Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Semesterarbeit

XML Import/Export für Moses

Roland Kurmann

WS 99/00

Betreuung:
Dipl.-Ing. Martin Naedele
Dipl.-Inform. Jörn Janneck
Prof. Dr. Lothar Thiele

Dieser Bericht wurde mit Hilfe von LyX, LaTeX 2ε, dem KOMA-Script und xfig erstellt.

Zusammenfassung

Moses ist ein Programm für die Modellierung und Simulation komplexer, heterogener Systeme. Gewisse Datenstrukturen müssen persistent gespeichert oder für den Datenaustausch mit anderen Programmen exportiert werden.

Es treten zwei Typen von Datenstrukturen auf: deskriptororientierte und graphenähnliche.

Das Repository, die Projektverwaltung, von Moses wird mit Deskriptoren gespeichert. Die Deskriptoren speichern die Informationen zu einem Projekteintrag in Attributen. Die Deskriptoren sind untereinander durch Zeiger verknüpft.

Die in Moses modellierten Systeme werden als Moseskomponenten bezeichnet und werden als attributierte, verschachtelte Graphen gespeichert. Der Graph selbst, seine Kanten und seine Knoten, können Informationen in Attributen speichern. In den Attributen können wiederum Graphen gespeichert sein und so eine Hierarchie modellieren.

Bisher wurden diese Daten in verschiedenen proprietären Formaten gespeichert. Das Ziel war nun, das Programm auf ein standardisiertes Format abzuändern bzw. zu erweitern. Durch ein standardisiertes Format sollte der Datenaustausch mit anderen Programmen erleichtert werden.

Die *Extensible Markup Language* (XML) ist ein Standard zur Strukturierung von textuellen Dokumenten. Die Strukturierung der Dokumente wird mit sogenannten Tags vorgenommen. Tags sind Markierungen im Text, die in spitzige Klammern eingeschlossen sind. Die Tags und und deren Gebrauch können in einer Grammatik definiert werden, der *Document Type Declaration* (DTD). Mit einer DTD wird also die Stuktur einer XML Datei festgelegt.

In der Semesterarbeit wurden verschiedene Entwürfe solcher DTDs für Moses gemacht. Diese wurden analysiert und die geeignetsten für Moses ausgewählt.

Es wurden die Klassen für den Export und den Import geschrieben, einmal fürs Repository und einmal für die Moseskomponenten. Dazu wurden die vorhandenen XML Werkzeuge untersucht. Bei der Semesterarbeit wurden aufgrund dieser Erkenntnisse XML Parser eingesetzt. Es gibt zwei verschiedene XML Parser Standards, das *Simple API for XML* (SAX) und das *Document Object Model* (DOM). Diese zwei Standards unterscheiden sich in Programmierung und Geschwindigkeit. In der Semesterarbeit wurden beide eingesetzt. Durch die Programmierung des Imports des Repository mit beiden Parserstandards konnte ein konkreter Vergleich zwischen SAX und DOM gemacht werden.

Bei der Programmierung wurden die bestehenden Schnittstellen von Moses verwendet.

Die neue XML Funktionalität konnte während der Semesterarbeit in Moses integriert werden und ist in Gebrauch.

Inhaltsverzeichnis

1. Einleitung

1.1. Hintergrund

Moses ist ein Modellierungs- und Simulationsprogramm für komplexe und heterogene Systeme. Das Programm erlaubt die Modellierung der Systeme mit verschiedenen Formalismen, wie Petrinetzen, Zustandsdiagrammen und Datenflussgraphen. Die modellierten Systeme werden als Moseskomponenten bezeichnet. Der Vorteil von Moses liegt darin, dass diese Komponenten direkt getestet und simuliert werden können. Moses wird in Java entwickelt und ist objektorientiert aufgebaut.

In Moses wurden zu Beginn proprietäre Formate zur Speicherung der Daten definiert. Mit der Verbreitung von Moses kam das Bedürfnis, Daten mit anderen Programmen auszutauschen. Für den Austausch braucht es standardisierte und textbasierte Formate. Als 1998 die Extensible Markup Language (XML) definiert wurde, war ein Werkzeug da, um Datenformate und Datendokumente zu standardisieren.

1.2. Aufgabenstellung

In Moses sollen zwei Programmbereiche auf XML umgestellt oder erweitert werden:

Das Repository

Die Repository Speicher- und Ladefunktionalität soll vom bestehenden proprietären auf das XML Format umgestellt werden. Dazu muss zuerst eine DTD entwickelt werden, die das XML Datenformat beschreibt. Weiter müssen neue Filter programmiert werden, die die Repository Datenstruktur vom Hauptspeicher nach XML exportieren bzw. importieren. Diese Filter sollen die dafür vorgesehenen Schnittstellen implementieren.

Wenn möglich soll eine Abhängigkeit zu einem bestimmten Parser vermieden werden.

Export und Import von Moseskomponenten

Die Moseskomponenten werden standardmässig durch Serialisierung[1] gespeichert. Dies verhindert jedoch Änderungen am Programmcode des Moseskomponentensystems, da Java bei einer Programmänderung die alten Daten nicht mehr lesen kann. Die Speicherung soll im XML Format erfolgen. Ebenso soll, durch das XML Format, der Austausch mit anderen Programmen ermöglicht werden.

Die Moseskomponenten werden intern als attributierte, verschachtelte Graphen gespeichert, die durch das Typsystem von Moses spezifiziert werden können.

Es soll ein Datenformat für Graphen mit einer DTD spezifiziert werden. Es soll eine Filterklasse geschrieben werden, die die Graphen gemäss dem Typsystem speichert und lädt. Moses soll so erweitert werden, dass der Benutzer den Export und den Import von Moseskomponenten im GUI anwählen kann. Die Filterklassen sollen die Speicher- und Ladeschnittstellen für Moseskomponenten implementieren.

[1] Serialisierung ist die Speicherung durch Java. Java traversiert die Objektstruktur anhand der Zeiger und speichert alle Felder der Objekte ab. Java benutzt die Klassendefinition.

2. XML Überblick

2.1. Eine kurze Einführung in XML

Mit der *Extensible Markup Language* (XML) [1] können textuelle Daten strukturiert werden. Die Daten werden, wie von HTML bekannt, mit Tags gegliedert. Im Unterschied zu HTML können die Tags selbst definiert werden und haben keine graphische Bedeutung. XML wurde für den Datenaustausch zwischen Programmen entworfen. Die Struktur einer XML Datei kann mit einer *Document Type Declaration* (DTD) vorgegeben werden.

XML 1.0 wurde Anfang 1998 vom *World Wide Web Consortium* (W3C) als Standard empfohlen. XML ist eine Weiterentwicklung der *Standard Generalized Markup Language* (SGML), die 1986 als ISO 8879 [21] standardisiert wurde. XML wurde im Vergleich zu SGML vereinfacht. Während des Schreibens dieses Berichts wurde HTML in XML neuformuliert und als Standard XHTML 1.0 definiert.

2.1.1. Bestandteile von XML 1.0

Es gibt zwei Arten von XML Dateien: *well formed* und *valid*. Mit *well formed* werden die Dateien bezeichnet, die strukturell korrekt aufgebaut sind. Bei Dateien, die *valid* sind, existiert eine Strukturbeschreibung in Form einer DTD und die Dateien entsprechen genau dieser Beschreibung. Dokumente die *valid* sind sind auch *well formed*. Der Aufbau einer DTD wird in Kapitel 2.2 näher beschrieben.

Die Gross- und Kleinschreibung wird in XML unterschieden.

Eine XML Datei besteht aus folgenden Elementen:

XML Prolog

Jedes XML Dokument muss mit einem Prolog beginnen, der dieses Dokument als XML Dokument qualifiziert.

Der minimale Prolog ist:
`<?xml version=”1.0”?>`

Der Prolog kann weitere Informationen zum XML Dokument haben:

- `encoding`: Der Zeichensatz, der im Dokument verwendet wird.

- `standalone`: Es kann angegeben werden, ob diese Datei allein ist oder ob es weitere externe Dateien gibt.
- Mit "`<!DOCTYPE`" können das Haupttag, die DTD und weitere Entitäten angegeben werden. Das Haupttag bezeichnet das Tag, welches als erstes im XML Dokument erscheint. Die Angabe des Haupttags ist bei *valid* Dokumenten zwingend.

Beispiel:
```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
```

Die Definition des Zeichensatzes ist beim Austausch von Informationen zwischen verschiedenen Computerplattformen wichtig.

Tags und Attribute

Tags sind "Markierungen", eingeschlossen in spitzigen Klammern im Dokument. Tags kommen in XML immer als Paare vor (Starttag und Endtag). Das Starttag und das Endtag können zu einem einzigen Emptytag "komprimiert" werden.

Beispiel:
```
<starttag>...</endtag>
<emptytag/>
```

Die Tags können verschachtelt werden. Dies erlaubt eine hierarchische Strukturierung.

Beispiel:

```
<graph name="TimePetriNet">
    <vertex id="i53">. . .</vertex>
    <edge A="i53" B="i72">. . .</edge>
</graph>
```

Den Tags können Attribute hinzugefügt werden. Die Attribute werden innerhalb der eckigen Klammern des Starttags eingefügt. Die Attribute können nicht geschachtelt werden. Beim obigen Beispiel sind verschiedene Attribute zu sehen: `name`, `id`, `A` und `B`. Den Attributen kann ein Typ zugeordnet werden.

Es ist vielfach möglich geschachtelte Informationen als Tag oder als Attribute zu modellieren. Es gibt keine festen Regeln, wie vorzugehen ist. Es sind verschiedenen Hilfen oder Heuristiken bekannt [5, 9].

Entitäten

Entitäten sind vereinfacht gesagt Ersetzungsregeln. Es gibt zwei Arten von Entitäten: interne und externe. Die externen verweisen auf andere Dokumente, die internen stehen für Textstücke.

Beispiel zur Definition:
```
<!ENTITY intern "Dies ist eine interne Entität">
<!ENTITY extern SYSTEM "http://www.address.com/extern_entity.xml">
```

Beispiel zum Einsatz:
```
<title>Titel: &intern;</title>
<title>Titel: &extern;</title>
```

Die Entitäten werden beim Einlesen ersetzt.

Für die von XML reservierten Zeichen sind entsprechende Entitäten vordefiniert, zum Beispiel "`<`" für "<".

Das Konzept der Entitäten ist sehr mächtig und erlaubt die Modularisierung von XML Dokumenten. Ich verweise auf weitere Literatur [4], die dies im Detail beschreibt.

Kommentare

In XML können Kommentare angegeben werden. Diese werden beim Einlesen ignoriert.

Beispiel:
```
<!-- Kommentar -->
```

Processing Instructions

Eine XML Datei kann Programmanweisungen (*processing instructions*) enthalten. Diese geben dem Einleseprogramm gewisse Hinweise. Die *processing instructions* sind nicht weiter standardisiert und sind programmspezifisch.

Beispiel:
```
<?target instructions?>
```

CDATA

Die *CDATA* Bereiche werden nicht weiter geparst, sie werden als normaler Text behandelt. Damit ist es möglich, die Verarbeitung von Tags oder ähnlichem Text auszuklammern.

Beispiel:
```
<title><![CDATA>[<xml> ist gut.]]</title>
```

Conditional Sections

In XML ist es möglich Bereiche zu definieren, die bedingt beachtet werden. Bereiche mit dem Schlüsselwort `IGNORE` werden ausgelassen und Bereiche mit dem Schlüsselwort `INCLUDE` werden beachtet. Diese Schlüsselwörter können auch in Entitäten gespeichert werden.

Beispiel:
```
<![IGNORE[Ignoriere dies!]]>
<![INCLUDE[Nehme dies hin zu!]]>
<!ENTITY % ok 'INCLUDE'>
<![%ok;[Nehme dies hinzu!]]>
```

2.2. DTD

In der *Document Type Declaration* (DTD) wird die Struktur einer XML Datei festgelegt. Die DTD beschreibt eine Grammatik.

Nachfolgend werden die Bestandteile einer DTD beschrieben. Diese Übersicht soll ein intuitives Verständnis ermöglichen.

Prolog

Die DTD beginnt, wie die XML Datei, mit einem Prolog. Er hält die XML Version fest. Der Zeichensatz kann optional angegeben werden.

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

Tags

Die Tags werden in mit dem Schlüsselwort `Element` definiert. Ein Tag wird wie folgt definiert:

```
<!ELEMENT Tag_Name Struktur_der_Daten>
```

`Struktur_der_Daten` gibt die Regel für den Inhalt eines Tags an. Der Inhalt eines Tags ist Text zwischen dem Start- und dem Endtag.

Ein Tag kann folgende Strukturen enthalten:

- Es können *Kinder* definiert werden, durch Angabe deren Tags. Die Anzahl der Kinder kann bestimmt werden:
  - keine Angabe: genau einmal
  - "`?`": die Angabe im XML Dokument ist optional
  - "`+`": mindestens einmal, sonst beliebig vielmal
  - "`*`": beliebig viele Wiederholungen
- Es kann Text sein. Dieser wird mit "`#PCDATA`" gekennzeichnet.
- Durch die Angabe von "`EMPTY`" wird angezeigt, dass dieser Tag keine Daten enthält.
- Die Daten können ein Gemisch der Art "`(#PCDATA | tag)*`" sein.

Beispiel:
```
<!ELEMENT tag #PCDATA>
<!ELEMENT graph (name?, vertex+, edge*)>
<!ELEMENT leer EMPTY>
```

Attribute

Die Attribute zu den Tags werden mit dem Schlüsselwort "`ATTLIST`" definiert. Eine Attribute Deklaration ist wie folgt aufgebaut:

```
<!ATTLIST Tag_Name
          Attr_Name    Type    Angabe
          . . .
          Attr_Name    Type    Angabe>
```

XML kennt folgende Datentypen: `CDATA`, `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN`, `NMTOKENS`, `NOTATION` und eine Auswahlliste der vordefinierten Werte. Mit `ID`/`IDREF` können Beziehungen zwischen Tags hergestellt werden. Der Parser prüft bei den `ID` Attributen die Eindeutigkeit und bei den `IDREF` Attributen die Existenz des referenzierten Elemente.

Die Attribute haben einen fixen Wert (`#FIXED`), können optional sein (`#IMPLIED`) oder müssen angegeben werden (`#REQUIRED`).

Beispiel:

```
<!ATTLIST tag
          Auswahl    (eins | zwei | drei | . . .)   #IMPLIED
          Fix        "fixer_Wert"    #FIXED>
<!ATTLIST vertex
          id  ID       #REQUIRED>
<!ATTLIST edge
          A    IDREF    #REQUIRED
          B    IDREF    #REQUIRED>
```

Kommentare

In der DTD können Kommentare angegeben werden.

Beispiel:
`<!-- Kommentar -->`

2.3. XML Werkzeuge

Zu XML gibt es verschiedene Werkzeuge, die gebraucht werden können. Die am häufigsten gebrauchten Werkzeuge sind die XML Parser. Es gibt zwei verschiedene Arten von Parsern, die *Simple API for XML* (SAX) Parser und die *Document Object Model* (DOM) Parser.

Die SAX Parser haben eine sehr einfache Schnittstelle. Sie rufen bei jedem Tag eine benutzerdefinierte Methode via *Callback* auf. Dies wird als *Event* bezeichnet. Die Parser sind im allgemeinen schnell, aber nicht sehr leicht zu programmieren.

Die DOM Parser liefern ein komplettes Objektmodell der XML Datei. Das Objektmodell stellt die XML Datei in einem Baum dar, der aus verschiedenen Knoten besteht. Die

Tags werden als Knoten dargestellt. Der Zugriff auf das Objektmodell geschieht auf standardisierte Art und Weise.

Verschiedene Unternehmen stellen Parser dieser zwei Standards zur Verfügung:

- Sun Microsystems [12], [13]
- Oracle [14]
- IBM [15]
- apache.org [16]

Diese Parser sind gratis und können bei Produkten mitgeliefert werden. Genaue Details sind den Lizenzen zu entnehmen.

Die Parser der verschiedenen Hersteller wurden im Kapitel 3.3.2.3 anhand des Moses Repository verglichen.

2.4. Übersicht der XML Standards

Neben dem Hauptstandard XML 1.0, werden und wurden eine Reihe anderer Standards definiert. Ich möchte mit der folgenden Tabelle ein aktuelle Übersicht geben. (Stand: 9.2.2000).

Standard	Beschreibung	Status
SGML	Voränger aller Strukturierungssprachen.	ISO 8879
XML 1.0	Der Basisstandard.	recommendation
DTD	In XML 1.0 definierte einfache Sprache zur Deklaration von XML Dokumenten.	recommendation
XLink	Dieser Standard definiert die *Hyperlinks* von XML. XLink geht weiter als HTML, es sind zum Beispiel auch doppelt verknüpfte Links möglich.	draft
XPointer	XPointer definiert die Verknüpfung innerhalb eines Dokumentes. Xpointer kann mit XLink kombiniert werden, um Teile in anderen Dokumenten zu adressieren.	draft
XSL	Mit XSL wird die Darstellung, zum Beispiel in Browsern, von XML Dokumenten definiert.	recommendation
XSLT	Dieser Standard ist zur Transformation von XML Dokumenten mit Hilfe von XSL definiert worden.	recommendation
XPath	Wird von XSLT gebraucht um Dokumententeile zu adressieren, verwendet XPointer.	recommendation
Namespaces	Beschreibt, wie in XML Namensräume definiert werden. Dies ist wichtig, wenn zum Beispiel zwei verschiedene DTDs im selben Dokument vorkommen.	recommendation
Schemas	Schemas erfüllen die gleiche Funktion wie eine DTD, nur sind diese flexibler und bieten mehr Möglichkeiten.	draft
XML Anwendungen		
XHTML 1.0	Neuformulierung von HTML 4.01 in XML. Damit werden die Vorteile von XML auch für das World Wide Web nutzbar. Dokumente können modularisiert werden oder andere *Markup Languages* einbinden, wie zum Beispiel MathML.	recommendation
MathML	Standard zur Beschreibung mathematischer Formeln. Bisher mussten alle Formeln als Grafiken gespeichert werden.	recommendation
RDF	Standard zur Beschreibung von Resourcen mit Metadaten.	recommendation
PGML	Standard für 2D Vektorgrafiken.	draft
SMIL	Standard zur Beschreibung von Multimediainhalten.	recommendation

Mit *recommendation* wird beim W3C ein Standard bezeichnet, der Inkraft getreten ist. Wenn Sie noch in Arbeit sind werden sie mit *working draft* bezeichnet.

Die Standards können beim W3C abgerufen werden unter: `http://www.w3.org/TR/`.

