung in eine andere Update-Sprache. Alternativ ist es auch möglich, die Übersetzung mit Compilerbau-Techniken umzusetzen. Da die XML-Technologien ausgereift sind, erfolgt die Übersetzung unter Ausnutzung des DOM.

2.2 XUpdate

XUpdate ist eine speziell zum Update von XML-Dokumenten entworfene Sprache. Der Sprachvorschlag wurde von der XML:DB-Initiative entwickelt und liegt seit September 2000 als Working Draft vor. Die Arbeit an XUpdate ist scheinbar eingestellt worden, da die Webseite, auf die der XUpdate-Namespace (http://www.xmldb.org/xupdate) verweist, nicht mehr existiert. Zudem ist das bei Sourceforge (siehe [The03]) gehostete Projekt seit 2003 nicht mehr aktualisiert worden. Da XML-Schemata durch ihre Syntax als XML-Dokumente aufgefasst werden können, eignet sich XUpdate auch für die Evolution von XML-Schemata.

Ein XUpdate-Dokument besitzt ein root-Element, *xupdate:modifications*. Die Kindknoten entsprechen wie bei XSLT den einzelnen Update-Anweisungen.

XUpdate verfügt über folgenden Funktionsumfang:

- *xupdate:insert-before, xupdate:insert-after, xupdate:append*: Diese Operatoren dienen zum Einfügen neuer Element- und Attributknoten.
- *xupdate:remove*: remove entfernt bestehende Knoten aus den XML-Dateien.
- *xupdate:update*: Durch diese Anweisung werden Elemente, Attribute und Textknoten verändert.
- *xupdate:rename*: Soll ein Attribut oder Element umbenannt werden, wird dieser Operator benutzt.
- *xupdate:variable*: Soll ein Ausdruck, z.B. ein XML-Fragment oder XPath-Ausdruck, wiederverwendet werden, so kann er an eine Variable gebunden werden.
- *xupdate:value-of*: value-of wertet einen XPath-Ausdruck oder eine Variable aus. Dadurch kann der XSEL-Befehl move realisiert werden, der in XUpdate nicht existiert.

- *xupdate:if*: Dieser Befehl wird im XUpdate-Working Draft nicht weiter erklärt. Aus der Spezifikation lässt sich nur entnehmen, dass if für bedingte Anweisungen verwendet wird.

Im Gegensatz zu XSEL verfügen XUpdate-Anweisungen nur über ein select-Attribut, der zur Selektion der zu verändernden Knoten dient. Die Modellierung neuer Elemente und Attribute erfolgt über spezielle Elemente, die als Kindknoten eingefügt werden. Für das Ändern der Struktur eines XML-Schemas werden dabei folgende beide Komponenten benötigt:

- *xupdate:element*
- *xupdate:attribute*

Dadurch können neue Elemente und Attribute eingefügt werden.

Eine einzelne XUpdate-Anweisung, die äquivalent zur oben angegebenen XSEL-Anweisung ist, sieht wie folgt aus:

```
<xupdate:append select="//xs:complexType[@name='kontakt']/xs:sequence">
    <xupdate:element name='xs:element'>
        <xupdate:attribute name='name'/>Nachname</xupdate:attribute>
        <xupdate:attribute name='type'/>xs:string</xupdate:attribute>
    </xupdate:element>
</xupdate:append>
```

2.3 XSLT

Die Extensible Stylesheet Language for Transformations (XSLT) wurde ursprünglich dazu entwickelt XML-Dateien in verschiedenste Ausgabeformate zu transformieren. Da auch XML-Dateien aus Ausgabe möglich sind und der Sprachumfang von XSLT das Einfügen, Löschen und Verändern von Elementen und Attributen ermöglicht, eignet sich diese Sprache auch für die XML-Schema Evolution.

XSLT benutzt ebenfalls XPath zur Selektion der zu verändernden Knoten. Die einzelnen Transformationsregeln werden in sogenannten Template-Rules ausgedrückt. Wie im XUpdate-Sprachvorschlag existieren in XSLT spezielle Elemente zum Einfügen von Elementen und Attribute.

Die Besonderheit an XSLT ist es, dass nur die Dokumentteile übernommen werden, die in den Template-Regeln durch das match-Attribut angegeben wurden. Außerdem kann jeder Knoten nur einmal durch eine Template-Regel angesprochen werden, mehrfache Operationen auf demselben Knoten sind nicht möglich.

Das folgende Beispiel in XSLT ist äquivalent zu der oben angegeben XSEL-Anweisung:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xsl:output method="xml" indent="yes" encoding="utf-8"/>

  <xsl:template match="//xs:complexType[@name='kontakt']/xs:sequence">
    <xsl:copy>
      <xsl:apply-templates select="attribute::*"/>
      <xsl:apply-templates select="child::node()"/>
      <xsl:element name="xs:element">
        <xsl:attribute name="name">Nachname</xsl:attribute>
        <xsl:attribute name="type">xs:string</xsl:attribute>
      </xsl:element>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@*|node()">
```

```
    <xsl:copy>
      <xsl:apply-templates select="attribute::* | child::node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

2.4 Vergleich

Im Folgenden werden XUpdate und XSLT hinsichtlich Funktionalität, Verständlichkeit, Verfügbarkeit von Prozessoren und Übersetzungsaufwand ausgehend von XSEL miteinander verglichen.

2.4.1 Funktionalität

Sowohl XUpdate als auch XSLT eignen sich zum Update von XML-Dokumenten. Da ein XML-Schema ebenfalls ein XML-Dokument ist, eignen sich beide Sprachen auch zur XML-Schema-Evolution. Einschränkungen ergeben sich bei der Umsetzung des move-Operators, da dieser weder in XSLT noch in XUpdate existiert. Um diese Operation umzusetzen, wird der Befehl durch das Hintereinanderausführen einer Einfüge- und einer Lösch-Anweisung realisiert.

2.4.2 Verständlichkeit

Durch die Ähnlichkeit von XUpdate und XSEL ist es für einen Nutzer, der bereits mit XSEL-Dateien gearbeitet hat, leicht die äquivalenten XUpdate-Anweisungen zu verstehen. Durch die Strukturierung der XUpdate-Anweisungen mittels xupdate:element- und xupdate:attribute-Kindknoten ist es sogar einfacher die XUpdate-Dateien zu lesen.