3. Repository

Das Repository ist in Moses für die Verwaltung der Projekte, Klassen und Moseskomponenten und deren Beziehungen verantwortlich. Der *RepositoryManager* speichert und lädt das Repository über eine *RepositoryPersistenceManager* Strategie. Diese Funktion wurde bisher vom *RepositoryPlaintexter* übernommen, der das Repository in einem durch eine EBNF spezifizierten Format speichert . Das Ziel war nun, eine neue Strategie für XML zu implementieren, die Klasse *RepositoryXML*. Auf den Klassendiagrammen in Abbildung 3.1 ist ersichtlich, wie die XML Speicherung eingebunden werden muss.

Die XML Strategie wurde auf mehrere Klassen verteilt, um die zwei verschiedenen Ademöglichkeiten von XML zu Nutzen: *RepositoryXML*, *RepositoryXMLSax* und *RepositoryXMLDom*. *RepositoryXML* ist die Oberklasse der anderen beiden Klassen und ist für die Ausgabe des XML Dokuments verantwortlich. *RepositoryXMLSax* verwendet einen SAX Parser zum Laden einer Datei. *RepositoryXMLDom* lädt die Datei in einen DOM Baum, der dann in die Mosesstruktur konvertiert wird.

3.1. Datenstruktur des Moses Repository

Das Repository besteht aus einer Reihe von Deskriptoren. Jeder Deskriptor hat mehrere Einträge. Ein Eintrag besteht aus einem Schlüssel und einem Datenobjekt. Die Datenobjekte können Strings, Listen, Objekte, Klassen oder Zeiger auf andere Deskriptoren sein.

Beispiel eines Deskriptors für eine Moseskomponente, die ein Petrinetz benutzt: Dieser Deskriptor könnte folgende Einträge haben:

- `ProjectManager_Name = GuardBuffer`
- `MosesComponent_Type = /Common/Formalisms/TimePetriNet`
- `hasContent = true`
- `ProjectManager_Parent = Descriptor Objekt`
- `ObjectAdapter = MosesComponentAdapter Objekt`

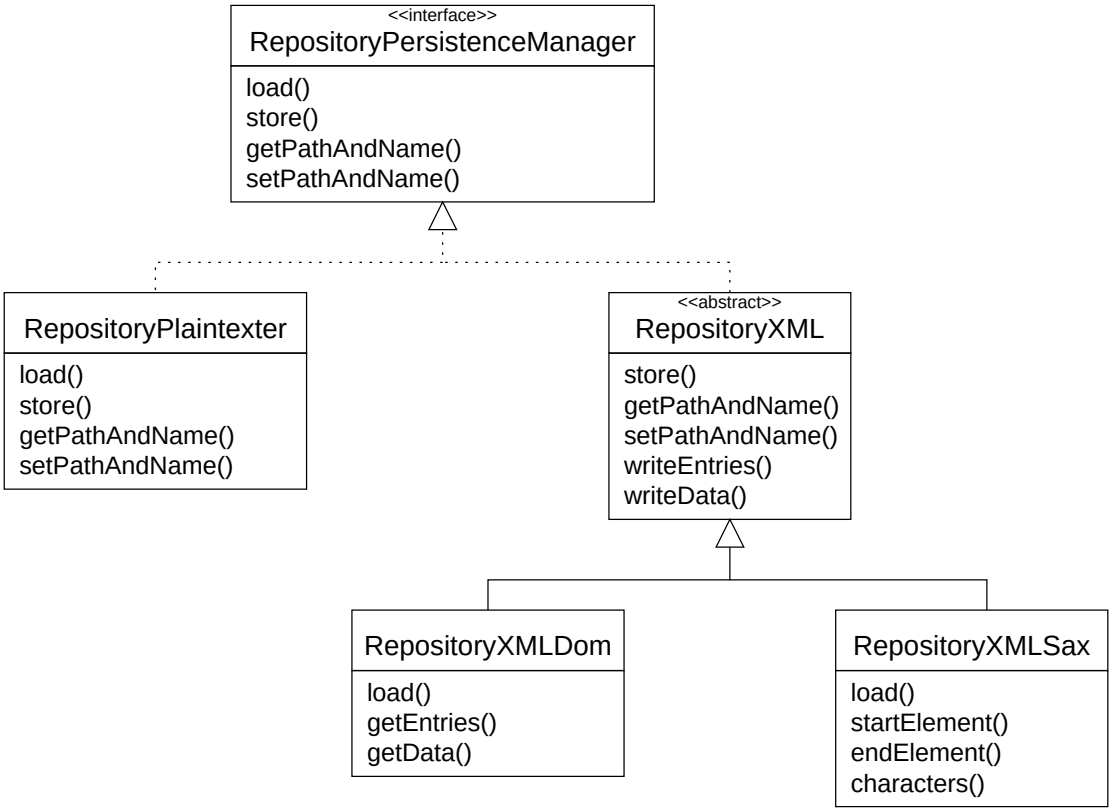

Abbildung 3.1.: Klassendiagramm der Klasse *RepositoryXML*

3.2. DTD

Es galt nun, eine DTD für das Moses Repository zu definieren. Es musste überlegt werden, wie die Datenstruktur des Repositorys am besten in eine Datei abgebildet werden kann. Um eine geeignete DTD zu finden, wurden verschiedene Entwürfe gemacht. Jeder Versuch folgt einer bestimmten Idee.

Bei den DTDs wurde darauf geachtet, dass sie

- möglichst einfach sind,
- weitere Verschachtelungen zulassen,
- die Tag und Attribute Namen aussagekräftig sind.

Entwurf 1

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

<!-- document type declaration for the moses repository -->
<!-- Versuch 2, Roland Kurmann -->

<!ELEMENT   repository      (descriptor*)>
<!ATTLIST   repository>
<!ELEMENT   descriptor      (entry*)>
<!ATTLIST   descriptor
            id              CDATA          #REQUIRED>
<!ELEMENT   entry           (value|list)>
<!ATTLIST   entry
            key             CDATA       #REQUIRED>
<!ELEMENT   value           (#PCDATA)>
<!ATTLIST   value
            type            CDATA   #REQUIRED>
<!ELEMENT   list            ((value|list)*)>
```

Abbildung 3.2.: Entwurf 1 der Repository DTD

Dieser erste Entwurf, der in Abbildung 3.2 dargestellt ist, verfolgt das Ziel, eine kleine und sehr einfache DTD zu definieren. Die Bedeutung der Tags lässt sich durch die klaren Namen leicht verstehen:

- `repository` speichert das ganze Repository und dient im XML File als Haupttag.

- `descriptor` speichert einen Moses Deskriptor und eine eindeutige Bezeichnung mit dem Attribute `id`
- `entry` entspricht einem Eintrag eines Deskriptors, speichert den Namen im Attribute `key`
- `list` ist für die Speicherung von Mengen zuständig
- `value` enthält Daten, die mit dem Attribute `type` spezifiziert werden

Dieser Entwurf zeichnet sich durch seine Offenheit aus, die Typangabe im `value` Tag schreibt keine bestimmten Typen vor. Dies erlaubt den Gebrauch weiterer Datentypen ohne die DTD anzupassen. Das Ladeprogramm kann hingegen allein mit dieser DTD nicht wissen, welche Typen zu behandeln sind. Es braucht wahrscheinlich eine andere Definition oder Konvention für die unterstützen Datentypen.

Die Verwaltung der Referenzen zwischen Deskriptoren muss in diesem Ansatz selbst übernommen werden. Die Importfunktion ist dafür verantwortlich, mit fehlenden Referenzen umzugehen.

Entwurf 2

Bei diesem Entwurf, der in Abbildung 3.3 abgebildet ist, wurde darauf geachtet, dass die Struktur möglichst eindeutig in der DTD spezifiziert wird.

Das Repository startet jetzt mit einem speziellen Deskriptor, dem Initdeskriptor, welcher die Funktion einer Wurzel hat. Dies erleichtert den Aufbau des Repository im Hauptspeicher beim Einlesen, da die Wurzel gleich zu Beginn bekannt ist.

Die Namen der Deskriptoren werden hier mit dem Attribute `id` gespeichert, welches den XML Attributetyp ID hat. Es wurde auch das `ref` Tag eingeführt, welches für die Referenzen zwischen Deskriptoren da ist. Dieses Tag speichert die Referenz im Attribute `id`, welches den XML Attributetyp IDREF hat. Durch den Attributetyp ID, wird beim Einlesen der XML Datei geprüft, ob alle Namen eindeutig sind. IDREF veranlasst den XML Parser zu prüfen, ob alle Referenzen gültig sind. Diese Aufgaben werden also durch den Parser übernommen und müssen bei diesem Ansatz nicht mehr selbst programmiert werden.

Die Datentypen, die ein `value` Tag speichern kann, werden jetzt als Auswahl angegeben. Mit dieser Angabe herrscht eine klare Vereinbarung, auch für die Programme, welche Datentypen behandelt werden müssen. Die Typprüfung wird vom XML Parser durchgeführt.

Der Name eines Moses Attributs wir nicht mehr als XML Attribute gespeichert, sondern in einem eigenen `key` Tag. Dies stellt den Wert und den Namen eines Moses Attributs auf die gleiche Ebene.

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

<!-- document type declaration for the moses repository -->
<!-- 5.12.1999 Roland Kurmann -->

<!ELEMENT   repository      (initdescriptor, descriptor*)>
<!ATTLIST   repository
            version         CDATA       #REQUIRED>
<!ELEMENT   initdescriptor  (entry*)>
<!ELEMENT   descriptor      (entry*)>
<!ATTLIST   descriptor
            id              ID          #REQUIRED>
<!ELEMENT   entry           (key, (ref|value|list))>
<!ELEMENT   key             (#PCDATA)>
<!ELEMENT   ref             EMPTY>
<!ATTLIST   ref
            id              IDREF       #REQUIRED>
<!ELEMENT   value           (#PCDATA)>
<!ATTLIST   value
            type            (string|object|class)   #REQUIRED>
<!ELEMENT   list            ((ref|value|list)*)>
```

Abbildung 3.3.: Entwurf 2 der Repsitory DTD

Dem Tag `repository` wurde das Attribute `version` hinzugefügt. Diese Erweiterung verlangt die Angabe der Version des gespeicherten Inhaltes, was später bei einer eventuellen Änderung des Formates hilft, den Inhalt zu klassifizieren. So kann zum Beispiel beim Importieren, je nach Version, die alte oder die neue Einlesemethode benutzt werden.

Beschreibung der neuen Tags:

- `initdescriptor` speichert die Wurzel des Repository
- `key` enthält den Namen eines Moses Attributs
- `ref` speichert eine Referenz im Attribute `id`, welches vom XML Attributetyp IDREF ist

Entwurf 3

Dieser Entwurf, der in Abbildung 3.4 gezeigt wird, ist ziemlich ähnlich zum vorherigen. Der Unterschied besteht bei der Speicherung eines Moses Attributs. In Entwurf 2 wurde es in einem `value` Tag und der Datentypangabe im Attribute `type` gespeichert. Die Designidee ist jetzt, für jeden Typ ein eigenes Tag zu machen. Dies bietet eventuell später Vorteile, wenn zu einem Datentyp noch weitere Informationen in Form von Attributen gespeichert oder diese Tags weiter geschachtelt werden sollen.

Repository DTD

Als DTD für das Repository wurde der zweite Entwurf gewählt. Diese Variante bietet die meisten Vorteile. Die Verwaltung der Referenzen mit ID/IDREF erübrigt eigene Kontrollen beim Einlesen. Auch kann man beim Erstellen der Datenstrukturen sich auf die in der DTD spezifizierten, dadurch automatisch kontrollierten Typen bei den `value` Tags verlassen. Die ID/IDREF Attribute erhöhen zudem die Struktur des XML Dokuments, in dem Sinn, dass dies auf standardisierte Weise geschieht und so es anderen Werkzeugen ermöglicht diese Struktur zu nutzen. Es ist kein Nachteil, dass bei neuen Typen die DTD angepasst werden muss, denn auch der Parser muss angepasst werden. Durch das Vorhandensein des Initdeskriptors, wird die Struktur des XML Datei klarer und das Einlesen wird erleichtert.

Die dritte Variante ist zu wenig flexibel und verschachtelt die Daten unnötig.

3.3. Implementation in Moses

Nun galt es, das Moses Repository vom Hauptspeicher in eine XML Datei zu schreiben und wieder zu lesen. Als Datenformat wird die in Abbildung 3.3 angegebene DTD verwendet.

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

<!-- document type declaration for the moses repository -->
<!-- Versuch 1, Roland Kurmann -->

<!ELEMENT   repository      (initdescriptor, descriptor*)>
<!ATTLIST   repository
            version         CDATA       #REQUIRED>
<!ELEMENT   initdescriptor  (entry*)>
<!ELEMENT   descriptor      (entry*)>
<!ATTLIST   descriptor
            id              ID          #REQUIRED>
<!ELEMENT   entry           (key, (string|object|class|ref|list))>
<!ELEMENT   key             (#PCDATA)>
<!ELEMENT   string          (#PCDATA)>
<!ELEMENT   object          (#PCDATA)>
<!ELEMENT   class           (#PCDATA)>
<!ELEMENT   ref             EMPTY>
<!ATTLIST   ref
            id              IDREF       #REQUIRED>
<!ELEMENT   list            (((string|object|class|ref|list)*)>
```

Abbildung 3.4.: Entwurf 3 der Repsitory DTD

Für die Speicherung des Repository müssen die Import/Export Filter das *RepositoryPersistenceManager* Interface implementieren. Ein Implementationsziel war die Anpassung des Imports/Exports möglichst ohne Neukompilierung, zum Beispiel soll der XML Parser ohne Neukomilierung ausgewechselt werden können. Durch die *.properties* Dateien, die durch Java *ResouceBundle* unterstützt werden, können die Parameter zur Beeinflussung des Programms gespeichert werden. Die *.properties* Dateien sind im Textformat gespeichert und können mit einem normalen Editor geändert werden.

Die Datei *Repository.properties* enthält die Parameter für die XML Speicherung des Repository. Im Anhang C.1 ist die Datei *Repository.properties* abgedruckt. Die Angaben dieser Dateien werden vom Programm in keiner Weise überprüft und müssen den spezifizierten Bedingungen genügen. Die einzelnen Einträge in der *.properties* Datei sind jeweils mit Beispielen oder häufig gebrauchten Werten versehen. Die Datei ist in drei Teile gegliedert: Zuerst der allgemeine Teil, der die Namen der verwendeten Tags und Attribute enthält, danach die Parameter zum Laden und am Schluss die Einstellungen zum Speichern. Die externe Speicherung der Tags und Attribute würde eine Umbenennung der Tags und Attribute ohne Neukompilierung der RepositoryXML Klassen zulassen. Es gibt Tools [15], die aus einer XML Datei und der dazugehörigen DTD, die XML Datei in die Struktur einer neuen DTD konvertieren können.

Bei den *Exceptions* wurde folgende Strategie angewendet: Wenn eine Methode eine bestimmte *Exception* durch *throws* weiterreichen kann, wird diese weitergegeben. Falls dies nicht möglich ist, wird sie abgefangen und auf der Konsole ausgedruckt. Die *RepositoryXML* Klassen haben keinen Bezug zur grafischen Oberfläche und können deshalb keine Fehlerdialogbox aufrufen.

Ein weiteres Ziel bei der Implementierung war die Herstellerunabhängigkeit. Weil es verschiedene proprietäre Schnittstellen gibt, sollen diese durch eine jeweils eigene Strategie gekapselt werden.

Der Export und der Import werden nun in eigenen Kapiteln genau betrachtet.

3.3.1. Export

Der Export kann mit den folgenden Parametern beeinflusst werden:

- `BACKUP_EXTENSION`: Dateiendung, die dem alten Repository Datei hinzugefügt werden soll.
- `WRITE_ENCODING`: Die XML Datei wird in diesem Zeichensatz geschrieben und im XML Prolog angegeben. Der Zeichensatz muss von Java unterstützt und von den XML Parsern beim Laden erkannt werden. Die Angabe muss in der von XML verlangen Version,angegeben werden, zum Beispiel "UTF-8" oder "ISO-8859-1". Die Details sind in der Regel 80 des XML 1.0 Standards [1] nachzulesen.
- `REP_DTD_FILE`: Name der DTD, die in der zu schreibenden XML Datei angeben wird.
- `ID_PREFIX`: Präfix, welcher bei IDs vorangestellt werden soll. IDs werden in XML wie Namen behandelt. Diese müssen mit einem Buchstaben, "_" oder ":" beginnen. Der Präfix muss den Bedingungen eines XML *Name* genügen.
- `INDENT_STR`: Einrückungszeichen oder Zeichenfolge, welche bei jeder weiteren Verschachtelung der Tags vorangestellt werden soll. Es sollen Leerzeichen, Tabulatoren oder nichts angegeben werden.
- `LINE`: Bestimmt, ob eine Linie zwischen zwei Deskriptoren in der XML Datei eingefügt werden soll.

Das Repository wird in der Klasse *RepositoryXML* mit Methode *store* gespeichert. Der Export wurde selbst programmiert und benötigt keine anderen Werkzeuge.

Nach der Sicherheitskopie der alten Datei, wird ein *PrintWriter* mit der angegebenen Zeichencodierung geöffnet. Danach wird der XML Prolog geschrieben.

Es wird die Version der Repositorystruktur geschrieben, die jetzt "1.0" ist. Diese Versionsnummer soll es in Zukunft ermöglichen, zu erkennen in welcher Struktur das Repository gespeichert wurde. Eine andere Möglichkeit ist, dass bei jeder Änderung der

Struktur eine neue DTD definiert wird. Die Versionsnummer hat den Vorteil, dass damit auch semantische Änderungen unterschieden werden können. Bei den Deskriptoren wird der Initdeskriptor zuerst gespeichert, danach die restlichen Deskriptoren. Die Methode *writeEntries* schreibt alle Moses Attribute eines Deskriptors, wobei sie die Methode *writeData* aufruft, die den Inhalt eines Moses Attributs schreibt. Diese beiden Methoden können rekursiv gebraucht werden.

3.3.2. Import

Um das Repository von einer XML Datei zu laden gibt es 3 Möglichkeiten: Erstens einen eigenen Parser zu programmieren, zweitens SAX oder drittens DOM benutzen. Die erste Möglichkeit fiel schnell ausser Betracht, da der Aufwand einen korrekten Parser zu schreiben, der alle Anforderungen von XML erfüllen kann, erheblich ist. So blieben die Varianten zwei und drei. Es gibt verschiedene Gründe, sich für die eine oder andere Variante zu entscheiden. SAX ist schneller als DOM, aber aufwendiger zu programmieren, insbesondere muss die Datei in einem Durchgang geparst werden. DOM stellt ein vollständiges Objektmodell zur Verfügung.

Da die DOM Variante einfacher zu sein schien, testete ich die Geschwindigkeit von DOM mit einem vereinfachten Repository. Das Ziel des Tests war die Frage, ob die Geschwindigkeit bei DOM linear zur Grösse der Eingabedatei ist. Für den Test wurde die konvertierte *demo.rep* Datei verwendet, die aus ca. 400 Deskriptoren besteht und als XML Datei 600 KByte gross ist. Diese Datei wurde als 1 normiert. Die grösseren Dateien wurden durch Zusammenkopieren dieser Referenzdatei erhalten, die ID/IDREF Überprüfung wurde in der DTD ausgeschaltet, damit die doppelten `ID` Attribute keinen Fehler erzeugen. Es wurde der Sun DOM Parser XTR2 [11] als Testparser verwendet, da dieser als Arbeitsparser diente. Als Testumgebung wurde ein Sun Ultra 30C mit 640 MByte RAM verwendet, mit Java JDK 1.3beta.

Grösse der Eingabedatei	Ladezeit DOM
1x	10s
2x	20s
3x	29s
4x	38s
8x	74s

Da der Test die Linearität bestätigte, wurde die Implementation mit DOM begonnen.

3.3.2.1. DOM

Die DOM Parser, die zu Beginn der Arbeit zur Verfügung standen, mussten alle auf eine andere Art instanziert werden. Ich entschloss mich deshalb, die Instanzierung eines

Parsers durch eine *DOMParserWrapper* Strategie vornehmen zu lassen, die die Instanzierung kapselt. Durch diesen Schritt wird die Klasse *RepositoryXMLDom* unabhängig von den Parser Herstellern. Die Schnittstelle und die verschiedenen Parser Strategien sind in Kapitel 5.2 beschrieben. Gegen Ende der Semesterarbeit wurde von Sun ein Standard [10] zur einheitlichen Instanzierung vorgelegt. Dieser Standard wurde auch als Strategie eingebaut: *DOMParser_javax*. Wenn sich dieser Standard durchgesetzt hat, wird die *DOMParserWrapper* Strategie obsolet.

Der Parser wird in der *RepositoryXML.properties* Datei angegeben:

- `DOM_PARSER`: Dieser Parameter stellt die DOM Parser Strategie ein. Man gibt die Strategie mit dem Klassennamen an. Der Parser und die Strategie müssen vorhanden sein. Mit *moses.xml.DOMParser_javax* wird der neue Sun Standard eingestellt.
- `JAVAX_XML_FACTORY`: Dieser Parameter wird nur gebraucht, wenn bei DOM_PARSER die *DOMParser_javax* Strategie, also der neue Sun Standard, gewählt wurde. Der Parameter darf leer sein, aber nicht weggelassen werden. Mit diesem Parameter wird die Factory [12] zur Erstellung der Parser eingestellt.

Ich möchte nun die wichtigen Schritte des Programms skizzieren:

Beim Laden wird zuerst der Parser instanziert, die Datei eingelesen und als Document Object Model zur Verfügung gestellt. Als erstes wird eine Versionsüberprüfung vorgenommen. Danach werden alle Deskriptor Objekte erzeugt. Die Verknüpfung zwischen den, in der XML Datei gespeicherten und den erzeugten Deskriptoren wird mit java.lang.Map *refs* hergestellt. Damit können später die Referenzen zwischen den Deskriptoren wiederhergestellt werden. Im nächsten Schritt werden die Deskriptoren mit ihren Moses Attributen gefüllt, was durch die Methode *getEntries* geschieht. Diese durchläuft im DOM Baum alle `entry` Tags. Es wird der Name des Moses Attributs aus dem `key` Tag gelesen und der Wert wird mit der Methode *getData* geladen. Der Wert wird dem XML Attribute `type` gemäss restauriert. Die beiden Methoden *getEntries* und *getData* sind "dual" zu den Schreibmethoden *writeEntries* und *writeData*. Die Methoden können rekursiv verwendet werden.

Bemerkungen

Bei der Implementation sind mir verschiedene Dinge aufgefallen, die ich weitergeben möchte und um spätere mögliche Fehlerquellen zu vermeiden.

Bei DOM besitzen die Klassen *Document* und *Element* die Methode *getElementsByTagName*. Diese Methode macht eine *preorder* Traversierung beginnend beim aufgerufenen Objekt und liefert die gefunden Tags. Vom Namen her wäre aber zu vermuten, dass nur die direkten Subtags geliefert werden.

Beim IBM Parser werden teilweise noch andere Knoten ausser Tags geliefert, es muss daher geprüft werden, ob ein Knoten vom dynamischen Typ *Element* ist.

3.3.2.2. SAX

Es wurde eine Variante mit SAX implementiert, um auf der einen Seite einen Vergleich mit DOM und auf der anderen Seite eine schnellere Lösung zu haben.

Die SAX Parser müssen nicht gekapselt werden und können direkt instanziert werden, da bei SAX im Standard eine *Parser Factory* definiert ist.

- `SAX_PARSER`: Dieser Parameter gibt den SAX Parser an, der verwendet werden soll.

Die Klasse *RepositoryXMLSax* implementiert die *load* Methode vom *RepositoryPersistenceManager*, wie auch die Methoden von den *DocumentHandler* und *ErrorHandler* Interfaces. Die *load* Methode instanziert im Wesentlichen nur den SAX Parser, das eigentliche Einlesen findet in den *DocumentHandler* Methoden statt.

Die Implementation mit SAX verlangt die Programmierung eines "one pass" Parsers. Es können also nicht mehr zuerst die Deskriptoren erzeugt werden, um nachher bei der Bildung der Referenzen schon vorhanden zu sein. Durch den Charakter als Eventparser ist kein direkter Programmablauf ersichtlich und keine Rekursivität möglich. Die Verarbeitung geschieht hauptsächlich in den Methoden *startElement* und *endElement*. Die meisten Information können nicht direkt beim Aufruf der "event" Methode verarbeitet werden, so sind eine Reihe von Variablen oder Stacks nötig.

- *curStr*: Speichert den zuletzt, von der Methode *characters* eingelesenen String.
- *curRefId*: Speichert die zuletzt eingelesene Referenz, die in einem Deskriptor oder einer Liste gespeichert wird.
- *curDesc*: Speichert den aktuellen Deskriptor.
- *curValType*: Speichert den Typ des Eintrags, der bei *startElement* angegeben wird und bei *endElement* gebraucht wird.
- *curKey*: Speichert den Schlüssel des Eintrags.
- *curData*: Speichert den Wert des Eintrags.
- *stack*: Zwischenspeicherung für die Elemente einer Liste. Die Elemente können wiederum Listen sein, deshalb braucht es einen Stack.

Ausser bei den `list` Tags genügt eine Variable zur Zwischenspeicherung, da diese Informationen nicht verschachtelt vorkommen können. Die Variablen werden auf *null* zurückgesetzt, sobald sie nicht mehr gebraucht werden. Dies geschieht aus Sicherheitsgründen, da so Fehler leichter entdeckt werden können, ausser bei *curStr*. Wenn *curStr* gleich *null* ist, bedeutet dies, dass ein neuer String eingelesen wird. Bei der Methode *characters*, die

via *Callback* aufgerufen wird, kann man nicht davon ausgehen, dass die ganzen Daten in einem Aufruf geliefert werden.

Wenn ein anderes Deskriptor Objekt referenziert wird, wird geprüft, ob dieses Objekt schon existiert. Falls dies nicht der Fall ist, wird ein neuer Deskriptor erzeugt. Dies ermöglicht die Verarbeitung in einem Durchlauf.

Exceptions die in den Methoden *startElement* und *endElement* auftreten, werden als *SAXException* nach oben weitergereicht. Wenn die Versionprüfung in der Methode *start-Element* einen Fehler anzeigt, wird eine *SAXException* erzeugt.

Bemerkungen

Die Methode *characters*, die im *Callback* Verfahren aufgerufen wird, kann die Daten in mehreren Stücken liefern. Die Stücke müssen als Teilstücke identifiziert werden und dann zusammengesetzt werden. Dies kann durch eine Variable geschehen, die immer auf *null* zurückgesetzt wird nach Gebrauch. Somit muss eine Variable ungleich *null* ein Teilstück sein.

3.3.2.3. Vergleich

Nachdem beide Varianten programmiert worden sind, konnte ein Vergleich von SAX und DOM am Beispiel des Repository gemacht werden.

Programmcodevergleich SAX vs DOM

Der Programmcode sieht bei beiden ziemlich unterschiedlich aus. Bei der Lösung mit SAX ist kein direkter Programmablauf ersichtlich, da die Steuerung durch die "events" vorgegeben ist. Bei DOM hingegen wird der Ablauf vom Programmierer bestimmt, der Ablauf gliedert sich in die aufeinanderfolgenden logischen Schritte. Insbesondere kann bei DOM rekursiv programmiert werden, die Variante mit DOM ist dadurch besser lesbar und wartbar. Eine spätere Veränderung oder Erweiterung wird deshalb bei DOM leichter sein.

Geschwindigkeitesvergleich SAX vs DOM

Ich habe die Geschwindigkeiten der verschiedenen Lösungen verglichen. Ich habe die SAX und DOM Varianten, wie die Parser der verschiedenen Hersteller verglichen. Die Resultate sind in der Tabelle 3.1 verglichen. Die Tests wurden auf einer Sun Ultra 30C mit 640 MByte RAM durchgeführt. Ich benutzte Java 1.3beta. Die Tests wurden folgendermassen durchgeführt: Ich veränderte die Methode *load* der Klasse *Repository*, damit diese die Zeit vor und nach dem Laden des Repository ausgibt. Damit wird die effektive Ladezeit gemessen. Ich führte das Laden jeweils dreimal durch und mittelte die Ergebnisse. Als Testdaten wurde das zur Zeit aktuelle Moses Repository genommen, welches ca. 400 Deskriptoren besitzt. Das Repository hat als XML Datei eine Grösse von 640 KByte, die sich über 25700 Zeilen erstrecken.

Parser	DOM	SAX
javax: Project X TR2	20s	8s
Sun: Project X TR2	16s	8s
IBM: XML Parser for Java V2	20s	8s
Oracle*: XML Parser v2	15s	7s
Plaintexter (proprietär)	3s	

Tabelle 3.1.: Geschwindigkeitsvergleich der verschiedenen Parser

Die Tests zeigen einen deutlichen Unterschied zwischen DOM und SAX, DOM braucht mehr als doppelt soviel Zeit um die Moses Datenstruktur aufzubauen. Die Parser der verschiedenen Hersteller sind bei SAX ziemlich gleich, ausser der Oracle Parser, der durch die verwendete Instanzierung nur einen "nonvalidating" Parser erzeugen kann. Bei DOM hat der Hersteller mehr Freiheiten, was sich bei unterschiedlicheren Resultaten niederschlägt. Am schnellsten war der Parser von Oracle, gefolgt von demjenigen von Sun. Der Parser von IBM liegt am Schluss. In der Beschreibung zum javax Parser heisst es, dass einzig die Schnittstelle unterschiedlich zum *Project X TR2* Parser von Sun ist. Für mich ist deshalb der Zeitunterschied nicht richtig erklärbar, ausser vielleicht damit, dass der javax Parser eine "early access" Produkt ist.

Speichervergleich SAX vs DOM

Während des Ladens mit einem DOM Parser sind zwei Datenstrukturen mit den gleichen Daten vorhanden, die Moses Datenstruktur und das Objektmodell von DOM. Es stellte sich die Frage, wieviel Speicher das Objektmodell benötigt.

Zu diesem Zweck schrieb ich ein Testprogramm, welches nur ein Repository einliest. Ein Speichertest ist in Java wegen der Garbage Collection nicht so einfach. Ich startete das Speichertestprogramm mit einem Initialspeicher[1] von 20 Mbyte. Ich führte vor dem Laden eine erzwungene Garbage Collection durch, gab den freien und totalen Speicher aus, startete den Ladeprozess und zeigte den Speicher erneut. Danach führte ich nochmals

Parser	Moses Datenstruktur	DOM Objektmodell
Sun: Project X TR2	1001488 Bytes	1034696 Bytes

Tabelle 3.2.: Speichervergleich SAX vs DOM

eine erzwungene Garbage Collection aus, die das Objektmodell von DOM wieder freigab und damit nur noch die Moses Datenstruktur vorhanden war. Der Test wurde auf dem gleichen Computer wie oben durchgeführt. Der Test zeigt, dass das Objektmodell in etwa gleichviel Speicher braucht wie Moses selbst. Der Verbrauch von einem Megabyte ist kein Argument für oder gegen DOM und kann vernachlässigt werden.

[1] Parameter bei Java 1.3beta: -Xms20m

Dateigrössenvergleich

Ich verglich die XML Datei mit der alten *Plaintexter* Variante. Durch die Metadaten, also die Tags und die Attribute wurde die XML Datei grösser. Die gespeicherte Repository XML Datei ist 641270 Bytes gross und hatte 25769 Zeilen. Die *demo.rep* Datei belegte 278824 Bytes und hatte 19938 Zeilen. Die Anzahl der Zeilen ist jedoch sehr stark abhängig vom Layout, das zur Speicherung gewählt wird.

Geschwindigkeitsvergleich mit der *Plaintexter* Variante

Der *Plaintexter* ist der alte Import/Export Filter mit dem proprietären Datenformat. Wenn man die Geschwindigkeit von der *Plaintexter* Variante betrachtet, ist diese deutlich schneller als bei XML, auch mit SAX Parsern. Dies liegt daran, dass bei XML die Struktur genau anhand der DTD überprüft wird.

Fazit

Langfristig ist die DOM Variante besser geeignet, da sie wartbarer ist und eine Objektmodell zur Verfügung stellt. Solange keine Änderung des Programms gemacht werden muss, kann die SAX Lösung verwendet werden. Was geschwindigkeitsmässig noch zu versuchen wäre ist, die Verwendung von "nonvalidating" Parsern. Bei SAX können diese leicht angegeben werden, bei DOM müssen neue ParserWrapper geschrieben werden, oder auf den einheitlichen Standard mit der *ParserFactory* gewartet werden. Bei den "nonvalidating" Parsern wird die mögliche Geschwindigkeitssteigerung durch einen Verlust an Sicherheit bezahlt. Ich habe deshalb während meiner Arbeit dies ausser Betracht gelassen.

3.3.3. Converter

Um die Daten des alten Repository weiter gebrauchen zu können habe ich ein Konvertierprogramm geschrieben. Das Programm wurde allgemein gehalten, so dass zwischen beliebigen Repositorys konvertiert werden kann. Das Programm ist ein Konsolenprogramm und wird über die *RepositoryConverter.properties* Datei gesteuert. Dies hat den Vorteil, dass von einer Vorlage ausgegangen werden kann und die Befehle nicht immer ganz eingegeben werden müssen. Die Datei hat folgende Parameter:

- `INPUT_MANAGER`: Hier wird die Ladeklasse angegeben, die die alten Daten richtig einlesen kann. Diese Klasse muss das *RepositoryPersistenceManager* Interface implementieren.
- `INPUT_FILE`: Angabe der Datei, die die alten Daten enthält.
- `OUTPUT_MANAGER`: Hier wird die Speicherklasse angegeben, die die Daten im neuen Format schreibt. Diese Klasse muss das *RepositoryPersistenceManager* Interface implementieren.

- `OUTPUT_FILE`: Der Pfad und Name der neuen Datei.

Das Konvertierprogramm ist ein eigenständiges Java Programm. Das Programm wird folgendermassen gestartet:

```
java moses.repository.structure.RepositoryConverter -classpath Moses:XML
```

Die Angabe der Parser (oben als `XML` bezeichnet) im Classpath darf nicht vergessen werden. Das Programm zeigt jeden Schritt den es macht an und ist fertig, wenn die Ausgabe "successful conversion" erscheint. Sonst erscheint eine Exception, die das Problem näher beschreibt. Häufige Fehlerursachen sind falsche oder unvollständige Klassenangaben oder falsche Pfade bei den Dateien.

3.4. Einsatz in Moses

Um Moses mit der XML Funktionalität zu erweitern muss ein XML Parser vorhanden sein. Der Parser muss im Classpath angegeben werden. Es können auch mehrere XML Parser angegeben sein. Die Parser sind meist in einer *Jar* Datei.

Die Repository Speicherung mit XML wird in der Datei *Repository.properties* eingestellt. Dort wird die Klasse und die Datei festgelegt.

Beispiel:
```
persistencemanager=moses.repository.structure.RepositoryXMLSax
```

Die Steuerung des Repositorys wird in der Datei *RepositoryXML.properties* vorgenommen.

Die Repository DTD muss zum jetzigen Zeitpunkt im selben Verzeichnis sein, wie die XML Datei. Dies ist nicht ganz befriedigend, denn dies kann leicht vergessen werden. Im Kapitel 6 wird näher auf dieses Problem eingegangen.

4. Moseskomponenten

Die Moseskomponenten enthalten die mit verschiedenen Formalismen modellierten Systeme. In diesem Teil der Semesterarbeit geht es um die Umstellung der Speicherung dieser Komponenten auf XML. Die Komponenten werden intern als attributierte, verschachtelte Graphen gespeichert. Diese müssen mit XML gespeichert und geladen werden können.

Ich wiederhole im Kapitel 3 bereits beschriebenes nicht mehr.

4.1. Datenstruktur

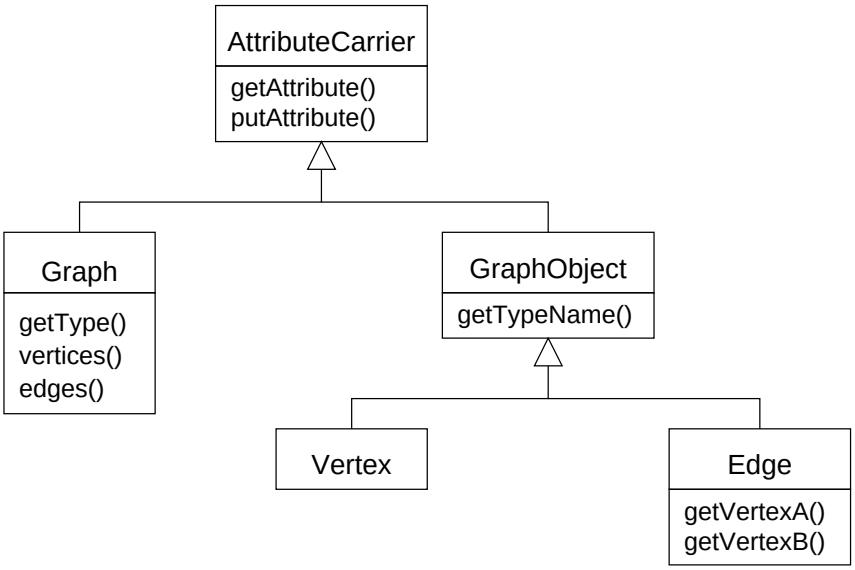

Abbildung 4.1.: Vereinfachtes Klassendiagramm der Graphen

Die attributierten, verschachtelten Graphen sind mathematische Graphen mit Knoten und Kanten, die zusätzlich zu jedem Knoten, zu jeder Kante und zum Graph selbst, Attribute aufnehmen können. Ein Attribut besteht aus einem Schlüssel und einem Datum, wie zum Beispiel die Länge oder die Farbe. Alle Daten des Graphen werden in Form von Attributen gespeichert. Die Verschachtelung der Graphen kommt dadurch zustande, dass Graphen selbst wieder in Attributen enthalten sein können. Auf der Abbildung 4.1 sieht man die Klassenstruktur eines Graphen.

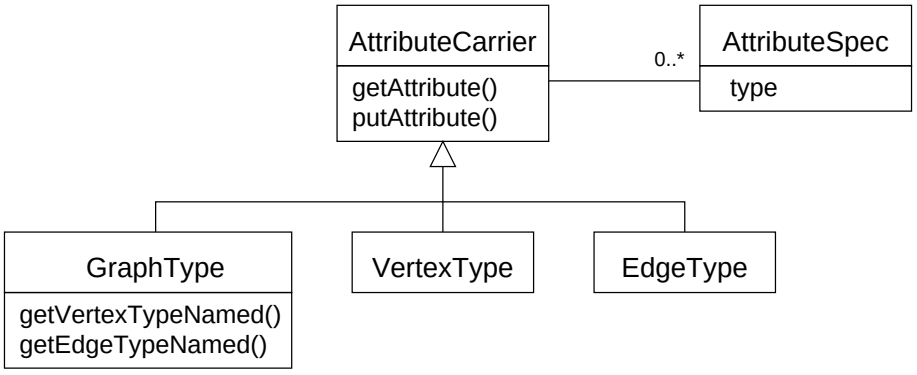

Abbildung 4.2.: Vereinfachtes Klassendiagramm zu den Graphtypen

Die Graphen können eine Typ haben. Dieser Graphtyp spezifiziert die zulässigen Knoten- und Kantenarten eines Graphen, bzw. ihre Attribute. Auf Abbildung 4.2 sieht man das Klassendiagramm zum Typsystem. Ein Attribut wird durch ein Objekt der Klasse *AttributeSpec* spezifiziert. Diese Klasse hat ein Feld namens *type*, welches den Typ als *GraphType* Konstante enthält. Im Graph sind nicht direkt die Typobjekte gespeichert, sondern nur deren Namen. Ein Beispiel ist auf Abbildung 4.3 dargestellt. Es zeigt die Objekte, die bei einem *Graph* Objekt Car vorkommen, das mit einem *TimePetriNet* modelliert ist.

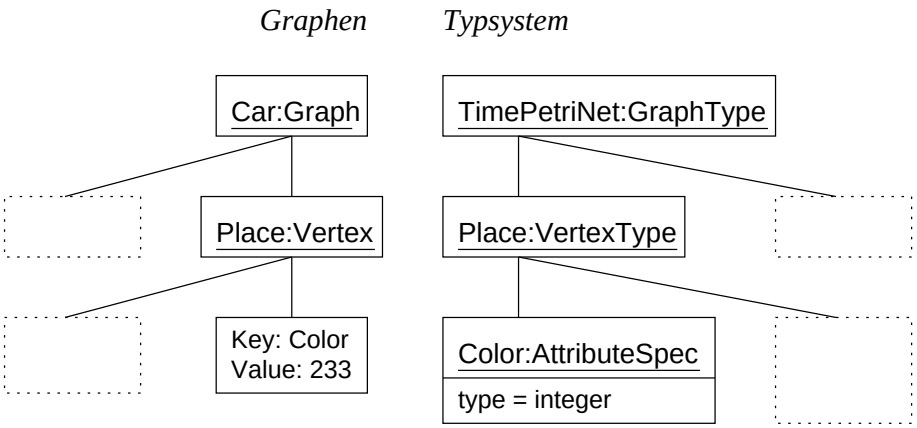

Abbildung 4.3.: Beispiel eines Graphen mit dem dazugehörenden Typsystem

Zum Zeitpunkt der Arbeit war das Mosestypsystem nicht vollständig. Bei vielen Attributen ist keine Typangabe vorhanden.

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

<!-- document type declaration for a graph storage -->
<!-- 14.1.2000 Roland Kurmann -->

<!ELEMENT   graphstorage      (graph)>
<!ATTLIST   graphstorage
            version           CDATA       #REQUIRED>
<!ELEMENT   graph             ((value|graph)*, vertex*, edge*)>
<!ATTLIST   graph
            name              CDATA       #REQUIRED>
<!ELEMENT   vertex            (value|graph)*>
<!ATTLIST   vertex
            id                ID          #REQUIRED>
<!ELEMENT   edge              (value|graph)*>
<!ATTLIST   edge
            A                 IDREF       #REQUIRED
            B                 IDREF       #REQUIRED>
<!ELEMENT   value             (#PCDATA)>
<!ATTLIST   value
            name              CDATA      #REQUIRED
            type              (integer|real|string|text|color|
                               type|expressionframe|expression|
                               parameters|parsed|extraparser|
                               boolean|double|point)     #REQUIRED
            parser            CDATA      #IMPLIED>
```

Abbildung 4.4.: DTD für attributierte, verschachtelte Graphen

4.2. DTD

Die Aufgabe war nun, die Graphen in eine DTD abzubilden. Zu dieser Aufgabe haben sich zwei Werke als nützlich erwiesen, das eine enthält eine Vorgehensregel, das andere ist die Struktur eines ähnlichen Problems. In "XML Syntax Recommendation for Serializing Graphs of Data" [8] wird eine Anleitung gegeben, wie bei der Definition eines Graphen vorgegangen werden soll. In "MoML" [7] werden "Clustered Graphs" mit einer DTD beschrieben.

Ich entwickelte die in Abbildung 4.4 angegebene DTD. Diese DTD zeichnet sich durch ein einfaches, restriktives Design aus. Die DTD ist restriktiv in dem Sinne, dass die Grammatik die Struktur möglichst genau beschreibt und möglichst wenig optional lässt. Das heisst, man braucht nur die DTD zu kennen und nicht irgendwelche anderen Definitionen und Konventionen. Es sind deshalb zum Beispiel alle möglichen Typen für das `value` Tags vorgegeben. Die einzige Ausnahme ist das Attribute `parser`, welches optional ist, aber in bestimmten Fällen angegeben werden muss. Ich führte auch das Haupttag `graphstorage` ein, welches das zwingende Attribute `version` hat und nicht geschachtelt werden kann.

Durch diesen restriktiven Ansatz muss die DTD wahrscheinlich öfter angepasst werden. Dies ist aber meiner Ansicht nach nicht weiter schlimm, denn die DTD bildet eine Einheit mit dem Export und den Parsern und eine Strukturänderung muss auf allen diesen Ebenen angepasst werden.

Bei der Verschachtelung der Graphen wurde Kopiersemantik [23] angenommen, das bedeutet, dass ein Subgraph direkt in die umgebende Struktur "hineinkopiert" wird. Falls zwei Attribute auf den selben Subgraphen verweisen, würde in diesem Fall der Subgraph zweimal geschrieben. Bei der Referenzsemantik hingegen würde es nur Graphen auf gleicher Ebene geben, die durch Referenzen verknüpft sind. Bei den Deskriptoren im Kapitel 3.2 wird Referenzsemantik verwendet. Die Kopiersemantik ist bei den Graphen der Moseskomponenten in meinen Augen besser geeignet, da sie die Struktur besser modelliert.

Bemerkungen

Wenn man eine DTD definieren muss, ist es ein guter Ansatz, die Datenstrukturen des Programms nachzumodellieren.

In der DTD darf `#PCDATA` nach Regel 51 der XML 1.0 Definition [1], nur in der Form `(#PCDATA)` oder `(#PCDATA (| tag)*)*` vorkommen. Das folgende lässt sich deshalb nicht definieren: `(#PCDATA | tag)`. Bei der DTD auf Abbildung 4.4 wäre dies wünschenswert gewesen:

```
<!ELEMENT   value              (#PCDATA|graph)>
```

Das Problem wurde so umgangen, dass Graphen nicht in `value` Tags verschachtelt werden, sondern direkt in `graph`, `vertex` und `edge` eingefügt werden. Dazu mussten die Graphen angepasst werden, damit sie den Attributschlüssel selbst speichern können. Die DTD hat deshalb folgende Definitionen:

```
<!ELEMENT   graph           ((value|graph)*, vertex*, edge*)>
<!ATTLIST   graph
            name            CDATA       #REQUIRED>
<!ELEMENT   vertex          (value|graph)*>
<!ELEMENT   edge            (value|graph)*>
<!ELEMENT   value           (#PCDATA)>
```

4.3. Implementation in Moses

Alle Klassen, die in Moses etwas Speichern, müssen das Interface *ObjectStorageStrategy* implementieren. Da aber für die Speicherung mit XML, nur den *InputStream* zu kennen zu wenig ist, wurde diese Klasse erweitert und ein neues Interface *ObjectXMLStorageStrategy* definiert. Beim Einlesen einer XML Datei muss eine Pfadangabe vorliegen, damit die XML Parser die weiteren in der XML Datei angegebenen Dateien finden kann. Dies ist zum Beispiel die DTD, die von der XML Datei zum *parsen* gebraucht wird. Das Interface ist in Abbildung 4.5 dargestellt. Die Speicherung und das Laden wird in der Klasse *GraphXMLStorage* vorgenommen.

Die Steuerung der Speicherung geschieht mit der Datei *GraphXMLStorage.properties*. Es werden die Tags, Attribute, die Parser, der Zeichensatz, der Name der zugrundeliegenden DTD und Konstanten definiert. Die Konstanten wurden deshalb aufgenommen, damit nicht interne Konstanten direkt gespeichert werden müssen, die XML Datei wird also davon abgekoppelt. Zweitens, da die DTD und die XML Datei mit externen Werkzeugen zusammen konvertiert werden können. Zu diesen Konstanten zählen die Typen, die ein `value` Tag haben kann, wie integer, expression, parameter, ...

Bei den Exceptions wird die gleiche Strategie wie beim Repository in Kapitel 3.3 angewendet.

4.3.1. Export

Die Graphen müssen nun mit Hilfe der Graphtypen gespeichert werden. Die Speicherung wurde wiederum selbst programmiert.

Da XML ein textuelles Format ist, müssen alle Informationen in Form von Text ausgedrückt werden. Es gibt drei Arten von Moses Attributen, die sich durch den Typ unterscheiden:

1. Zum Attribut existiert eine *AttributeSpec* mit Typangabe.

   (a) Der Typ beschreibt einen Java Datentyp. Dieser Typ kann direkt gespeichert werden.

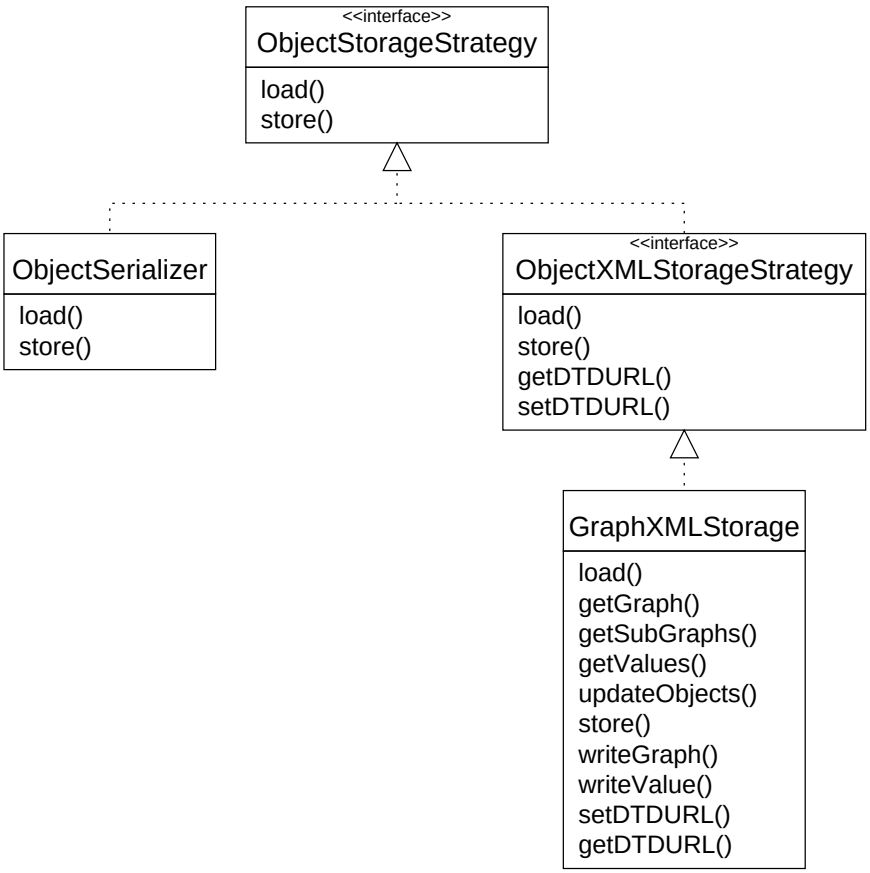

Abbildung 4.5.: Klassendiagramm zur Speicherung der Moseskomponenten

   (b) Dieser Fall trifft zu, wenn kein Java Datentyp ist. Diese Attribute können nicht direkt gespeichert werden. In diesem Fall muss ein Attribut mit dem selben Namen mit einem Anhängsel *sourceSuffix* aus der Klasse *GraphConstants* vorhanden sein, welches die textuellen Informationen bereit hält, damit später ein Mosesparser diese Objekte wieder erstellen kann.

2. Wenn keine *AttributeSpec* vorhanden ist, können zu einem Attribut, zwei andere Attribute mit den Anhängseln *sourceSuffix* und *parserSuffix* vorhanden sein. Es können dann diese Informationen in der Datei abgespeichert werden.

3. Keiner der obigen Fälle zutrifft zu und der Typ des Attributs ein Java Datentyp ist. Die Daten können direkt gespeichert werden.

Auf der Abbildung 4.3 ist der erste Fall dargestellt: es ist eine *AttributeSpec* vorhanden.

Beim Export mussten diese verschiedenen Arten abgespeichert werden. Man entschied sich die erste und die zweite Art zu unterstützen. Ich entschied mich, die dritte Art ebenfalls im Ansatz zu unterstützen, damit Tests möglich waren, da noch zuwenig Daten der ersten Art vorhanden waren. Die dritte Art hat den Nachteil, dass es viele Fallunterscheidungen braucht.

Die Implementation wurde rekursiv programmiert. Die Methode *store* schreibt den XML Prolog und ruft die Methode *writeGraph* auf. Diese Methode behandelt einen Graphen, indem der Reihe nach alle Attribute, alle Attribute der Knoten und alle Attribut der Kanten mit der Methode *writeValue* gespeichert werden. Die Methode *writeValue* speichert ein Attribute nach der Typart, wie in obiger Liste aufgezählt, ab. Die Attribute der dritten Art, werden mit einem vorgestellten XML Kommentar "`<!-- special -->`" gekennzeichnet, da diese mit der Zeit verschwinden sollen. Beim Import wird der Kommentar ignoriert und diese Attribute sind von den anderen nicht zu unterscheiden. In anderen Worten, einem Integer sieht man nicht an, ob er mit dem Mosestypsystem oder als Java Datentyp erstellt wurde.

4.3.2. Import

Beim Einlesen habe ich mich für DOM entschieden, da bei verschachtelten Strukturen die DOM Variante wesentlich einfacher ist und zudem die Datenmenge viel kleiner ist als beim Repository ist, welches bedeutet, dass die Geschwindigkeit keine Rolle spielt.

Die Methoden fürs Einlesen habe ich wiederum "dual" zum Export programmiert. Die *load* Methode ruft den DOM Parser auf, prüft die Version der XML Datei und ruft die Methode *getGraph* auf. Die Methode getGraph baut einen Graphen auf. Es werden die Attribute des Graphen, die Knoten mit ihren Attributen und die Kanten mit ihren Attributen gelesen. Die Attribute werden mit den Methoden *getSubGraphs*, für die verschachtelten und *getValues*, für die übrigen Werte eingelesen. Am Ende eines Graphen werden die "derived Attributs" mit *updateObjects* erstellt. Diese Attribute sind

nicht eigenständig und werden aus anderen Daten abgeleitet, sie müssen deshalb nicht abgespeichert werden.

In der Methode *getValues* werden die Objekte, gemäss dem in der XML Datei angegebenen Typ erstellt. Einige Daten, zum Beispiel "expressions", werden an einen Parser weitergeleitet, um daraus die Objekte zu erzeugen.

4.3.3. Einbindung

Die Einbindung in Moses geschieht über das Menü. Der Benutzer kann auf Moseskomponenten die Funktion XML Export bzw. XML Import anwählen. Die Einbindung solcher Menüpunkte ist in Moses leicht möglich durch die Ergänzung der Datei *MosesComponentAdapter.properties*. Dort werden die Klassen angegeben, die beim Anklicken gestartet werden sollen. Diese Klassen werden nachfolgend kurz beschrieben.

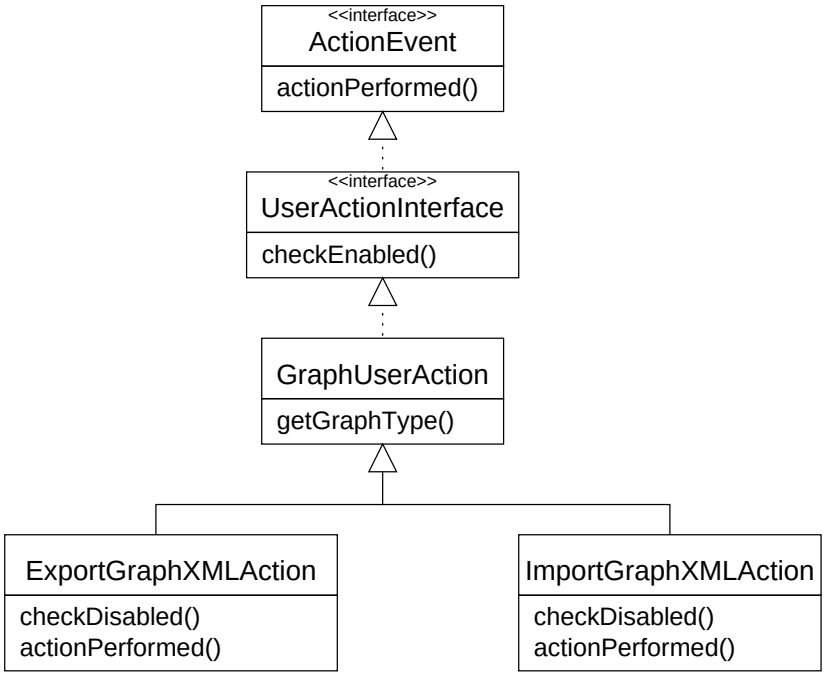

Abbildung 4.6.: Vereinfachte Klassenhierarchie zur graphischen Einbindung

ExportGraphXMLAction

Die Methode *actionPerformed* führt die Aktionen zum Exportieren durch. Zuerst wird der zu exportierende Graph geladen. Danach wird ein Dateiauswahldialog[1] gezeigt. Es wird ein *GraphXMLStorage* Objekt erzeugt, diesem wird ein *GraphTypeFinder* mitgegeben. Danach wird die *store* Methode aufgerufen.