XSLT-Dateien sind hingegen schwer zu verstehen. Dadurch, dass die XSLT-Anweisungen sequentiell abgearbeitet werden müssen und für jede Anweisung eine Kopierroutine für die unbeteiligten Knoten erstellt werden muss, ist die resultierende Dateigröße um ein Vielfaches höher als für eine äquivalente XUpdate-Datei. Außerdem wird die XSLT-Datei durch die XSLT-spezifischen Anweisungen unübersichtlich und für Nutzer, die zuvor nicht mit XSLT gearbeitet haben, auch schwerer verständlich.

Abbildung 2.1 zeigt deutlich die Unterschiede zwischen den einzelnen Dateigrößen hinsichtlich der Anzahl an Programmzeilen (Lines of Code, LoC).In einem Test mit einer 15-zeiligen XSEL-Datei mit 13 Update-Anweisungen ergab die Übersetzung ein 93-zeiliges, äquivalentes XUpdate-Programm, während das XSLT-Stylesheet über 500 Zeilen groß ist.

2.4.3 Verfügbare Prozessoren

Für XUpdate existieren nur wenige frei verfügbare Prozessoren. Zu den bekanntesten gehören die Referenzimplementierung Lexus(Bestandteil von Xindice), das von DB2 benutzte Xindice([The07]) und Jaxup([Bol03]). Die Wartung für diese Prozessoren wurde bereits vor Jahren eingestellt und im Rahmen dieser Projektarbeit konnte keiner dieser Implementierungen lauffähig gemacht werden. Grund hierfür sind Veränderungen in einigen Bibliotheken, von denen die XUpdate-Implementierungen abhängen. Dadurch kommt es zur Laufzeit zu Fehlermeldungen, darunter Cast-Fehlern oder nicht auffindbare Methoden. Allerdings existiert eine alternative Implementierung von Adrian Grigore([Gri05]), welche die XUpdate-Anweisungen in ein äquivalentes XSLT-Stylesheet übersetzt.

Für die Umsetzung von XSLT-Stylesheets sind mehrere Implementierungen vorhanden. Der bekannteste und weit verbreitetste XSLT-Prozessor ist Xalan ([The06]) von der Apache-Foundation. Diese Implementierung ist gut gewartet und die Fehleranfälligkeit ist gering.

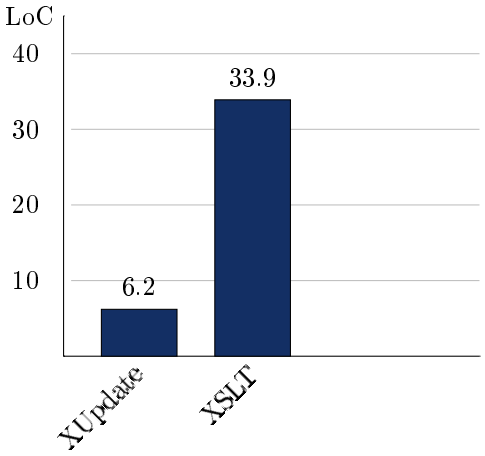

Abbildung 2.1: Lines of Code für XUpdate und XSLT im Verhältnis zu einer Zeile XSEL

2.4.4 Übersetzungsaufwand

Durch die hohe Ähnlichkeit von XSEL und XUpdate fällt der Übersetzungsaufwand zwischen diesen beiden Update-Sprachen gering aus. Die Übersetzung beschränkt sich auf das Konvertieren der Namen der einzelnen Evolutionsschritte, das Übernehmen des select-Attributes und der Umwandlung der content-/value-Attribute in XUpdate-Elemente. Außerdem muss der move-Operator aus XSEL in eine Einfüge- (append, insert-before oder insert-after) und eine Lösch-Operation (delete) zerlegt werden.

Der Aufwand für das Erstellen eines gleichwertigen XSLT-Stylesheets ist höher, da in XSLT keine entsprechenden Funktionen zum Update von XML-Dokumenten existieren. Die Funktionsweise von XSLT basiert auf der Idee, dass Elemente aus den Quell-Dateien ausgelesen werden und aus diesen Informationen eine neue Datei generiert wird. Außerdem ergeben sich folgende zwei Probleme:

- Für nicht veränderte Elemente und Attribute muss eine eigene Kopier-Routine angelegt werden, da sie ansonsten in den veränderten XML-Dokumenten nicht vorkommen.
- Wurde ein Element- oder Attribut-Knoten bereits für den Aufbau der neuen Datei verwendet, kann dieser Knoten nicht erneut für einen Evolutions- bzw Update-Schritt verwendet werden. Dadurch können mehrere Update-Anweisungen auf gleichen Knoten oder deren Kinder nicht parallel durchgeführt werden. Aus diesem Grund muss für jeden Evolutionsschritt ein eigenes XSLT-Stylesheet erstellt werden. Diese Dateien werden dann sequentiell abgearbeitet.

2.4.5 Entscheidung

Eine Übersicht über die Vergleichskriterien wird in Tabelle 2.1 gegeben. XUpdate ist ein leicht verständlicher Sprachvorschlag, der speziell für das Update von XML-Dokumenten entworfen wurde. Allerdings existieren keine lauffähigen Prozessoren, die XUpdate direkt umsetzen können. Die Übersetzung von XSEL in XSLT ist zwar prinzipiell möglich, würde vom Aufwand aber den Rahmen dieser Projektarbeit überschreiten.

Um XSEL in eine Sprache zu übersetzen, die die Updates ausführen kann, aber keinen hohen Übersetzungsaufwand verursacht, werden folgende Schritte unternommen:

1. Die XSEL-Anweisungen werden in äquivalente XUpdate-Ausdrücke übersetzt.
2. Anschließend erfolgt eine Übersetzung von XUpdate in XSLT-Stylesheets unter Verwendung des Skripts von Adrian Grigore.

3. Das XML-Schema und die dazugehörigen XML-Dokumente werden durch die Stylesheets und des XSLT-Prozessors Xalan transformiert.

Dieses Vorgehen hat den Vorteil, dass die Transformationen des Schemas und der XML-Dokumente sowohl in XUpdate als auch in XSLT vorliegen. Die Übersetzungszeit ist zwar höher als eine direkte Übersetzung von XSEL in XSLT, durch die geringe Dateigröße von XSEL-Dateien aber unbedeutend.