"Export" kann vom Benutzer aufgerufen werden, wenn der Deskriptor von der Komponente nicht gesperrt und nicht leer ist.

ImportGraphXMLAction

Die Methode *actionPerformed* führt die Aktionen zum Importieren durch. Das Importieren ist etwas aufwendiger als das Exportieren. Als erstes wird der Deskriptor, der zu ladenden Komponente gesperrt. Der Benutzer kann darauf, die zu importierende XML Datei auswählen. Es wird nun ein *GraphXMLStorage* Objekt erzeugt, mit der Angabe der XML Datei und des *GraphTypeFinders*. Die Datei wird anschliessend geladen. Im Anschluss wird der Deskriptor aktualisiert mit den Angaben zur neuen Moseskomponente. Das Repository wird gespeichert und entsperrt.

"Import" kann vom Benutzer nur aufgerufen werden, wenn der Deskriptor nicht gesperrt ist.

4.4. Einsatz in Moses

Export und Import von Moseskomponenten konnte zur Zeit der Semesterarbeit nicht direkt eingesetzt werden, da das Mosestypsystem noch unvollständig ist und noch ergänzt werden muss. Doch durch den Export der gebräuchlichen Java Datentypen, auch ohne Mosestypsystem, funktioniert der Import/Export ziemlich gut. Diese Attribute sind, wie oben erwähnt mit dem XML Kommentar "`<!-- special -->`" versehen.

Die DTD muss im selben Verzeichnis sein, wie die XML Datei. Im Kapitel 6 wird näher darauf eingegangen.

[1] Es wird der JFileChooser Dialog aus "Swing" verwendet. Beim Testen des eigenen Programms wurde, ein Fehler festgestellt der bei JDK1.3beta auftritt. Der Dialog liefert ein "*null File*" zurück, wenn eine Datei mit der Maus ausgewählt und der Dateinamen abgeändert wird. In Moses wird eine Fehlermeldung ausgegeben, dass keine Daten gelesen werden konnten. Der Fehler war Sun bereits angemeldet worden und ist unter der ID 4271669 katalogisiert.

5. XML Hilfsklassen für Moses

In diesem Kapitel werden die Hilfsklassen beschrieben, die während der Semesterarbeit entstanden.

5.1. Xml

Diese Klasse enthält die statische Methode *normalize*, welche beim Schreiben der XML Datei, die von XML selbst benötigten Zeichen durch entsprechende Entitäten ersetzt, um Missverständnisse beim späteren Parsen zu vermeiden. So wird zum Beispiel das Zeichen "<" zu "<".

Diese Methode muss beim Schreiben aller Daten aufgerufen werden, die solche reservierten Zeichen enthalten könnten.

5.2. DOMParserWrapper

Die DOM Parser der verschiedenen Hersteller haben zu Beginn meiner Arbeit kein einheitliches Interface zur Instanzierung gehabt. Um trotzdem verschiedene Parser einsetzen zu können und Herstellerunabhängigkeit zu erlangen, habe ich mich entschlossen, die Instanzierung zu kapseln und in eine Strategie zu verlegen.

Die XML Datei kann entweder als *URL* oder als *InputSource* aus dem Package org.xml.sax angegeben werden. Das fertige *Document Object Model* wird als Rückgabewert geliefert.

Die Klassennamen der Strategien habe ich einprägsam und nach einheitlichem Muster benannt, da diese in den *.properties* Dateien angegeben werden müssen.

Gegen den Schluss der Arbeit hatte Sun einen Standardisierungsvorschlag [12] gemacht, der die Instanzierung für Java vereinheitlichen soll. Dieser Vorschlag befand sich während der Semesterarbeit in der "early access" Phase. Ich habe diesen Vorschlag als weitere Strategie eingebaut. Später, wenn alle Hersteller diesen Vorschlag unterstützen, wird die *DOMParserWrapper* Strategie obsolet.

5.3. IDStrategy

Mit der *IDStrategy* soll ein einheitliches Interface zur Generierung von IDs geschaffen werden. Ein Merkmal der objektorientierten Programmierung ist die Unsichtbarkeit der Objekt IDs für die Programme. Diese IDs müssen nun zur Speicherung selbst erzeugt werden, wobei es keine grosse Rolle spielt auf welche Art, denn für die Programme ist nur die semantische Bedeutung wichtig. Die IDs müssen eindeutig sein. Bei XML müssen die Verknüpfungen zwischen Objekten textuell ausgedrückt werden.

Die IDs müssen eventuell eine besondere Form haben, bei XML müssen die IDs in Form von Text dargestellt werden können. Deshalb wurde dies in einer Strategie gekapselt.

Ich habe zwei Strategien programmiert. Die Strategie *CountID* zählt die neuen Objekte der Reihe nach durch, bei schon bekannten wird die gespeicherte ID zurück gegeben. Die Strategie *HashID* benutzt den Java Hashwert zur Erzeugung von IDs. Der Hashwert muss nach der Java Spezifikation nicht eindeutig sein, was aber bei IDs gefordert wird. Deshalb wird von der Benutzung der *HashID* Strategie abgeraten.

5.4. VersionStrategy

Diese Strategie kann zur Überprüfung einer Version benutzt werden. Wenn die Methode *valid* den Wert *true* liefert ist die zu überprüfende Version korrekt.

Eine Version ist ein String, der aus zwei durch einen Punkt getrennte Zahlen besteht.

Es wurden verschiedene Strategien programmiert. Diese akzeptieren eine Version, wenn die Version exakt gleich, grösser oder kleiner ist als die Referenzversion ist.

Es wurde darauf geachtet, dass die Namen den exakten Sachverhalt beschreiben.

Beim Repository und bei den Moseskomponenten wurde die Strategie *ExactVersion* benutzt.

6. Schlussbemerkungen

Erreichtes und Einschränkungen

Im Laufe der Arbeit konnte die Aufgabenstellung erfolgreich umgesetzt und alle geforderten Punkte erfüllt werden. Der XML Import/Export wurde bereits in Moses integriert und wird damit ausgeliefert.

Während der Arbeit wurde das Datenformat der XML Dateien für das Repository und die Moseskomponenten mit einer DTD festgelegt. Die Import/Export Filter wurden programmiert.

Der XML Export von Moseskomponenten ist nur eingeschränkt möglich, da das Mosestypsystem noch nicht vollständig ist. Der Export benötigt das Typsystem zum Schreiben der XML Dateien. Wenn das Typsystem von Moses vollständig ist, ist der XML Import/Export von Moseskomponenten voll einsatzfähig.

Da die Speicherung des Repository und der Moseskomponenten etwas wichtiges und häufig gebrauchtes ist, habe ich viel Wert auf das Testen gelegt.

Während der Arbeit entstanden verschiedene Hilfsklassen, die für weitere XML Projekte eingesetzt werden können.

Ausblick

Bei jeder Definition eines XML Datenformats entsteht eine DTD Datei. Man sollte sich ein Konzept erarbeiten, das die Verwaltung der DTDs regelt. Wünschenswert wäre ein Verzeichnis für alle DTDs. Man sollte sich überlegen, wie Strukturanpassungen einer XML Datei gehandhabt werden. Es gibt zwei Möglichkeiten, erstens die DTD anzupassen oder zweitens eine neue DTD mit der neuen Struktur zu schreiben. Durch das `version` Attribut kann den XML Dateien eine Versionsnummer gegeben werden, was zum Beispiel zur Unterscheidung von verschiedenen XML Strukturen gebraucht werden kann.

In einem weiteren Schritt könnte das Repository auf mehrere XML Dateien aufgeteilt werden, zum Beispiel eine Datei pro Projekt. Dies würde eine Zusammenarbeit mit einem Versionsverwaltungssystem ermöglichen. XML unterstützt die Modularisierung von XML Dateien mittels externen Entitäten.

XML ist noch ziemlich neu und entwickelt sich rasch, zum Beispiel wurden während meiner Semesterarbeit zwei neue Standards definiert, XHTML 1.0 und eine Java Schnittstelle zur Instanzierung von XML Parsern. Die Entwicklung von XML sollte weiter verfolgt werden.

Persönliche Bemerkungen

Meine Ziele zu dieser Semesterarbeit waren: In einem Projekt mitzuarbeiten, eine vollständige Arbeit inklusive Dokumentation abzuliefern, das Zeit- und Projektmanagement zu erproben, XML zu erlernen und die Ideen der Vorlesung "Entwurf und Entwicklung von Softwaresystemen mit Java" von Dr. Dominik Gruntz in der Praxis umzusetzen. Ein Punkt in der Vorlesung von Dominik Gruntz war: Ein Programm wird viel öfter gelesen als geschrieben und deshalb ist beim Programmieren auf gute Lesbarkeit zu achten. Ich habe mich bemüht lesbaren Code zu schreiben.

Meine persönlichen Ziele zu dieser Semesterarbeit wurden erfüllt.

Martin Naedele und Jörn Janneck möchte ich danken für ihre gute Betreuung und Unterstützung.

Literaturverzeichnis

[1] Tim Bray et al., *Extensible Markup Language (XML) 1.0*, http://www.w3.org/TR/1998/REC-xml-19980210.html, 1998

[2] W3C, *XML Hompage*, http://www.w3.org/XML/

[3] W3C, *Recommendations*, http://www.w3.org/TR/

[4] Tim Bray, *The Annotated XML Specification*, http://www.xml.com/axml/axml.html, 1998

[5] Charles Goldfarb und Paul Prescod, *The XML Handbook*, Prentice Hall, 1998

[6] Organization for the Advancement of Structured Information Standards (OASIS), *Homepage*, http://www.oasis-open.org/cover/

[7] Edward A. Lee und Steve Neuendorffer, *MoML*, http://www.ptolemy.eecs.berkeley.edu/ptolemyII/ptII0.4/ptII0.4beta/doc/design/design.pdf

[8] A. Layman, *XML Syntax Recommendation for Serializing Graphs of Data*, http://www.w3.org/TandS/QL98/pp/microsoft-serializing.html, 1998

[9] Robin Cover, SGML/XML: Using Elements and Attributes, http://www.oasis-open.org/cover/elementsAndAttrs.html

[10] Eric Armstrong, XML Tutorial: *Working with XML*, http://java.sun.com/xml/tutorial_intro.html

[11] Nazmul Idris, *XML Tutorials*, http://www.developerlife.com

[12] Sun Microsystems, *Java API for XML Parsing 1.0* (Early Acces 1), http://developer.java.sun.com/developer/earlyAccess/xml/

[13] Sun Microsystems, XML Parser: *Java Project X Techology Release 2*, http://developer.java.sun.com/developer/products/xml/

[14] Oracle, *XML Parser for Java v2*, http://technet.oracle.com/tech/xml/parser_java2/

[15] IBM, ibm parser , *XML for Java Parser v2.0*, http://www.alphaworks.ibm.com/tech/xml4j

[16] apache.org, XML Parser: *Xerces-J 1.0.0*, `http://xml.apache.org/xerces-j/index.html`

[17] IBM alphaworks, *Homepage*, `http://www.alphaworks.ibm.com/tech`

[18] Oracle, *Homepage*, `http://technet.oracle.com/tech/xml/`

[19] David Megginson, *Simple API for XML*, `http://www.megginson.com/SAX/`

[20] Unicode Consortium, Unicode Standard, `http://www.unicode.org`

[21] International Organization for Standarization (ISO), *Standart Generalized Markup Language (SGML)*, ISO 8879, 1986

[22] Gamma et al., *Design Patterns*, Addison-Wesly, 1995

[23] H.-J. Schek, Vorlesung: *Informationssysteme Kernfach*, WS99/00, D-INFK

A. DTDs

A.1. repository.dtd

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

<!-- document type declaration for the moses repository -->
<!-- 5.12.1999 Roland Kurmann -->

<!ELEMENT   repository      (initdescriptor, descriptor*)>
<!ATTLIST   repository
            version         CDATA       #REQUIRED>
<!ELEMENT   initdescriptor  (entry*)>
<!ELEMENT   descriptor      (entry*)>
<!ATTLIST   descriptor
            id              ID          #REQUIRED>
<!ELEMENT   entry           (key, (ref|value|list))>
<!ELEMENT   key             (#PCDATA)>
<!ELEMENT   ref             EMPTY>
<!ATTLIST   ref
            id              IDREF       #REQUIRED>
<!ELEMENT   value           (#PCDATA)>
<!ATTLIST   value
            type            (string|object|class)   #REQUIRED>
<!ELEMENT   list            ((ref|value|list)*)>
```

Abbildung A.1.: repository.dtd

A.2. graphstorage.dtd

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

<!-- document type declaration for a graph storage -->
<!-- 14.1.2000 Roland Kurmann -->

<!ELEMENT   graphstorage      (graph)>
<!ATTLIST   graphstorage
            version           CDATA       #REQUIRED>
<!ELEMENT   graph             ((value|graph)*, vertex*, edge*)>
<!ATTLIST   graph
            name              CDATA       #REQUIRED>
<!ELEMENT   vertex            (value|graph)*>
<!ATTLIST   vertex
            id                ID          #REQUIRED>
<!ELEMENT   edge              (value|graph)*>
<!ATTLIST   edge
            A                 IDREF       #REQUIRED
            B                 IDREF       #REQUIRED>
<!ELEMENT   value             (#PCDATA)>
<!ATTLIST   value
            name              CDATA      #REQUIRED
            type              (integer|real|string|text|color|
                               type|expressionframe|expression|
                               parameters|parsed|extraparser|
                               boolean|double|point)     #REQUIRED
            parser            CDATA      #IMPLIED>
```

Abbildung A.2.: DTD für attributierte, verschachtelte Graphen

B. XML Beispiele von Moses

B.1. Repository

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<!-- Moses Repository -->
<!DOCTYPE repository SYSTEM "repository.dtd">

<repository version="1.0">

<initdescriptor>
  <entry>
    <key>ObjectManager_Attributes</key>
    <ref id="i0"/>
  </entry>
  <entry>
    <key>ClassManager_HierarchyRoot</key>
    <ref id="i1"/>
  </entry>
  <entry>
    <key>ObjectAdapters</key>
    <list>
      <value type="class">moses.repository.objecttypes.
                          simpletext.TextAdapter</value>
      <value type="class">moses.repository.objecttypes.
                          mosesformalism.MosesFormalismAdapter</value>
      <value type="class">moses.repository.objecttypes.
                          mosescomponent.MosesComponentAdapter</value>
      <value type="class">moses.repository.objecttypes.
                          simconf.SimulationConfigurationAdapter</value>
      <value type="class">moses.repository.objecttypes.
                          javacode.JavaCodeAdapter</value>
    </list>
  </entry>
  <entry>
    <key>PackageManager_Entries</key>
```

```
    <list>
    </list>
  </entry>
  <entry>
    <key>ProjectManager_projectRoot</key>
    <ref id="i2"/>
  </entry>
</initdescriptor>

<!-- ********************************** -->
<descriptor id="i1">
  <entry>
    <key>ClassManager_Name</key>
    <value type="string">Root</value>
  </entry>
  <entry>
    <key>ClassManager_Children</key>
    <list>
      <ref id="i5"/>
      <ref id="i6"/>
      <ref id="i7"/>
      . . .
      <ref id="i304"/>
    </list>
  </entry>
</descriptor>

. . .