Kriterium	**XUpdate**	**XSLT**
Verständlichkeit	leicht, Quellcode übersichtlich	schwer, ca. 10-mal so viele LoC wie XUpdate
Standard	nein, Working Draft seit 2000	ja, Recommendation seit 1999
verfügbare Prozessoren	wenige, nicht lauffähig	viele, weit verbreitet
Funktionalität	alle Update- und Evolutions-schritte umsetzbar	alle Update- und Evolutions-schritte umsetzbar
Übersetzungsaufwand	gering	hoch

Tabelle 2.1: Vergleich von XUpdate und XSLT

Kapitel 3

Übersetzung der XSEL-Ausdrücke in XUpdate

In diesem Kapitel wird beschrieben, wie die automatisch generierten XSEL-Ausdrücke zu XUpdate-Anweisungen übersetzt werden.

XSEL ist eine auf XML basierende Sprache. Daher ist es möglich, die Anweisungen mittels eines DOM-Prozessors auszulesen und aus diesen Informationen die XUpdate-Befehle zu generieren. Das Übersetzen erfolgt dabei in drei Schritten:

1. Die einzelnen Evolutionsschritte liegen als child-Knoten des Wurzelknotens in einer NodeList vor. Über diese NodeList wird iteriert und gemäß Punkt zwei und drei die XSEL-Anweisung in einen XUpdate-Ausdruck oder mehrere XUpdate-Ausdrücke übersetzt. Außerdem wird zu jeder XSEL-Anweisung ein Kommentarknoten generiert, der eine fortlaufende Nummerierung enthält, damit nach der Übersetzung erkannt werden kann, welche XUpdate-Statements zusammengehören.

2. Von jedem child-Knoten wird der Elementname ausgelesen. Dieser Name kennzeichnet die XSEL-Operation, die anschließend auf die entsprechende XUpdate-Anweisung(en) abgebildet wird. Die genaue Abbildung wird im folgenden Abschnitt beschrieben.

3. Aus jedem child-Knoten werden die content-, value-, before- und after-Attributknoten ausgelesen. Die Attributwerte werden anschließend in XUpdate-Elemente umgewandelt.

3.1 Zuordnung der Operationen

Tobias Tiedt orientierte sich beim Entwurf von XSEL am XUpdate-Sprachvorschlag. Aus diesem Grund weisen beide Sprachen eine hohe Ähnlichkeit auf, allerdings gibt es auch einige Unterschiede.

3.2 Übersetzung der einzelnen Ausdrücke

3.2.1 add-Operation

Hinzufügen eines neuen Elementes

Um den add-Operator für ein neues Element zu übersetzen, müssen folgende Schritte umgesetzt werden:

1. Das add-Element auf xupdate:append abbilden. Dabei müssen sowohl Start-, als auch Endtag eingefügt werden.

2. Übernahme des select-Attributes inkl. Attributwert.

3. Auslesen des content-Attributes und Abbilden des Inhaltes auf xupdate:element- und xupdate:attribute-Knoten. Wird der Inhalt des content-Attributes mit einem DOM-Prozessor ausgelesen, werden die darin enthaltenen Markup-Entitäten (< , " , ...) aufgelöst, wodurch der Inhalt selbst wieder ein XML-Fragment darstellt und mit einem DOM-Prozessor abgefragt werden kann.

Beispiel:

```
<add select="//xs:complexType[@name='kontakt']/xs:sequence"
    content="&amp;lt;xs:element name=&amp;quot;Nachname&amp;quot;
      type=&amp;quot;xs:string&amp;quot;/&amp;gt;"/>
```

wird übersetzt zu

```
<xupdate:append select="//xs:complexType[@name='kontakt']/xs:sequence">
    <xupdate:element name="xs:element">
        <xupdate:attribute name="name">Nachname</xupdate:attribute>
        <xupdate:attribute name="type">xs:string</xupdate:attribute>
    </xupdate:element>
</xupdate:append>
```

Hinzufügen eines neuen Attributes zu einem bestehenden Element

Das Hinzufügen eines neuen Attributes erfolgt analog zu dem Einfügen eines neuen Elementes.

1. Das add-Element auf xupdate:append abbilden. Dabei müssen sowohl Start-, als auch Endtag eingefügt werden.

2. Übernahme des select-Attributes inkl. Attributwert.

3. Auslesen des content-Attributes und Abbilden des Inhaltes auf einen xupdate:attribute-Knoten mit Attributwert.

Beispiel:

```
<add select="//xs:element" content="maxOccurs=5"/>
```

wird übersetzt zu

```
<xupdate:append select="//xs:element">
    <xupdate:attribute name="maxOccurs">5</xupdate:attribute>
</xupdate:append>
```

3.2.2 insert_before-Operation

Das Übersetzen des insert_before-Befehl ähnelt dem append-Befehl. Der Unterschied besteht darin, dass neben dem select- noch das before-Attribut verwendet wird. Dies wird verwendet, um festzulegen, vor welchem Kontextknoten eingefügt werden soll. Das before-Attribut existiert in XUpdate nicht, da das select-Attribut in der XUpdate-Anweisung die genaue Position zum Einfügen angibt. Die Übersetzung erfolgt durch folgende Schritte:

- Das insert_before-Element auf xupdate:insert-before abbilden. Dabei müssen sowohl Start-, als auch Endtag eingefügt werden.

- Übernahme des select-Attributes und des before-Attributes inkl. Attributwerte. Der select-Wert in der XUpdate-Anweisung ergibt sich aus dem alten select-Wert, gefolgt von einem Schrägstrich („/“) und des before-Wertes.
- Auslesen des content-Attributes und Abbilden des Inhaltes auf xupdate:element- und xupdate:attribute-Knoten.

Beispiel:

```
<insert_before select="/xs:schema/xs:complexType[@name='firma']/xs:sequence"
    before="xs:element[@name='Telefon']"
    content="&lt;xs:element name=&quot;Adresse&quot;
    type=&quot;adresse&quot; minOccurs=&quot;0&quot;/&gt;"/>
```

wird übersetzt zu

```
<xupdate:insert-before
  select="/xs:schema/xs:complexType[@name='firma']/
  xs:sequence/xs:element[@name='Telefon']">
    <xupdate:element name="xs:element">
        <xupdate:attribute name="name">Adresse</xupdate:attribute>
        <xupdate:attribute name="type">adresse</xupdate:attribute>
        <xupdate:attribute name="minOccurs">0</xupdate:attribute>
    </xupdate:element>
</xupdate:insert-before>
```

3.2.3 insert_after-Operation

Das Übersetzen des insert_after-Befehls erfolgt analog zu insert_before, mit dem Unterschied, dass nicht das before-Attribut ausgelesen wird, sonder das after-Attribut.

- Das insert_after-Element auf xupdate:insert-after abbilden. Dabei müssen sowohl Start-, als auch Endtag eingefügt werden.
- Übernahme des select-Attributes und des after-Attributes inkl. Attributwerte. Der select-Wert in der XUpdate-Anweisung ergibt sich aus dem alten select-Wert, gefolgt von einem Schrägstrich („/“) und des after-Wertes.
- Auslesen des content-Attributes und Abbilden des Inhaltes auf xupdate:element- und xupdate:attribute-Knoten.

Beispiel:

```
<insert_after select="//xs:complexType[@name='firma']/xs:sequence"
    after="xs:element[@name='Telefon']"
    content="&lt;xs:element name=&quot;e-mail&quot;
    type=&quot;email&quot;/&gt;"/>
```

wird übersetzt zu

```
<xupdate:insert-after
  select="//xs:complexType[@name='firma']/xs:sequence/xs:element[@name='Telefon']">
    <xupdate:element name="xs:element">
        <xupdate:attribute name="name">e-mail</xupdate:attribute>
        <xupdate:attribute name="type">email</xupdate:attribute>
    </xupdate:element>
</xupdate:insert-after>
```

3.2.4 move-Operation

Das Verschieben eines bestehenden Elementes oder Attributes ist in XUpdate nicht durch eine einzelne Operation möglich, da im XUpdate-Working Draft kein move-Befehl existiert. Um den Befehl zu realisieren, wird zunächst der bestehende Knoten mittels append und value-of an die neue Position kopiert und anschließend gelöscht. Entsprechend werden folgende Schritte ausgeführt:

1. Anlegen eines neuen xupdate:append-Elementes mit Start- und Endtag.
2. Einfügen eines xupdate:value-of Elementes innerhalb von xupdate:append.
3. Übernahme des select-Attributes in das xupdate:value-of Element.
4. Anlegen eines neuen xupdate:remove-Elementes. Ein Endtag ist für den remove-Befehl nicht notwendig, da remove keine Kinder besitzt. Dafür muss das Tag aber am Ende geschlossen werden.
5. Übernahme des select-Attributes in das xupdate:remove Element.

Beispiel:

```
<move
    select="//xs:complexType[@name='adresse']/
      xs:sequence/xs:element[@name='Land']"
    to="//xs:complexType[@name='firma']/xs:sequence"/>
```

wird übersetzt zu

```
<xupdate:append select="//xs:complexType[@name='firma']/xs:sequence">
    <xupdate:value-of
      select="//xs:complexType[@name='adresse']/
      xs:sequence/xs:element[@name='Land']"/>
</xupdate:append>
<xupdate:remove
  select="//xs:complexType[@name='adresse']/
  xs:sequence/xs:element[@name='Land']"/>
```

3.2.5 move_before-Operation

Diese Operation ist analog zum move-Operator, mit dem Unterschied, dass zum Einfügen insert_before statt append verwendet wird. Folgende Übersetzungsschritte finden dabei statt:

1. Anlegen eines neuen xupdate:insert-before-Elementes mit Start- und Endtag.
2. Einfügen eines xupdate:value-of Elementes innerhalb von xupdate:insert-before.
3. Übernahme des select-Attributes in das xupdate:value-of Element.
4. Anlegen eines neuen xupdate:remove-Elementes.
5. Übernahme des select-Attributes in das xupdate:remove Element.

Beispiel:

```
<move_before
    select="//xs:complexType[@name='adresse']/
      xs:sequence/xs:element[@name='Land']"
    before="//xs:complexType[@name='firma']/xs:
      sequence/xs:element[@name='Webseite']"/>
```

wird übersetzt zu

```
<xupdate:insert-before
  select="//xs:complexType[@name='firma']/
  xs:sequence/xs:element[@name='Webseite']">
    <xupdate:value-of select="//xs:complexType[@name='adresse']/
    xs:sequence/xs:element[@name='Land']"/>
    </xupdate:insert-before>
<xupdate:remove
  select="//xs:complexType[@name='adresse']/
  xs:sequence/xs:element[@name='Land']"/>
```

3.2.6 move_after-Operation

Diese Operation ist analog zum move-Operator, mit dem Unterschied, dass zum Einfügen insert_after statt append verwendet wird.

1. Anlegen eines neuen xupdate:insert-after-Elementes mit Start- und Endtag.
2. Einfügen eines xupdate:value-of Elementes innerhalb von xupdate:insert-after.
3. Übernahme des select-Attributes in das xupdate:value-of Element.
4. Anlegen eines neuen xupdate:remove-Elementes.
5. Übernahme des select-Attributes in das xupdate:remove Element.

Beispiel:

```
<move_after
    select="//xs:complexType[@name='adresse']/
      xs:sequence/xs:element[@name='Land']"
    after="//xs:complexType[@name='firma']/
      xs:sequence/xs:element[@name='Webseite']"/>
```

wird übersetzt zu

```
<xupdate:insert-after
  select="//xs:complexType[@name='firma']/
  xs:sequence/xs:element[@name='Webseite']">
    <xupdate:value-of select="//xs:complexType[@name='adresse']/
      xs:sequence/xs:element[@name='Land']"/>
    </xupdate:insert-after>
<xupdate:remove
  select="//xs:complexType[@name='adresse']/
  xs:sequence/xs:element[@name='Land']"/>
```

3.2.7 delete-Operation

Um das Löschen eines Elementes in XUpdate zu realisieren, werden folgende Transformationsschritte ausgeführt:

1. Das delete-Element auf xupdate:remove abbilden.
2. Übernahme des select-Attributes.

Beispiel:

```
<delete
  select="//xs:complexType[@name='firma']/
  xs:sequence/xs:element[@name='Webseite']"/>
```

wird übersetzt zu

```
<xupdate:remove
  select="//xs:complexType[@name='firma']/
  xs:sequence/xs:element[@name='Webseite']"/>
```

3.2.8 replace-Operation

Das Übersetzen der replace-Operation ähnelt dem der append-Operation:

1. Das replace-Element wird auf xupdate:update abgebildet. Dabei müssen sowohl Start- als auch Endtag eingefügt werden.
2. Übernahme des select-Attributes inkl. Attributwert.
3. Auslesen des content-Attributes und Abbilden des Inhaltes auf einen xupdate:element- und xupdate:attribute-Knoten.