<!-- ********************************** -->
<descriptor id="i25">
  <entry>
    <key>StorageFileName</key>
    <value type="string">Sieve21432</value>
  </entry>
  <entry>
    <key>ProjectManager_isProject</key>
    <value type="string">false</value>
  </entry>
  <entry>
    <key>MosesComponent_Type</key>
    <value type="string">/Common/Formalisms/TimePetriNet</value>
  </entry>
  <entry>
```

```
    <key>ModificationTime</key>
    <value type="string">27/08/1999 15:38:29</value>
  </entry>
  <entry>
    <key>ProjectManager_Name</key>
    <value type="string">Sieve</value>
  </entry>
  <entry>
    <key>hasContent</key>
    <value type="string">true</value>
  </entry>
  <entry>
    <key>ClassManager_Name</key>
    <value type="string">Sieve</value>
  </entry>
  <entry>
    <key>isNoExec</key>
    <value type="string">false</value>
  </entry>
  <entry>
    <key>ProjectManager_Parent</key>
    <ref id="i378"/>
  </entry>
  <entry>
    <key>isCompiled</key>
    <value type="string">true</value>
  </entry>
  <entry>
    <key>ClassManager_Parent</key>
    <ref id="i1"/>
  </entry>
  <entry>
    <key>PersistenceManager</key>
    <value type="object">moses.repository.objecttypes.mosescomponent.
                         MosesComponentPersistenceManager</value>
  </entry>
  <entry>
    <key>LockManager_Lock</key>
    <value type="string">unlocked</value>
  </entry>
  <entry>
    <key>isNoEdit</key>
    <value type="string">false</value>
  </entry>
```

```
  <entry>
    <key>MosesComponent_SourcePath</key>
    <value type="string">./storage/components</value>
  </entry>
  <entry>
    <key>ObjectAdapter</key>
    <value type="object">moses.repository.objecttypes.mosescomponent.
                         MosesComponentAdapter</value>
  </entry>
</descriptor>

<!-- ********************************** -->
<descriptor id="i387">
  <entry>
    <key>ProjectManager_Parent</key>
    <ref id="i376"/>
  </entry>
  <entry>
    <key>ProjectManager_Children</key>
    <list>
      <ref id="i70"/>
      <ref id="i71"/>
      <ref id="i72"/>
      <ref id="i73"/>
    </list>
  </entry>
  <entry>
    <key>ProjectManager_isProject</key>
    <value type="string">true</value>
  </entry>
  <entry>
    <key>ProjectManager_Name</key>
    <value type="string">HuffmanDecoder-2</value>
  </entry>
</descriptor>
</repository>
```

B.2. Graphen

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<!-- a moses graph -->
<!DOCTYPE graphstorage SYSTEM "graphstorage.dtd">

<graphstorage version="1.0">

<graph name="">
  <!-- special --><value name="TypeName"
                  type="string">/Common/Formalisms/TimePetriNet</value>
  <value name="Declarations" type="expressionframe"></value>
  <!-- special --><value name="PackageName" type="string"></value>
  <!-- special --><value name="geditor_scale" type="double">1.25</value>
  <!-- special --><value name="Name" type="string">Car</value>
  <value name="Parameters" type="parameters">id, dinit, vinit, computeV</value>

  <vertex id="i0">
    <!-- special --><value name="TypeName" type="string">OutputConnector</value>
    <!-- special --><value name="OffsetY" type="integer">-20</value>
    <!-- special --><value name="OffsetX" type="integer">0</value>
    <!-- special --><value name="NameShow" type="boolean">true</value>
    <value name="Name" type="string">arrivalMsg</value>
    <!-- special --><value name="OriginY" type="integer">41</value>
    <!-- special --><value name="OriginX" type="integer">133</value>
  </vertex>

  <vertex id="i1">
    <!-- special --><value name="TypeName" type="string">InputConnector</value>
    <!-- special --><value name="OffsetY" type="integer">0</value>
    <!-- special --><value name="OffsetX" type="integer">-56</value>
    <!-- special --><value name="NameShow" type="boolean">true</value>
    <value name="Name" type="string">finished</value>
    <!-- special --><value name="OriginY" type="integer">504</value>
    <!-- special --><value name="OriginX" type="integer">784</value>
  </vertex>

  <vertex id="i2">
    <!-- special --><value name="TypeName" type="string">Transition</value>
    <!-- special --><value name="NameShow" type="boolean">true</value>
    <!-- special --><value name="OriginY" type="integer">224</value>
    <!-- special --><value name="OriginX" type="integer">680</value>
  </vertex>
```

```
  <vertex id="i3">
    <!-- special --><value name="TypeName" type="string">InputConnector</value>
    <!-- special --><value name="OffsetY" type="integer">-20</value>
    <!-- special --><value name="OffsetX" type="integer">0</value>
    <!-- special --><value name="NameShow" type="boolean">true</value>
    <value name="Name" type="string">lightState</value>
    <!-- special --><value name="geditor_angle" type="integer">90</value>
    <!-- special --><value name="OriginY" type="integer">37</value>
    <!-- special --><value name="OriginX" type="integer">45</value>
  </vertex>

. . .

  <edge A="i4" B="i2">
    <!-- special --><value name="TypeName" type="string">Arc</value>
    <!-- special --><value name="OffsetY" type="integer">0</value>
    <!-- special --><value name="OffsetX" type="integer">0</value>
    <!-- special --><value name="geditor_intermediatePoints" type="point"></value>
    <!-- special --><value name="ConnectorB" type="string"></value>
    <!-- special --><value name="ConnectorA" type="string"></value>
    <!-- special --><value name="NameShow" type="boolean">true</value>
    <value name="Name" type="string">p</value>
    <!-- special --><value name="OriginY" type="integer">112</value>
    <!-- special --><value name="OriginX" type="integer">0</value>
  </edge>

  <edge A="i23" B="i0">
    <!-- special --><value name="TypeName" type="string">Arc</value>
    <!-- special --><value name="OffsetY" type="integer">0</value>
    <!-- special --><value name="OffsetX" type="integer">0</value>
    <!-- special --><value name="geditor_intermediatePoints" type="point"></value>
    <!-- special --><value name="ConnectorB" type="string"></value>
    <!-- special --><value name="ConnectorA" type="string"></value>
    <!-- special --><value name="NameShow" type="boolean">true</value>
    <value name="Name" type="string">msg</value>
  </edge>

</graph>

</graphstorage>
```

C. Konfigurationsdateien

C.1. Repository

RepositoryXML.properties

```
#Properties for RepositoryXML, RepositoryXMLDom and RepositoryXMLSax
#REK 14.1.2000

#tag and attribute names have to correspond with the DTD

#Tags used in the XML file
ROOT_TAG = repository
ROOT_VER_ATTR = version
INIT_DESC_TAG = initdescriptor
DESC_TAG = descriptor
DESC_ID_ATTR = id
ENTRY_TAG = entry
KEY_TAG = key
REF_TAG = ref
REF_ID_ATTR = id
VAL_TAG = value
VAL_TYPE_ATTR = type
VAL_TYPE_STR = string
VAL_TYPE_OBJ = object
VAL_TYPE_CLASS = class
LIST_TAG = list


#load

#DOM
#Specify a parser wrapper class (strategy pattern)
#DOM_PARSER = moses.xml.DOMParser_IBM_XML4J2
DOM_PARSER = moses.xml.DOMParser_Sun_XTR2
#DOM_PARSER = moses.xml.DOMParser_Oracle_v2
```

```
#DOM_PARSER = moses.xml.DOMParser_javax

#DocumentBuilderFactory in the case of using the moses.xml.DOMParser_javax
#wrapper class
#this property may be empty
JAVAX_XML_FACTORY = com.sun.xml.parser.DocumentBuilderFactoryImpl

#validating SAX parser
SAX_PARSER = com.sun.xml.parser.ValidatingParser
#SAX_PARSER = com.ibm.xml.parsers.ValidatingSAXParser
#Oracle is default nonvalidating
#SAX_PARSER = oracle.xml.parser.v2.SAXParser


#store

BACKUP_EXTENSION = .bak

#Specify a valid encoding for xml output
#e.g.: UTF-8, ISO-8859-1
WRITE_ENCODING = UTF-8

#Repository DTD filename
REP_DTD_FILE = repository.dtd

#Prefix for ID attributes in the XML file
#identifiers have to be valid XML names (e.g. character at the beginning)
ID_PREFIX = i

#Indentation characters for the xml elements
#space: \u0020
#tab: \t
#no indentation: empty
INDENT_STR = \u0020\u0020

#a Line between descriptors? (true/false)
LINE = true
```

repository.properties

In dieser Datei wird die Speicherstrategie für das Repository und der Dateiname des Repositorys bestimmt.

```
#MN 14.7.98
#Rob 19.7.98
#MN 22.7.98
#Rob 7.9.99
#REK 14.1.2000

#persistencemanager=moses.repository.structure.RepositoryXMLDom
persistencemanager=moses.repository.structure.RepositoryXMLSax
#persistencemanager=moses.repository.structure.RepositoryPlaintexter

#storagelocation is now obsolete
storagelocation=./demo.rep

#the repository is now stored relative to the userPath Preferences.getUserPath()
storagename=repository.xml
#storagename=demo.rep
```

C.2. Moseskomponenten

GraphXMLStorage.properties

```
#Properties for GraphXMLStorage
#REK 14.1.2000

#tag and attribute names have to correspond with the DTD graphstorage.dtd

#Tags used in the XML file
ROOT_TAG = graphstorage
ROOT_VER_ATTR = version
GRAPH_TAG = graph
GRAPH_NAME_ATTR = name
VERTEX_TAG = vertex
VERTEX_ID_ATTR = id
EDGE_TAG = edge
EDGE_A_ATTR = A
EDGE_B_ATTR = B
VAL_TAG = value
VAL_NAME_ATTR = name
VAL_TYPE_ATTR = type
VAL_PARSER_ATTR = parser

#types
TYPE_INTEGER = integer
TYPE_REAL = real
TYPE_STRING = string
TYPE_TEXT = text
TYPE_COLOR = color
TYPE_TYPE = type
TYPE_EXPRESSIONFRAME = expressionframe
TYPE_EXPRESSION = expression
TYPE_PARAMETERS = parameters
TYPE_PARSED = parsed
TYPE_EXTRAPARSER = extraparser
TYPE_BOOLEAN = boolean
TYPE_DOUBLE = double
TYPE_POINT = point

#constants
TRUE = true
FALSE = false
POINT_DELIM = ,
```

```
#load

#Specify a parser wrapper class (strategy pattern)
#DOM_PARSER = moses.xml.DOMParser_IBM_XML4J2
DOM_PARSER = moses.xml.DOMParser_Sun_XTR2
#DOM_PARSER = moses.xml.DOMParser_Oracle_v2
#DOM_PARSER = moses.xml.DOMParser_javax

#DocumentBuilderFactory in the case of using the moses.xml.DOMParser_javax
#wrapper class
#this property may be empty
JAVAX_XML_FACTORY = com.sun.xml.parser.DocumentBuilderFactoryImpl


#store

#Specify a valid encoding for xml output
#e.g.: UTF-8, ISO-8859-1
WRITE_ENCODING = UTF-8

#Repository DTD filename
GS_DTD_FILE = graphstorage.dtd

#Indentation characters for the xml elements
#space: \u0020
#tab: \t
#no indentation: empty
INDENT_STR = \u0020\u0020

#Prefix for ID attributes in the XML file
#identifiers have to be valid XML names (e.g. character at the beginning)
ID_PREFIX = i
```

MosesComponentAdapter.properties

Diese Datei wurde um die Einträge `importGraphXML` und `exportGraphXML` ergänzt.

```
Icons=LockedIcon:UnlockedIcon
LockedIcon=moses/repository/images/generic2.gif
UnlockedIcon=moses/repository/images/c.gif
ObjectPersistenceManager=moses.repository.objecttypes.mosescomponent.
    MosesComponentPersistenceManager

Actions=view:edit:unlock:compile:animate:importint:importRood:importGrail:
    exportGrail: importGraphXML:exportGraphXML
view=moses.repository.objecttypes.mosescomponent.ViewAction
edit=moses.repository.objecttypes.mosescomponent.EditAction
editDisableOptions=isNoEdit
unlock=moses.repository.objecttypes.UnlockAction
compile=moses.repository.objecttypes.mosescomponent.CompileAction
compileDisableOptions=isNoExec
animate=moses.repository.objecttypes.mosescomponent.AnimateAction
animateDisableOptions=isNoExec
simulate=moses.repository.objecttypes.mosescomponent.SimulateAction
simulateDisableOptions=isNoExec
importint=moses.repository.objecttypes.mosescomponent.ImportInternalAction
importintDisableOptions=isNoEdit
importRood=moses.repository.objecttypes.mosescomponent.ImportRoodAction
importRoodDisableOptions=isNoEdit
importGrail=moses.repository.objecttypes.mosescomponent.ImportGrailAction
importGrailDisableOptions=isNoEdit
exportGrail=moses.repository.objecttypes.mosescomponent.ExportGrailAction
importGraphXML=moses.repository.objecttypes.mosescomponent.ImportGraphXMLAction
importGraphXMLDisableOptions=isNoEdit
exportGraphXML=moses.repository.objecttypes.mosescomponent.ExportGraphXMLAction
#exportGraphXMLDisableOptions=isNoEdit
StorageLocationBase=./storage/components
DefaultPackage=

ClassPath=./classes
SourcePath=./source
```

C.3. RepositoryConverter

```
#Properties for RepositoryConverter
#REK 14.1.2000

#Input RepositoryPersitenceMangager
INPUT_MANAGER = moses.repository.structure.RepositoryPlaintexter

#Input File including path
INPUT_FILE = /home/rkurmann/moses/system/defaultUserData/demo.rep
#INPUT_FILE = c:\\projects\\moses\\moses\\system\\defaultUserData\\demo.rep

#Output RepsitoryPersitenceManager
OUTPUT_MANAGER = moses.repository.structure.RepositoryXMLSax

#Output File including path
OUTPUT_FILE = /home/rkurmann/moses/system/defaultUserData/repository.xml
#OUTPUT_FILE = c:\\projects\\moses\\moses\\system\\defaultUserData\\repository.xml


#the other direction
#INPUT_MANAGER = moses.repository.structure.RepositoryXMLSax
#INPUT_FILE = /home/rkurmann/moses/system/defaultUserData/test.repository.xml

#OUTPUT_MANAGER = moses.repository.structure.RepositoryPlaintexter
#OUTPUT_FILE = /home/rkurmann/moses/system/defaultUserData/test.demo.rep
```

D. Code

D.1. Repository

Auf den nächsten Seiten ist der Programmcode abgedruckt.

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.repository.structure;

import moses.xml.*;
import java.io.*;
import java.util.*;
import java.beans.*;

/**
 * This abstract class stores the repository using XML.
 * Only the storing is implemented, the load method is implemented
 * in subclasses.
 * <p>
 * The repository is passed as a dictionary of its entries.
 * <p>
 * The structure of the xml file is specified in the file "repository.dtd".
 * The tags and attribute names and other constants are stored in "RepositoryXML
.properties".
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created first version based on RepositoryPlaintexter.
java
 *
 * @see RepositoryPersistenceManager
 * @see RepositoryPlaintexter
 * @see Repository
 * @see IDStrategy
 */
abstract public class RepositoryXML
        implements RepositoryPersistenceManager
{

/** Repository structure version produced with store method. */
public static final String OUTPUT_VERSION = "1.0";

/** Holds the strategy for checking the input repository version. */
VersionStrategy version = null;

/** Holds the repository main root. */
Dictionary applications = null;

/** Holds the repository, used by load. */
Dictionary rep = null;

/** Holds the strategy for producing unique identifiers. */
IDStrategy id = null;
```

```
/** Holds path and name for the repository file. */
String pathAndName = null;

/** Holds the resource bundle with all the constants. */
ResourceBundle res = ResourceBundle.getBundle("RepositoryXML",
                                              Locale.getDefault());

// get tag names
final String ROOT_TAG = res.getString("ROOT_TAG");
final String ROOT_VER_ATTR = res.getString("ROOT_VER_ATTR");
final String INIT_DESC_TAG = res.getString("INIT_DESC_TAG");
final String DESC_TAG = res.getString("DESC_TAG");
final String DESC_ID_ATTR = res.getString("DESC_ID_ATTR");
final String ENTRY_TAG = res.getString("ENTRY_TAG");
final String KEY_TAG = res.getString("KEY_TAG");
final String REF_TAG = res.getString("REF_TAG");
final String REF_ID_ATTR = res.getString("REF_ID_ATTR");
final String VAL_TAG = res.getString("VAL_TAG");
final String VAL_TYPE_ATTR = res.getString("VAL_TYPE_ATTR");
final String VAL_TYPE_STR = res.getString("VAL_TYPE_STR");
final String VAL_TYPE_OBJ = res.getString("VAL_TYPE_OBJ");
final String VAL_TYPE_CLASS = res.getString("VAL_TYPE_CLASS");
final String LIST_TAG = res.getString("LIST_TAG");

/**
 * Prefix for xml identifiers.
 * Identifiers must start with a letter.
 */
final String ID_PREFIX = res.getString("ID_PREFIX");

/** Writes the output with this encoding, e.g.: UTF-8 */
final String WRITE_ENCODING = res.getString("WRITE_ENCODING");

/** Name of the dtd file used for output. */
final String REP_DTD_FILE = res.getString("REP_DTD_FILE");

final String BACKUP_EXTENSION = res.getString("BACKUP_EXTENSION");

/** Indentation added in each level. */
final String INDENT_STR = res.getString("INDENT_STR");

/** A Line between descriptors? */
final boolean LINE = res.getString("LINE").equals("true");

/**
 * Set the storage location.
 *
 * @param pathAndName Platform dependent string specifying the
 * storage location completely.
 */
public void setPathAndName(String pathAndName) {
    this.pathAndName = pathAndName;
}

/**
 * Returns the storage location.
 */
public String getPathAndName() {
    return pathAndName;
}

/**
 * Write the repository to a file, specified with setPathAndName.
 * The old file will be backuped.
 * Characters reserved for xml such as &gt; will be converted
 * in predefined entities.
 * Identifiers for the different objects were generated through an IDStrategy.
 *
 * @param rep The repository to be stored.
```

```
     * @throws IOException Exceptions not handled here were reported as
     * IOExceptions.
     */
    public void store(Dictionary rep) throws IOException {
        System.out.println("Store repository");
        // backup the old file
        File origFile = new File(pathAndName);
        File bakFile = new File(pathAndName + BACKUP_EXTENSION);
        origFile.renameTo(bakFile);

        // id strategy
        id = new CountID();

        // output writer
        PrintWriter pw = new PrintWriter(
                         new BufferedWriter(
                         new OutputStreamWriter(
                         new FileOutputStream(pathAndName), WRITE_ENCODING)));

        // write xml prolog
        pw.print("<?xml version='1.0'");
        pw.println(" encoding='" + WRITE_ENCODING + "' standalone='no'?>");
        pw.println("<!-- Moses Repository -->");
        pw.println("<!DOCTYPE " + ROOT_TAG + " SYSTEM \"" + REP_DTD_FILE + "\">");
        pw.println();

        // write root tag
        pw.println("<" + ROOT_TAG + " "
                    + ROOT_VER_ATTR + "=\"" + OUTPUT_VERSION + "\">");
        pw.println();

        // write InitDescriptor
        pw.println("<" + INIT_DESC_TAG + ">");
        Descriptor initDesc = (Descriptor) rep.get(Repository.INIT_DESCRIPTOR);
        writeEntries(initDesc, INDENT_STR, pw);
        pw.println("</" + INIT_DESC_TAG + ">");

        // go through all descriptors, except the Initdescriptor
        for (Enumeration en = rep.keys(); en.hasMoreElements(); ) {
            Object key = en.nextElement();
            if (!(key instanceof String
                    && key.equals(Repository.INIT_DESCRIPTOR))) {
                Descriptor d = (Descriptor) rep.get(key);
                pw.println();
                if (LINE) {
                    pw.println("<!-- ********************************** -->");
                }
                pw.print("<" + DESC_TAG + " " + DESC_ID_ATTR );
                pw.println("=\"" + ID_PREFIX + id.get(d) + "\">");
                writeEntries(d, INDENT_STR, pw);
                pw.println("</" + DESC_TAG + ">");
            } else {
                // don't write InitDescriptor
            }
        }

        pw.println("</" + ROOT_TAG + ">");

        pw.flush();
        pw.close();
    }

    /**
     * Write all entries of the Descriptor.
     * An entry consists of a key and some data.
     *
     * @param d The Descriptor to be stored.
     * @param indent Indentation used by this recursion.
     * @param pw PrintWriter for the output data.
     */
```

```
void writeEntries(Descriptor d, String indent, PrintWriter pw) {
    try {
        for (Enumeration en = d.keys(); en.hasMoreElements(); ) {
            Object o = en.nextElement();
            Object content = d.get(o);

            pw.print(indent);
            pw.println("<" + ENTRY_TAG + ">");

            pw.print(indent + INDENT_STR);
            pw.print("<" + KEY_TAG + ">");

            pw.print(Xml.normalize(o.toString()));
            pw.println("</" + KEY_TAG + ">");

            writeData(content, indent + INDENT_STR, pw);

            pw.print(indent);
            pw.println("</" + ENTRY_TAG + ">");
        }
    } catch (PropertyVetoException e) {
        e.printStackTrace();
    }
}

/**
 * Writes the data stored in content with its tag.
 *
 * @param content The data to be stored.
 * @param indent Indentation used by this recursion.
 * @param pw PrintWriter for the output data.
 * @throws PropertyVetoException
 */
void writeData(Object content, String indent, PrintWriter pw)
        throws PropertyVetoException {
    if (content instanceof String) {
        pw.print(indent);
        pw.println("<" + VAL_TAG + " " + VAL_TYPE_ATTR + "=\""
                   + VAL_TYPE_STR + "\">"
                   + Xml.normalize((String) content) + "</" + VAL_TAG + ">");
    } else if (content instanceof Descriptor) {
        pw.print(indent);
        pw.println("<" + REF_TAG + " " + REF_ID_ATTR + "=\""
                   + ID_PREFIX + id.get(content) +"\"/>");
    } else if (content instanceof Class) {
        pw.print(indent);
        pw.println("<" + VAL_TAG + " "
                   + VAL_TYPE_ATTR + "=\"" + VAL_TYPE_CLASS+ "\">"
                   + ((Class) content).getName() + "</" + VAL_TAG + ">");
    } else if (content instanceof Vector) {
        pw.print(indent);
        pw.println("<" + LIST_TAG + ">");
        for (Iterator it = ((Vector) content).iterator(); it.hasNext(); ) {
            Object listItem = it.next();
            // recursive call for list items
            writeData(listItem, indent + INDENT_STR, pw);
        }
        pw.print(indent);
        pw.println("</" + LIST_TAG + ">");
    } else {
        // last possibility, content is an Object and store the class name
        pw.print(indent);
        pw.println("<" + VAL_TAG + " "
                   + VAL_TYPE_ATTR + "=\"" + VAL_TYPE_OBJ + "\">"
                   + content.getClass().getName() + "</" + VAL_TAG + ">");
    }
}

/**
 * Load the dictionary from a file. DescriptorConnectable classes are
```

```
     * automatically connected.
     *
     * @param rep The dictionary to be stored.
     *
     * @throws IOException
     * @throws ClassNotFoundException
     * @throws PropertyVetoException
     */
    abstract public Dictionary load(Dictionary applications)
            throws IOException, ClassNotFoundException, PropertyVetoException;


    }
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.repository.structure;

import moses.xml.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.reflect.*;
import java.beans.*;
// XML interfaces
import org.w3c.dom.*;
import org.xml.sax.*;

/**
 * This class loads a repository form a file using a validating DOM parser and D
OM tree.
 * The different DOM parsers were instantiated through a DOM parser strategy.
 * The store method is inherited.
 * <p>
 * The structure of the xml file is specified in the file "repository.dtd".
 * The tags and attribute names and other constants are stored in "RepositoryXML
.properties".
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 *
 * @see RepositoryPersistenceManager
 * @see RepositoryPlaintexter
 * @see Repository
 * @see DOMParserWrapper
 * @see RepositoryXMLSax
 */
public class RepositoryXMLDom
        extends RepositoryXML
{

/** Repository xml file version accepted by this load. */
final String ACCEPT_VERSION = "1.0";

/** Stores the id's and the descriptors. */
Map refs = null;

/** Property name for the java DOM parser builder factory. */
final String JAVAX_XML_PROPERTY = "javax.xml.parsers.DocumentBuilderFactory";

/** Use this DOM parser wrapper in the strategy pattern. */
```

```
final String DOM_PARSER = res.getString("DOM_PARSER");
/** Java parser builder factory. */
final String JAVAX_XML_FACTORY = res.getString("JAVAX_XML_FACTORY");

/**
 * Load the dictionary from a file. DescriptorConnectable classes are
 * automatically connected.
 * The repsository xml file version is validated during the loading process.
 *
 * @param rep The dictionary to be stored
 *
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws PropertyVetoException
 */
public Dictionary load(Dictionary applications)
        throws IOException, ClassNotFoundException, PropertyVetoException {
    System.out.println("Load Repository XMLDom");
    this.applications = applications;
    rep = new Hashtable();
    refs = new HashMap();
    // strategy for checking the input version of the repository file
    version = new ExactVersion(ACCEPT_VERSION);
    try {
        // setting of the java xml extension factory builder
        System.setProperty(JAVAX_XML_PROPERTY, JAVAX_XML_FACTORY);
        // set DOM parser
        DOMParserWrapper parser =
            (DOMParserWrapper) Class.forName(DOM_PARSER).newInstance();
        URL url = (new File(pathAndName)).toURL();
        Document doc = parser.parse(url);
        Element rootEl = (Element) doc.getElementsByTagName(ROOT_TAG).item(0);
        Element initEl = (Element)
                rootEl.getElementsByTagName(INIT_DESC_TAG).item(0);
        Descriptor initDesc = new Descriptor();
        NodeList descNl = rootEl.getElementsByTagName(DESC_TAG);
        int size = descNl.getLength();

        if (!version.valid(rootEl.getAttribute(ROOT_VER_ATTR))) {
            throw new IOException("wrong repository version!");
        }

        rep.put(Repository.INIT_DESCRIPTOR, initDesc);

        // get all descriptors
        for (int i = 0; i < size; i++) {
            Element el = (Element) descNl.item(i);
            Descriptor d = new Descriptor();
            rep.put(d, d);
            refs.put(el.getAttribute(DESC_ID_ATTR), d);
        }

        // fill InitDescriptor
        getEntries(initDesc, initEl);

        // fill all Descriptors
        for (int i = 0; i < size; i++) {
            Element el = (Element) descNl.item(i);
            Descriptor d =
                (Descriptor) refs.get(el.getAttribute(DESC_ID_ATTR));
            getEntries(d, el);
        }
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (SAXParseException e) {
        System.err.println("SystemId:" + e.getSystemId());
        System.err.println("PublicId: " + e.getPublicId());
        System.err.println("Line: " + e.getLineNumber()
                           + " Col: " + e.getColumnNumber());
        if (e.getException() != null) {
```

```
            System.err.println("inner: " + e.getException());
            System.err.println("message: " + e.getMessage());
        }
        e.printStackTrace();
    } catch (SAXException e) {
        if (e.getException() != null) {
            System.err.println("inner: " + e.getException());
            System.err.println("message: " + e.getMessage());
        }
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (ClassCastException e) {
        e.printStackTrace();
    }
    return rep;
}

/**
 * Fills a Descriptor using the DOM element.
 *
 * @param d The current Descriptor.
 * @param el The current Element from the DOM tree.
 *
 * @throws IOException Reports an invalid Descriptor reference.
 * @throws ClassNotFoundException Reports the missing of class files
 * specified in the repository.
 * @throws PropertyVetoException Access not allowed.
 */
void getEntries(Descriptor d, Element el)
        throws IOException, ClassNotFoundException, PropertyVetoException {
    NodeList nl = el.getElementsByTagName(ENTRY_TAG);
    for (int i = 0; i < nl.getLength(); i++) {
        Element entry = (Element) nl.item(i);
        // there are maybe other Nodes, hence it's impossible
        // to use the methods firstChild() and lastChild()

        // get the key data
        Element keyEl = (Element) entry.getElementsByTagName(KEY_TAG).item(0);

        // look for the second element with data
        // the DTD ensures a second element!
        Node n = keyEl.getNextSibling();
        while (!(n instanceof Element)) {
            n = n.getNextSibling();
        }
        Element data = (Element) n;
        String key = keyEl.getFirstChild().getNodeValue();

        d.set(key, getData(data, d));
    }
}

/**
 * Processes an element of the DOM tree
 * and returns the new object.
 * The data objects can refer to the whole repository through
 * the field applications.
 *
 * @param data The current element node from the DOM tree.
 * @param d Current Descriptor
 * , needed sometimes for connection with the data object.
 * @throws IOException Reports an invalid Descriptor reference.
 * @throws ClassNotFoundException Reports the missing of class files
 * specified in the repository.
 *
 * @return Data object from the repository.
 */
```

```
Object getData(Element data, Descriptor d)
        throws IOException, ClassNotFoundException {
    Object val = null;
    if (data.getTagName().equals(VAL_TAG)) {
        if (data.getAttribute(VAL_TYPE_ATTR).equals(VAL_TYPE_STR)) {
            if (data.hasChildNodes()) {
                val = data.getFirstChild().getNodeValue();
            } else {
                val = "";
            }
        } else if (data.getAttribute(VAL_TYPE_ATTR).equals(VAL_TYPE_OBJ)) {
            try {
                Class cl = Class.forName(data.getFirstChild().getNodeValue());
                Collection interfaces = Arrays.asList(cl.getInterfaces());
                boolean appcl = interfaces.contains(ApplicationClient.class);
                boolean descon = interfaces.contains(DescriptorConnectable.class
);
                if (appcl && descon) {
                    Class[] ctorpara = {Descriptor.class, Dictionary.class};
                    Constructor ctor = cl.getConstructor(ctorpara);
                    Object[] instpara = {d, applications};
                    val = ctor.newInstance(instpara);
                } else if (!appcl && descon) {
                    Class[] ctorpara = {Descriptor.class};
                    Constructor ctor = cl.getConstructor(ctorpara);
                    // is there a specalized constructor?
                    // (an interface can't say anything)
                    if (ctor != null) {
                        Object[] instpara = {applications};
                        val = ctor.newInstance(instpara);
                    } else {
                        val = cl.newInstance();
                        ((DescriptorConnectable) val).setDescriptor(d);
                    }
                } else if (appcl && !descon) {
                    val = cl.newInstance();
                    ((ApplicationClient) val).setApplications(applications);
                } else {
                    val = cl.newInstance();
                }
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (ClassCastException e) {
                e.printStackTrace();
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
        } else if (data.getAttribute(VAL_TYPE_ATTR).equals(VAL_TYPE_CLASS)) {
            try {
                val = Class.forName(data.getFirstChild().getNodeValue());
            } catch (ClassCastException e) {
                e.printStackTrace();
            }
        }
    } else if (data.getTagName().equals(REF_TAG)) {
        val = refs.get(data.getAttribute(REF_ID_ATTR));
        if (val == null) {
            throw new IOException("Descriptor not found");
        }
    } else if (data.getTagName().equals(LIST_TAG)) {
        NodeList nl = data.getChildNodes();
        Vector v = new Vector();
        for (int i = 0; i < nl.getLength(); i++) {
            Node n = nl.item(i);
            // ignore the other nodes, e.g. empty text nodes
            if (n instanceof Element) {
```

```
                // recursive call inoder to get the content
                v.add(getData((Element) n, d));
            }
        }
        val = v;
    }
    return val;
}

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.repository.structure;

import moses.xml.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.reflect.*;
import java.beans.*;
// XML interfaces
import org.xml.sax.*;
import org.xml.sax.helpers.*;

/**
 * This class loads a repository form a file using a validating SAX parser.
 * The load process is implemented as one pass parser.
 * This class processes
 * The store method is inherited.
 * <p>
 * The structure of the xml file is specified in the file "repository.dtd".
 * The tags and attribute names and other constants are stored in "RepositoryXML
.properties".
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 *
 * @see RepositoryPersistenceManager
 * @see RepositoryPlaintexter
 * @see Repository
 * @see RepositoryXMLDom
 */
public class RepositoryXMLSax
        extends RepositoryXML
        implements DocumentHandler, ErrorHandler
{

/** Repository xml file version accepted by this load. */
final String ACCEPT_VERSION = "1.0";

/** Stores the id's and the descriptors. */
Map refs = null;

// SAX specific variables

/** Stack for nested list tags. */
Stack stack = null;
```

```
// The structures aren't nested, so one variable is enough.
/** Holds the current string. */
String curStr = null;
/** Holds the current reference identifier. */
String curRefId = null;
/** Holds the current descriptor. */
Descriptor curDesc = null;
/** Holds the current VAL_TYPE_ATTR. */
String curValType = null;
/** Holds the current KEY_TAG. */
String curKey = null;
/** Holds the current data object. */
Object curData = null;

/** SAX parser class name for parser factory. */
final String SAX_PARSER = res.getString("SAX_PARSER");

/**
 * Load the dictionary from a file. DescriptorConnectable classes are
 * automatically connected.
 * <p>
 * The repository xml file version is validated during the loading process.
 *
 * @param rep The dictionary to be stored
 *
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws PropertyVetoException
 */
public Dictionary load(Dictionary applications)
        throws IOException, ClassNotFoundException, PropertyVetoException {
    System.out.println("Load Repository XMLSax");
    this.applications = applications;
    version = new ExactVersion(ACCEPT_VERSION);
    rep = new Hashtable();
    refs = new HashMap();
    stack = new Stack();
    try {
        URL url = (new File(pathAndName)).toURL();
        Parser parser = ParserFactory.makeParser(SAX_PARSER);
        parser.setErrorHandler(this);
        parser.setDocumentHandler(this);
        parser.parse(url.toExternalForm());
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (SAXParseException e) {
        System.err.println("SystemId:" + e.getSystemId());
        System.err.println("PublicId: " + e.getPublicId());
        System.err.println("Line: " + e.getLineNumber()
                           + " Col: " + e.getColumnNumber());
        if (e.getException() != null) {
            System.err.println("inner: " + e.getException());
            System.err.println("message: " + e.getMessage());
        }
        e.printStackTrace();
    } catch (SAXException e) {
        if (e.getException() != null) {
            System.err.println("inner: " + e.getException());
            System.err.println("message: " + e.getMessage());
        }
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (ClassCastException e) {
        e.printStackTrace();
    }
    return rep;
```

```
}


//*************************************************************************
// DocumentHandler interface implementation

/**
 * Do nothing.
 */
public void startDocument() throws SAXException {
    // do nothing
}

/**
 * Do nothing.
 */
public void endDocument() throws SAXException {
    // do nothing
}

/**
 * Called for each new tag.
 * Mostly, remember last tag and attribute name.
 * @throws SAXException Wrapper exception for all exceptions.
 */
public void startElement(String name, AttributeList attrs) throws SAXException {
    if (name.equals(ROOT_TAG)) {
        if (!version.valid(attrs.getValue(ROOT_VER_ATTR))) {
            throw new SAXException("moses: wrong repository version!");
        }
    } else if (name.equals(INIT_DESC_TAG)) {
        curDesc = new Descriptor();
        rep.put(Repository.INIT_DESCRIPTOR, curDesc);
    } else if (name.equals(DESC_TAG)) {
        String id = attrs.getValue(DESC_ID_ATTR);
        // descriptor may be created in advance
        if (!refs.containsKey(id)) {
            Descriptor desc = new Descriptor();
            refs.put(id, desc);
            rep.put(desc, desc);
        }
        curDesc = (Descriptor) refs.get(id);
    } else if (name.equals(ENTRY_TAG)) {
    } else if (name.equals(KEY_TAG)) {
    } else if (name.equals(REF_TAG)) {
        curRefId = attrs.getValue(REF_ID_ATTR);
    } else if (name.equals(VAL_TAG)) {
        curValType = attrs.getValue(VAL_TYPE_ATTR);
    } else if (name.equals(LIST_TAG)) {
        stack.push(new Vector());
    } else {
        throw new SAXException("unknown Tag");
    }
}

/**
 * Read the data between the tags.
 * Data may be fragmented and has to be recomposed.
 * @throws SAXException Wrapper exception for all exceptions.
 */
public void characters(char buf [], int offset, int len) throws SAXException {
    // characters may be delivered in different chunks
    // curStr == null == new string
    if (curStr == null) {
        curStr = new String(buf, offset, len);
    } else {
        // recompose
        curStr += new String(buf, offset, len);
    }
}
```

```
/**
 * Do nothing.
 */
public void ignorableWhitespace(char[] ch, int start, int length) {
    // do nothing
}

/**
 * Called for each end of a tag. The main work is done in this method.
 * This method uses all the current data and the stack.
 *
 * @throws SAXException Wrapper exception for all exceptions.
 */
public void endElement(String name) throws SAXException {
    if (name.equals(ROOT_TAG)) {
    } else if (name.equals(INIT_DESC_TAG)) {
        // for safety
        curDesc = null;
    } else if (name.equals(DESC_TAG)) {
        // for safety
        curDesc = null;
    } else if (name.equals(ENTRY_TAG)) {
        try {
            curDesc.set(curKey, curData);
        } catch (PropertyVetoException e) {
            throw new SAXException(e);
        }
        // for safety
        curKey = null;
        curData = null;
    } else if (name.equals(KEY_TAG)) {
        curKey = curStr;
        // for safety
        curStr = null;
    } else if (name.equals(REF_TAG)) {
        // if the target descriptor don't exist (forward reference)
        // create the descriptor in advance
        if (!refs.containsKey(curRefId)) {
            Descriptor d = new Descriptor();
            refs.put(curRefId, d);
            rep.put(d, d);
        }
        Descriptor desc = (Descriptor) refs.get(curRefId);
        // stack == empty == no nested structure
        if (stack.empty()) {
            curData = desc;
        } else {
            ((Vector) stack.peek()).add(desc);
        }
        // for safety
        curRefId = null;
    } else if (name.equals(VAL_TAG)) {
        Object val = null;
        Descriptor d = curDesc;
        if (curValType.equals(VAL_TYPE_STR)) {
            val = curStr;
        } else if (curValType.equals(VAL_TYPE_OBJ)) {
            try {
                Class cl = Class.forName(curStr);
                Collection interfaces = Arrays.asList(cl.getInterfaces());
                boolean appcl = interfaces.contains(ApplicationClient.class);
                boolean descon = interfaces.contains(DescriptorConnectable.class
);
                if (appcl && descon) {
                    Class[] ctorpara = {Descriptor.class, Dictionary.class};
                    Constructor ctor = cl.getConstructor(ctorpara);
                    Object[] instpara = {d, applications};
                    val = ctor.newInstance(instpara);
                } else if (!appcl && descon) {
```