Beispiel:

```
<replace
    select="//xs:complexType[@name='kontakt']/
      xs:sequence/xs:element[name='Nachname']"
    content="&lt;xs:element name=&quot;Name&quot;
    type=&quot;xs:string&quot; minOccurs=&quot;1&quot;
      nilable=&quot;false&quot;/&gt;"/>
```

wird übersetzt zu

```
<xupdate:update
  select="//xs:complexType[@name='kontakt']/
  xs:sequence/xs:element[name='Nachname']">
    <xupdate:element name="xs:element">
        <xupdate:attribute name="name">Name</xupdate:attribute>
        <xupdate:attribute name="type">xs:string</xupdate:attribute>
        <xupdate:attribute name="minOccurs">1</xupdate:attribute>
        <xupdate:attribute name="nilable">false</xupdate:attribute>
    </xupdate:element>
</xupdate:update>
```

3.2.9 rename-Operation

Die rename-Operation wird wie folgt übersetzt:

1. Rename wird auf xupdate:rename abgebildet. Dabei müssen sowohl Start- als auch Endtag eingefügt werden.
2. Übernahme des select-Attributes inkl. Attributwert.
3. Auslesen des name-Attributwertes. Der Wert wird als Textknoten unter xupdate:rename eingefügt.

Beispiel:

```
<rename
    select="//xs:complexType[@name='adresse']/
    xs:sequence" name="xs:all"/>
```

wird übersetzt zu

```
<xupdate:rename
  select="//xs:complexType[@name='adresse']/xs:sequence">
    xs:all
</xupdate:rename>
```

3.2.10 change-Operation

Ändern des Elementinhaltes

Die Übersetzung des change-Operators für das Ändern des Elementinhaltes entspricht der Übersetzung für die replace-Operation mit dem Unterschied, dass statt dem content-Attribut das value-Attribut ausgelesen wird:

1. Das change-Element wird auf xupdate:update abgebildet. Dabei müssen sowohl Start- als auch Endtag eingefügt werden.
2. Übernahme des select-Attributes inkl. Attributwert.
3. Auslesen des value-Attributes und Abbilden des Inhaltes auf einen xupdate:element- und xupdate:attribute-Knoten.

Beispiel:

```
<change
  select="//xs:simpleType[@name='telefon']/xs:restriction"
    value="&lt;xs:pattern value=&quot;+\d{2,2}\d{3,6}/ \d{4,8}&quot;/&gt;"/>
```

wird übersetzt zu

```
<xupdate:update
  select="//xs:simpleType[@name='telefon']/xs:restriction">
    <xupdate:element name="xs:pattern">
         <xupdate:attribute name="value">+\d{2,2}\d{3,6}/</xupdate:attribute>
    </xupdate:element>
</xupdate:update>
```

Ändern des Attributinhaltes

Die Übersetzung des change-Operators für das Ändern des Elementinhaltes entspricht der Übersetzung für die rename-Operation mit dem Unterschied, dass statt dem name-Attribut das value-Attribut ausgelesen wird:

1. Das change-Element wird auf xupdate:update abgebildet. Dabei müssen sowohl Start- als auch Endtag eingefügt werden.
2. Übernahme des select-Attributes inkl. Attributwert.
3. Auslesen des name-Attributwertes. Der Wert wird als Textknoten unter xupdate:update eingefügt.

Beispiel:

```
<change
  select="//xs:simpleType[@name='email']/
  xs:restriction/xs:pattern/attribute::value"
    value="[\w.-_]*@[\w.-_]*.[\w]{2,3}"/>
```

wird übersetzt zu

```
<xupdate:update
  select="//xs:simpleType[@name='email']/
  xs:restriction/xs:pattern/attribute::value">
    [\w.-_]*@[\w.-_]*.[\w]{2,3}
</xupdate:update>
```

Kapitel 4

Übersetzung XUpdate-Ausdrücken in XSLT

Im Rahmen der Diplomarbeit von Adrian Grigore ([Gri05]) wurde ein Skript entwickelt, welches XUpdate-Dokumente in äquivalente XSLT-Skripts übersetzt. Das Prinzip der Übersetzung ist in Abbildung 4.1 dargestellt. Die Transformation von XUpdate nach XSLT erfolgt über XSLT.

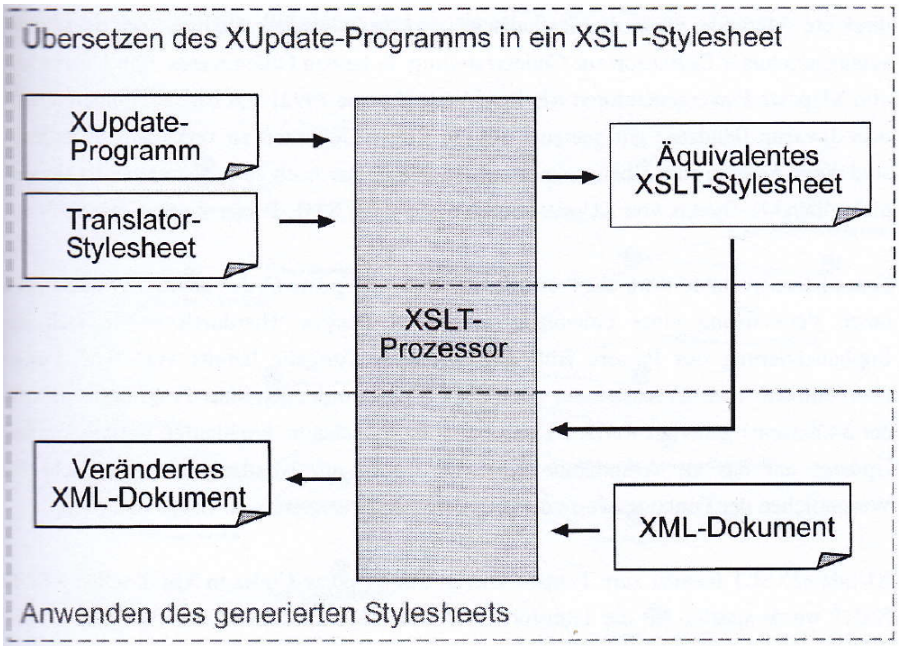

Abbildung 4.1: Funktionsweise von XUpdate2XSLT (aus [Gri05])

Das Skript benötigt als Eingabe eine gültige XUpdate-Datei. Für jeden Evolutions- bzw. Update-Schritt wird eine Template-Regel generiert. Außerdem wird eine Regel zum Kopieren nicht veränderter Knoten angelegt. Die einzelnen Regeln können anschließend als separate Stylesheets gespeichert werden, oder in einem größeren Stylesheet zusammengefasst werden. In beiden Fällen erfolgt die Abarbeitung der Updates in der Reihenfolge, wie sie in der XUpdate-Datei angegeben wurde. Die Stylesheets können anschließend dazu verwendet werden, Updates auf XML-Dokumenten auszuführen. Weitere Details können in seiner Diplomarbeit nachgelesen werden.

Kapitel 5

Implementation in CodeX

CodeX ist ein Plugin für die Programmierumgebung Eclipse. Es ist selbst aus mehreren Plugins aufgebaut. Die Java-Klassen, welche für die Übersetzung zuständig sind, wurden als Plugin zusammengefasst und zu CodeX hinzugefügt. Das Plugin trägt den Namen codex.x2x. Neben diesen Klassen enthält das Plugin außerdem noch Funktionalitäten, welche zur Einbindung in die grafische Oberfläche nötig sind.

5.1 Umsetzung

Die Zugriff in CodeX erfolgt über einen neuen Kontextmenü-Eintrag (siehe Abbildung 5.1), der nur sichtbar ist, wenn ein Rechtsklick auf eine XSEL-Datei erfolgt. Über diesen Menüpunkt ist es möglich, XSEL sowohl in XUpdate als auch in XSLT übersetzen zu lassen.

Die Erweiterung des Menüs erfolgt über die Extension-Schnittstelle der Rich-Client-Plattform von Eclipse. Über diese Schnittstelle lässt sich der Name, die Position und die Befehle des neuen Menüeintrages festlegen. Außerdem lässt sich ein Filter einrichten, damit der Menüeintrag nur bei bestimmten Dateitypen sichtbar ist. Die Struktur der Erweiterung und des Filters lässt sich Abbildung 5.2 entnehmen.

Wird der Übersetzungsprozess über das Kontextmenü angestoßen, öffnet sich zunächst ein Eclipse-Wizard (siehe Abbildung 5.3), mit dem die neue XUpdate- bzw. XSLT-Datei angelegt werden kann. Die eigentliche Übersetzung startet mit dem abschließenden Klick auf den Finish-Button. War der Übersetzungsprozess erfolgreich, so erscheint die neue Datei im Project-Explorer auf der linken Seite. Andernfalls wird eine Fehlermeldung ausgegeben.

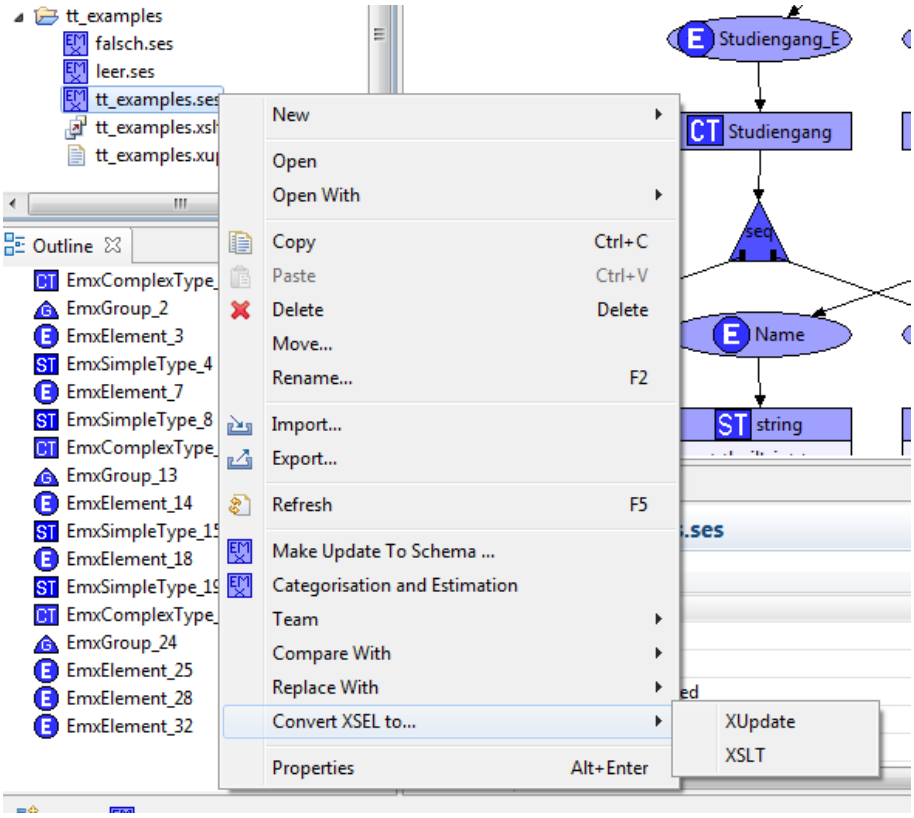

Abbildung 5.1: Neues Kontextmenü in CodeX

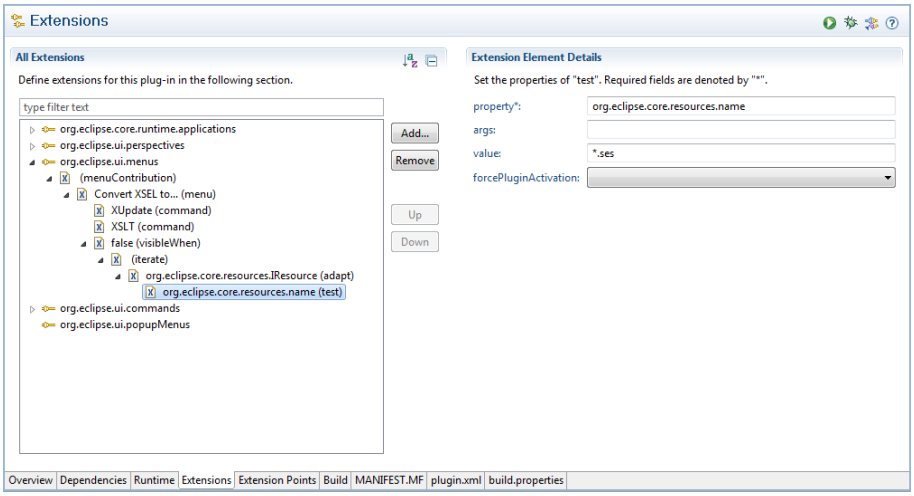

Abbildung 5.2: Erweiterung der CodeX-Funktionalitäten

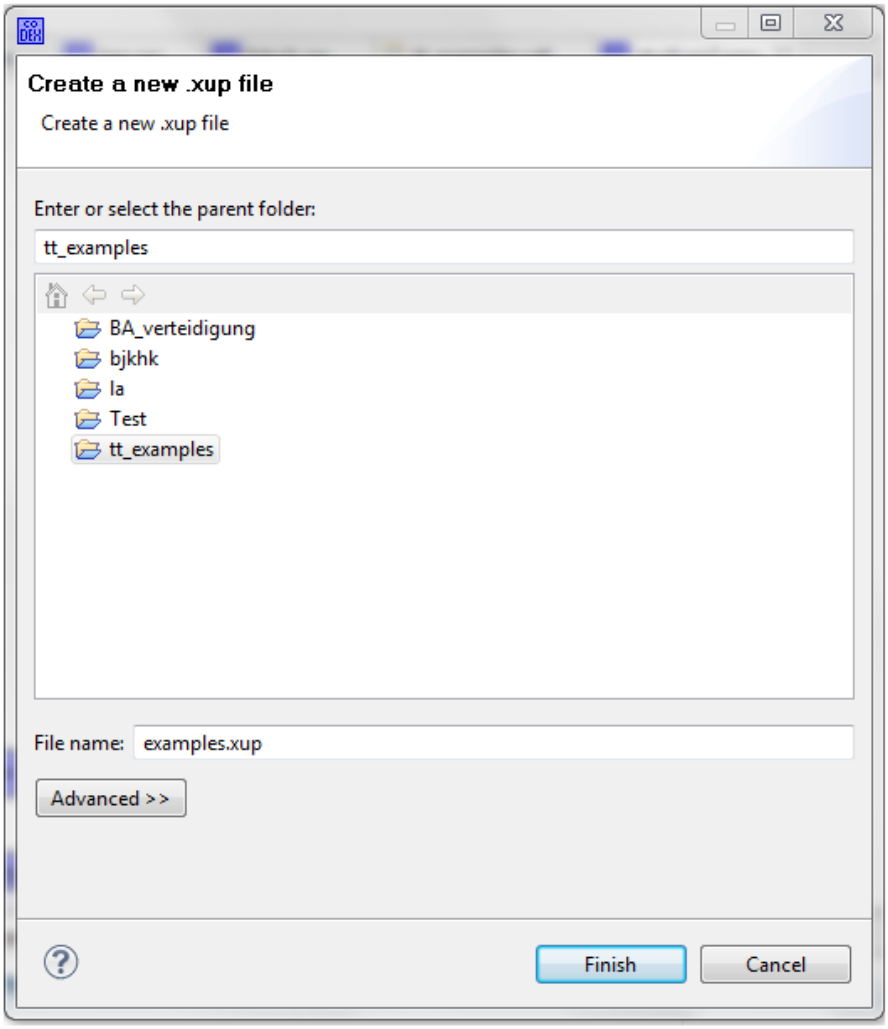

Abbildung 5.3: Wizard zum Übersetzen der XSEL-Datei

5.2 Architektur

Die Implementierung besteht im Wesentlichen aus vier Komponenten, die in Abbildung 5.4 dargestellt sind. Diese Komponenten sind der XSEL-Tokemizer, die Handler, die verschiedenen Wizards und der XSEL_Converter.

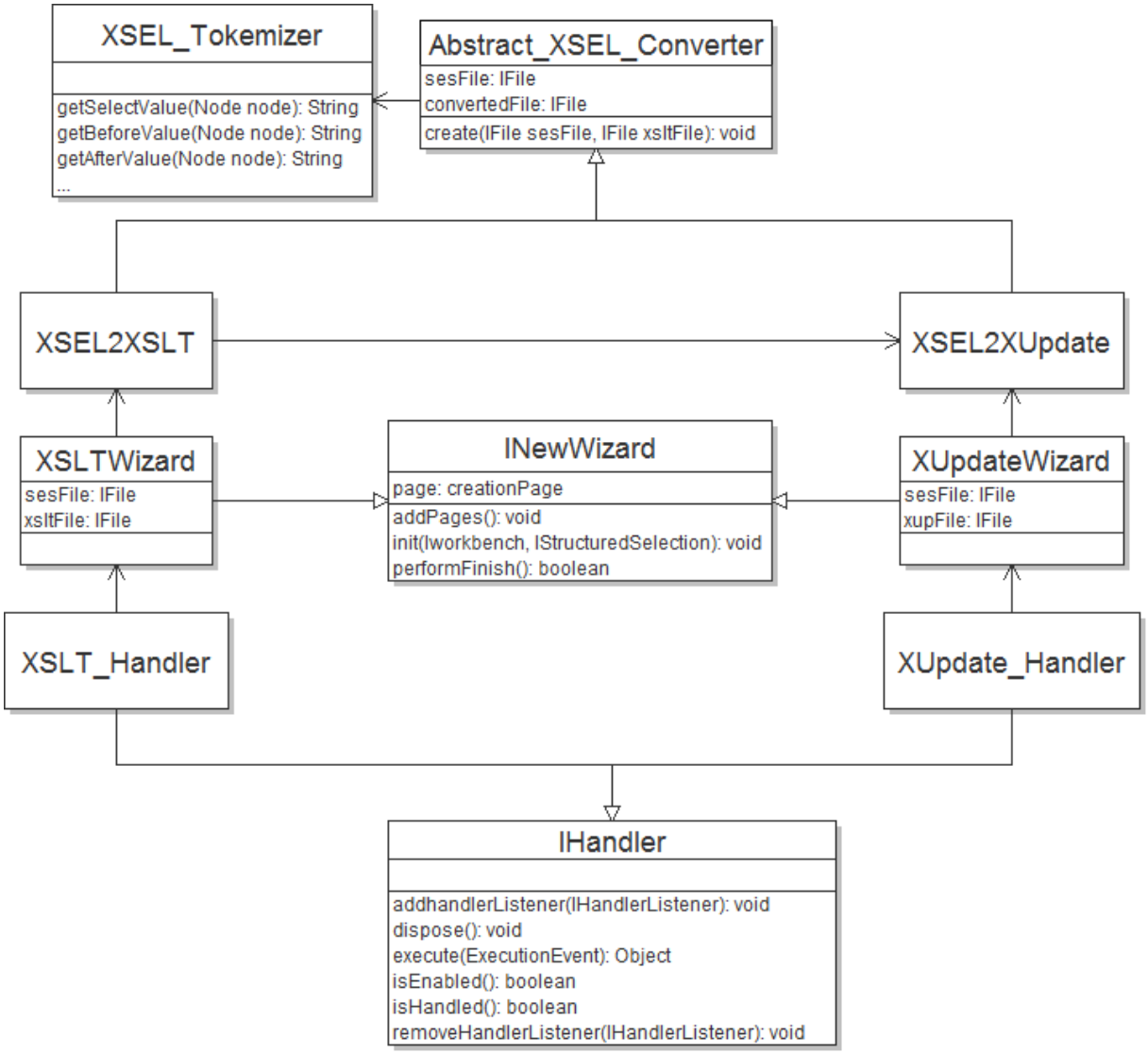

Abbildung 5.4: Klassendiagramm

5.2.1 XSEL_Tokemizer

Der Tokemizer ist für das Auslesen der Elementnamen und Attribute zuständig. Diese Werte werden zur Übersetzungszeit benötigt. Die Ermittlung dieser Informationen erfolgt über die Funktionen des DOM. **Beispiel:** Das Auslesen des select-Attributes aus einer XSEL-Anweisung

```
private String getSelectValue(Node node)
{
  String convertedSelect="";
  if(node.getAttributes().getNamedItem("select")!=null)
  {
    String selectValue = node.getAttributes().getNamedItem("select").getNodeValue();
    convertedSelect = selectValue.replaceAll(" ","");
  }
  return convertedSelect;
}
```

5.2.2 Handler und Wizards

Die Handler stellen die Schnittstelle zwischen den neuen Menüeinträgen und den Wizards dar. Der Aufruf der Schnittstelle erfolgt über die execute-Methode. In dieser Methode wird die zu übersetzende XSEL-Datei aus der aktuellen Selektion im Project-Manager ermittelt. Außerdem wird der Wizard zur Erzeugung der neuen XSLT oder XUpdate generiert. Die Wizards dienen zum Anlegen der neuen XUpdate- bzw XSLT-Dateien. Dem Nutzer wird die Möglichkeit geboten, den Speicherort und Namen der neuen Datei anzugeben oder die Übersetzung abzubrechen. Nähere Informationen zur Rich-Client-Plattform von Eclipse, die für das Verwalten der Handler und Wizards zuständig ist, können dem Tutorial von Lars Vogel([Vog12]) entnommen werden.

5.2.3 XSEL_Converter

Der XSEL_Converter ist für die eigentliche Übersetzung zuständig. Dabei erfolgt eine Iteration über alle Evolutionsschritte. Das Auslesen erfolgt über das Document Object Model und die Funktionen aus der Hilfsklasse XSEL_Tokemizer. Für jeden Evolutionsschritt wird der Tagname ermittelt und der Quellcode im dazugehörigen if-Block ausgeführt.
Beispiel: Die Übersetzung einer einzelnen insert_before-Anweisung nach XUpdate erfolgt analog zu den in Kapitel 3.2.2 beschriebenen Schritten:

1. Ermittlung des Operator-Namens. Dies geschieht mit Hilfe des Tokemizers, dessen Ergebnis in der Variable changeName gespeichert wird.
2. Abgleich von changeName mit allen Operatoren. Es wird genau die Anweisungsfolge innerhalb des if-Blocks abgearbeitet, auf die der Operator passt.
3. Ausgabe des Starttags.
4. Ermittlung und Konkatenation der select- und before-Werte zum neuen select-Wert.
5. Ausgabe des content-Wertes. Dies geschieht über die Hilfsfunktion writeElement.
6. Schließen des insert-before-Tags.

Quellcode für das Auslesen des Tagnamens und der dazugehörige if-Block

```
if(changeName.equals("insert_before"))
{
  pw.print("\t<"+XUP+"insert-before ");
  pw.println(convertSelectValue(node)+convertBeforeValue(node)+">");
  writeElement(node);
  pw.println("\t</"+XUP+"insert-before>");
}
```

Quellcode für die Hilfsfunktion writeElement:

```
private void writeElement(Node node)
{
  String content = tokemizer.getContentValue(node)
  if(content=null)
  {
    pw.print("\t\t<"+XUP+"element name=");
    writeElementName(contentValue);
    writeAttributes(contentValue);
    pw.println("\t\t</"+XUP+"element>");
  }
}
```

Kapitel 6

Zusammenfassung

Im Rahmen dieser Projektarbeit wurde die von Tobias Tiedt entworfene XML-Update-Sprache XSEL in XUpdate-Anweisungen und XSLT-Anweisungen übersetzt. Grundlage für die Übersetzung ist das Document Objekt Model, mit dem die Informationen aus XSEL ausgelesen wurden um anschließend eine XUpdate-Datei zu erstellen. Diese XUpdate-Datei leistet das Gleiche wie die ursprüngliche Datei. In einem zweiten Übersetzungsschritt werden die XUpdate-Anweisungen durch das Programm vonn Adrian Grigore in XSLT transformiert.

Die Umsetzung erfolgte als ein Plugin für das Projekt CodeX (Pluginname: codex.x2x). Der Zugriff auf die Methoden erfolgt in CodeX über einen neuen Kontextmenüeintrag, der einen Wizard zur Erstellung einer neuen Datei öffnet.

Die Übersetzung der einzelnen Evolutionsschritte wurde im Rahmen dieser Ausarbeitung kurz vorgestellt. Ein umfangreicheres Beispiel, welches neben XML-Schemata auch die dazugehörigen Dokumente anpasst, wird im Forschungsseminar präsentiert.

Abbildungsverzeichnis

Literaturverzeichnis

[Bol03] Bolwidt, Erwin: *Java XML UPdate*. http://klomp.org/jaxup/, zuletzt aufgerufen am 26.03.2012, 2003.

[Gri05] Grigore, Adrian: *XUpdate mittels XSLT - Ein XUpdate-Prozessor auf XSLT-Basis*. Diplomarbeit, Universität Rostock, 2005.

[The03] The XML:DB Initiative: *XUpdate - XML Update Language*. http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html, zuletzt aufgerufen am 26.03.2012, 2003.

[The06] The Apache Software Foundation: *Apache Xalan*. http://xml.apache.org/xalan-j/, zuletzt aufgerufen am 26.03.2012, 2006.

[The07] The Apache Software Foundation: *Apache Xindice*. http://xml.apache.org/xindice/1.2/api/org/apache/xindice/, zuletzt aufgerufen am 26.03.2012, 2007.

[Tie05] Tiedt, Tobias: *Schemaevolution und Adaption von XML-Dokumenten und XQuery-Anfragen*. Diplomarbeit, Universität Rostock, 2005.

[Vog12] Vogel, Lars: *Eclipse Plugin and Eclipse RCP Tutorials*. http://www.vogella.de/articles/EclipseRCP/article.html, zuletzt aufgerufen am 26.03.2012, 2012.