```
                    Class[] ctorpara = {Descriptor.class};
                    Constructor ctor = cl.getConstructor(ctorpara);
                    // is there a specalized constructor?
                    // (an interface can't say anything)
                    if (ctor != null) {
                        Object[] instpara = {applications};
                        val = ctor.newInstance(instpara);
                    } else {
                        val = cl.newInstance();
                        ((DescriptorConnectable) val).setDescriptor(d);
                    }
                } else if (appcl && !descon) {
                    val = cl.newInstance();
                    ((ApplicationClient) val).setApplications(applications);
                } else {
                    val = cl.newInstance();
                }
            } catch (ClassNotFoundException e) {
                throw new SAXException(e);
            } catch (InstantiationException e) {
                throw new SAXException(e);
            } catch (IllegalAccessException e) {
                throw new SAXException(e);
            } catch (ClassCastException e) {
                throw new SAXException(e);
            } catch (NoSuchMethodException e) {
                throw new SAXException(e);
            } catch (InvocationTargetException e) {
                throw new SAXException(e);
            }
        } else if (curValType.equals(VAL_TYPE_CLASS)) {
            try {
                val = Class.forName(curStr);
            } catch (ClassNotFoundException e) {
                throw new SAXException(e);
            } catch (ClassCastException e) {
                throw new SAXException(e);
            }
        }

        // stack == empty == no nested structure
        if (stack.empty()) {
            curData = val;
        } else {
            ((Vector) stack.peek()).add(val);
        }
        //for safety
        curValType = null;
        curStr = null;
    } else if (name.equals(LIST_TAG)) {
        Vector list = (Vector) stack.pop();
        // stack == empty == no nested structure
        if (stack.empty()) {
            curData = list;
        } else {
            ((Vector) stack.peek()).add(list);
        }
    } else {
        throw new SAXException("unknown Tag");
    }
}

/**
 * Do nothing.
 */
public void processingInstruction(String target, String data) {
    // do nothing
}

/**
```

```
     * Do nothing.
     */
    public void setDocumentLocator(Locator locator) {
        // do nothing
    }


    //============================================================================
    // ErrorHandler interface implementation

    /**
     * Report all SAX warnings.
     *
     * @exception SAXParseException
     */
    public void warning(SAXParseException e) throws SAXParseException {
        throw e;
    }

    /**
     * Report all SAX errors.
     *
     * @exception SAXParseException
     */
    public void error(SAXParseException e) throws SAXParseException {
        throw e;
    }

    /**
     * Report all SAX fatal errors.
     *
     * @exception SAXParseException
     */
    public void fatalError(SAXParseException e) throws SAXParseException {
        throw e;
    }

    }
```

D.2. Graphen

Auf den nächsten Seiten ist der Programmcode abgedruckt.

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.repository.objecttypes.mosescomponent;

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.beans.*;
//moses
import moses.repository.objecttypes.*;
import graph.*;
import graph.util.*;
import graph.graphtype.*;
import moses.xml.*;
// XML interfaces
import org.w3c.dom.*;
import org.xml.sax.*;

/**
 * This class manages the persistence of graphs using xml.
 * The graphs may be nested.
 * Informations about graphs are fetched through a GraphTypeFinder class.
 * <p>
 * The DTD in the loading process is specified in xml file.
 * For xml files with a relative dtd path, e.g. the filename, you have to
 * specify the xml file either in the consturctor or the method setXMLFile.
 * <p>
 * The structure of the xml file is specified in the file "graphstorage.dtd".
 * The tags and attribute names and other constants are stored in "GraphXMLStora
ge.properties".
 * <p>
 * The load method uses a DOM tree for rebuilding a graph.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 *
 * @see Graph
 * @see GraphType
 * @see Vertex
 * @see VertexType
 * @see Edge
 * @see EdgeType
 * @see AttributeSpec
 * @see GraphConstants
 * @see GraphTypeFinder
 * @see DOMParserWrapper
```

```
 * @see Xml
 */
public class GraphXMLStorage
        implements ObjectXMLStorageStrategy
{

/** Graphstorage structure version produced with store method. */
public static final String OUTPUT_VERSION = "1.0";

/** Current XML file. */
URL xmlFile = null;

/** Holds the strategy for producing unique identifiers. */
IDStrategy ID = new CountID();

/** Holds the strategy for checking the input graphstorage version. */
VersionStrategy version = new ExactVersion("1.0");

/** Holds a GraphTypeFinder for getting type informations about graphs. */
GraphTypeFinderInterface graphTypeFinder = null;

ResourceBundle res = ResourceBundle.getBundle("GraphXMLStorage",
                                              Locale.getDefault());

// tag and attribute names
final String ROOT_TAG = res.getString("ROOT_TAG");
final String ROOT_VER_ATTR = res.getString("ROOT_VER_ATTR");
final String GRAPH_TAG = res.getString("GRAPH_TAG");
final String GRAPH_NAME_ATTR = res.getString("GRAPH_NAME_ATTR");
final String VERTEX_TAG = res.getString("VERTEX_TAG");
final String VERTEX_ID_ATTR = res.getString("VERTEX_ID_ATTR");
final String EDGE_TAG = res.getString("EDGE_TAG");
final String EDGE_A_ATTR = res.getString("EDGE_A_ATTR");
final String EDGE_B_ATTR = res.getString("EDGE_B_ATTR");
final String VAL_TAG = res.getString("VAL_TAG");
final String VAL_NAME_ATTR = res.getString("VAL_NAME_ATTR");
final String VAL_TYPE_ATTR = res.getString("VAL_TYPE_ATTR");
final String VAL_PARSER_ATTR = res.getString("VAL_PARSER_ATTR");

// type names
final String TYPE_INTEGER = res.getString("TYPE_INTEGER");
final String TYPE_REAL = res.getString("TYPE_REAL");
final String TYPE_STRING = res.getString("TYPE_STRING");
final String TYPE_TEXT = res.getString("TYPE_TEXT");
final String TYPE_COLOR = res.getString("TYPE_COLOR");
final String TYPE_TYPE = res.getString("TYPE_TYPE");
final String TYPE_EXPRESSIONFRAME = res.getString("TYPE_EXPRESSIONFRAME");
final String TYPE_EXPRESSION = res.getString("TYPE_EXPRESSION");
final String TYPE_PARAMETERS = res.getString("TYPE_PARAMETERS");
final String TYPE_PARSED = res.getString("TYPE_PARSED");
final String TYPE_EXTRAPARSER = res.getString("TYPE_EXTRAPARSER");
final String TYPE_BOOLEAN = res.getString("TYPE_BOOLEAN");
final String TYPE_DOUBLE = res.getString("TYPE_DOUBLE");
final String TYPE_POINT = res.getString("TYPE_POINT");

final String TRUE = res.getString("TRUE");
final String FALSE = res.getString("FALSE");
final String POINT_DELIM = res.getString("POINT_DELIM");

/**
 * Prefix for xml identifiers.
 * Identifiers must start with a letter.
 */
final String ID_PREFIX = res.getString("ID_PREFIX");

/** Writes the output with this encoding, e.g.: UTF-8 */
final String WRITE_ENCODING = res.getString("WRITE_ENCODING");

/** Use this DOM parser in the strategy pattern. */
final String DOM_PARSER = res.getString("DOM_PARSER");
```

```
/** Name of the dtd file used for output. */
final String GS_DTD_FILE = res.getString("GS_DTD_FILE");

/** Indentation added in each level. */
final String INDENT_STR = res.getString("INDENT_STR");

/**
 * Create a GraphXMLStorage that uses a GraphTypeFinder.
 *
 * @param graphTypeFinder GraphTypeFinder class.
 */
public GraphXMLStorage(GraphTypeFinderInterface graphTypeFinder) {
    this.graphTypeFinder = graphTypeFinder;
}

/**
 * Create a GraphXMLStorage that uses a GraphTypeFinder and specifies the xml fi
le.
 *
 * @param graphTypeFinder GraphTypeFinder class.
 * @param file XML file.
 */
public GraphXMLStorage(GraphTypeFinderInterface graphTypeFinder, URL file) {
    this.graphTypeFinder = graphTypeFinder;
    setXMLFile(file);
}

/**
 * Set the XML file.
 *
 * @param file XML file.
 */
public void setXMLFile(URL file) {
    this.xmlFile = file;
}

/**
 * Get the XML file.
 *
 * @return XML file.
 */
public URL getXMLFile() {
    return xmlFile;
}

/**
 * Store a graph in xml file using an output stream.
 *
 * @param outputStream Write the graph to this OutputStream.
 * @param o Graph to store.
 */
public void store(OutputStream outputStream, Object o) throws IOException {
    System.out.println("Store graph");
    Graph graph = (Graph) o;

    PrintWriter pw = new PrintWriter(
                     new BufferedWriter(
                     new OutputStreamWriter(
                     outputStream, WRITE_ENCODING)));

    // write xml prolog
    pw.print("<?xml version='1.0'");
    pw.println(" encoding='" + WRITE_ENCODING + "' standalone='no'?>");
    pw.println("<!-- a moses graph -->");
    pw.println("<!DOCTYPE " + ROOT_TAG + " SYSTEM \"" + GS_DTD_FILE + "\">");
    pw.println();

    // write root tag
    pw.print("<" + ROOT_TAG + " ");
```

```
    pw.println(ROOT_VER_ATTR + "=\"" + OUTPUT_VERSION + "\">");
    writeGraph(graph, "", "", pw);
    pw.println("</" + ROOT_TAG + ">");

    pw.flush();
    pw.close();
}

/**
 * Write a graph to the PrintWriter. This method is prepared for recursive calls
.
 * Looks for type informations about the graph. If no type informations is
 * available, work only with the primitive JAVA data types.
 *
 * @param graph The graph to be stored.
 * @param name Key name of the attribute (Moses terminology) with this graph,
 * else "".
 * @param indent Indentation used by this recursion.
 * @param pw PrintWriter for the output data.
 */
void writeGraph(Graph graph, String name, String indent, PrintWriter pw)
        throws IOException {
    GraphType graphType = null;
    try {
        String graphTypeName = (String) graph.getType();
        graphType = graphTypeFinder.getGraphType(graphTypeName);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        return;
    } catch (PropertyVetoException e) {
        e.printStackTrace();
        return;
    } catch (NullPointerException e) {
        // the graph don't have a type
        graphType = null;
    }

    pw.println();
    pw.print(indent);
    pw.print("<" + GRAPH_TAG + " ");
    pw.println(GRAPH_NAME_ATTR + "=\"" + name + "\">");

    // graph values
    for(Iterator it = graph.keySet().iterator(); it.hasNext(); ) {
        String key = (String) it.next();
        writeValue(key, graph, graphType, indent + INDENT_STR, pw);
    }
    pw.println();

    // vertices
    for (Iterator it = graph.vertices().iterator(); it.hasNext(); ) {
        Vertex v = (Vertex) it.next();
        VertexType vType;
        try {
            vType = graphType.getVertexTypeNamed(v.getTypeName());
        } catch (NullPointerException e) {
            // no graphType available
            vType = null;
        }
        pw.print(indent + INDENT_STR);
        pw.print("<" + VERTEX_TAG + " ");
        pw.println(VERTEX_ID_ATTR + "=\"" + ID_PREFIX + ID.get(v) + "\">");
        for(Iterator it2 = v.keySet().iterator(); it2.hasNext(); ) {
            String key = (String) it2.next();
            writeValue(key, v, vType, indent + INDENT_STR + INDENT_STR, pw);
        }
        pw.print(indent + INDENT_STR);
        pw.println("</"+VERTEX_TAG+">");
        pw.println();
    }
```

```
    // edges
    for (Iterator it = graph.edges().iterator(); it.hasNext(); ) {
        Edge e = (Edge) it.next();
        EdgeType eType;
        try {
            eType = graphType.getEdgeTypeNamed(e.getTypeName());
        } catch (NullPointerException ex) {
            // no graphType available
            eType = null;
        }
        pw.print(indent + INDENT_STR);
        pw.print("<" + EDGE_TAG + " ");
        Vertex a = (Vertex) e.getAttribute(GraphConstants.vertexA);
        Vertex b = (Vertex) e.getAttribute(GraphConstants.vertexB);
        pw.print(EDGE_A_ATTR + "=\"" + ID_PREFIX + ID.get(a) + "\" ");
        pw.println(EDGE_B_ATTR + "=\"" + ID_PREFIX + ID.get(b) + "\">");
        for(Iterator it2 = e.keySet().iterator(); it2.hasNext(); ) {
            String key = (String) it2.next();
            writeValue(key, e, eType, indent + INDENT_STR + INDENT_STR, pw);
        }
        pw.print(indent + INDENT_STR);
        pw.println("</"+EDGE_TAG+">");
        pw.println();
    }
    pw.print(indent);
    pw.println("</"+GRAPH_TAG+">");
    pw.println();
}

/**
 * Write an attribute (Moses termininology) with the name key from the
 * AttributeCarrier using the AbstractType information to the PrintWriter.
 * <p>
 * This method looks first in field type of the AttributeSpec in order
 * to determine the output type.
 * Second, look if two attributes with suffixes GraphConstants.sourceSuffix
 * and GraphConstants.parserSuffix are available.
 * Third, write the primitive JAVA types like String, Integer, ... without
 * source and parser Suffix attributes out.
 *
 * @see GraphConstants
 *
 * @param key Key name of the attribute.
 * @param n AttributeCarrier with the attribute with the name of key.
 * @param type AbstractType with the type information about the
 * attribute with the name of key.
 * @param indent Indentation used by this recursion.
 * @param pw PrintWriter for the output data.
 *
 * @throws IOException Write errors.
 */
void writeValue(String key, AttributeCarrier n, AbstractType type,
                String indent, PrintWriter pw)
            throws IOException {
    Object val = n.getAttribute(key);
    AttributeSpec attrSpec;
    try {
        attrSpec = type.getAttributeSpec(key);
    } catch (NullPointerException e) {
        // no type provided
        attrSpec = null;
    }
    if (attrSpec != null) {
        if (attrSpec.type == GraphType.atInteger) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_INTEGER + "\">");
            pw.print(Xml.normalize(val.toString()));
```

```
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atReal) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_REAL + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atString) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_STRING + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atText) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_TEXT + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atColor) {
            /*pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_COLOR + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");*/
        } else if (attrSpec.type == GraphType.atType) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_TYPE + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atExpressionFrame) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_EXPRESSIONFRAME + "\">");
            String source = (String)
                n.getAttribute(key + GraphConstants.sourceSuffix);
            pw.print(Xml.normalize(source));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atExpression) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_EXPRESSION + "\">");
            String source = (String)
                n.getAttribute(key + GraphConstants.sourceSuffix);
            pw.print(Xml.normalize(source));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atParameters) {
            pw.print(indent);
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_PARAMETERS + "\">");
            String source = (String)
                n.getAttribute(key + GraphConstants.sourceSuffix);
            pw.print(Xml.normalize(source));
            pw.println("</" + VAL_TAG + ">");
        } else if (attrSpec.type == GraphType.atGraph) {
            // recursive call
            writeGraph((Graph) val, key, indent, pw);
        } else if (attrSpec.type == GraphType.atParsed) {
            // do nothing
        } else if (attrSpec.type == GraphType.atOther) {
            // do nothing
```

```
    } else if (attrSpec.type == GraphType.atDerived) {
        // do nothing
    } else if (attrSpec.type == GraphType.atUndef) {
        // do nothing
    } else {
        // unknown type
    }
} else {
    String source = (String)
        n.getAttribute(key + GraphConstants.sourceSuffix);
    String parser = (String)
        n.getAttribute(key + GraphConstants.parserSuffix);
    if (source != null && parser != null) {
        pw.print(indent);
        pw.print("<" + VAL_TAG + " ");
        pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
        pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_EXTRAPARSER + "\" ");
        pw.print(VAL_PARSER_ATTR + "=\"" + Xml.normalize(parser) + "\">");
        pw.print(Xml.normalize(source));
        pw.println("</" + VAL_TAG + ">");
    // else special cases, no type information is available
    // process the primitive types
    // these special cases may sometimes vanish
    } else if (!key.endsWith(GraphConstants.sourceSuffix)
                && !key.endsWith(GraphConstants.parserSuffix)
                && !key.equals(GraphConstants.vertexA)
                && !key.equals(GraphConstants.vertexB)) {
        if (val instanceof Integer) {
            pw.print(indent);
            pw.print("<!-- special -->");
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_INTEGER + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");
        } else if (val instanceof String) {
            pw.print(indent);
            pw.print("<!-- special -->");
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_STRING + "\">");
            pw.print(Xml.normalize(val.toString()));
            pw.println("</" + VAL_TAG + ">");
        } else if (val instanceof Boolean) {
            pw.print(indent);
            pw.print("<!-- special -->");
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_BOOLEAN + "\">");
            pw.print(((Boolean) val).booleanValue() ? TRUE : FALSE);
            pw.println("</" + VAL_TAG + ">");
        } else if (val instanceof Double) {
            pw.print(indent);
            pw.print("<!-- special -->");
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_DOUBLE + "\">");
            pw.print(((Double) val).toString());
            pw.println("</" + VAL_TAG + ">");
        } else if (val instanceof Vector) {
            //very special case
            pw.print(indent);
            pw.print("<!-- special -->");
            pw.print("<" + VAL_TAG + " ");
            pw.print(VAL_NAME_ATTR + "=\"" + Xml.normalize(key) + "\" ");
            pw.print(VAL_TYPE_ATTR + "=\"" + TYPE_POINT + "\">");
            String delim = "";
            //comma separated integers; a point is a pair
            for (Iterator it = ((Vector) val).iterator(); it.hasNext(); ) {
                Object o = it.next();
```

```
                    if (o instanceof Point) {
                        pw.print(delim);
                        delim = POINT_DELIM + " ";
                        pw.print((int) ((Point) o).getX());
                        pw.print(delim);
                        pw.print((int) ((Point) o).getY());
                    }
                }
                pw.println("</" + VAL_TAG + ">");
            } else {
                if (!key.endsWith(GraphConstants.sourceSuffix)
                        && !key.endsWith(GraphConstants.parserSuffix)
                        && !key.equals(GraphConstants.vertexA)
                        && !key.equals(GraphConstants.vertexB)
                        && !key.equals(GraphConstants.vertices)
                        && !key.equals(GraphConstants.edges)) {
                    System.err.println("not written: " + key);
                    //System.err.println(" " + val + " " + val.getClass());
                }
            }
        }
    }
}

/**
 * Load a graph form a xml file using an input stream.
 * The xml file is parsed through a DOM tree.
 * The dtd path is used.
 * <p>
 * The graphstorage xml file version is validated during the loading process.
 *
 * @param inputStream Data source for reading.
 *
 * @throws IOException
 * @throws ClassNotFountException
 */
public Object load(InputStream inputStream)
        throws IOException, ClassNotFoundException {
    System.out.println("Load graph");

    Graph graph = null;
    try {
        DOMParserWrapper parser =
            (DOMParserWrapper) Class.forName(DOM_PARSER).newInstance();
        InputSource inputSource = new InputSource(inputStream);
        try {
            inputSource.setSystemId(xmlFile.toExternalForm());
        } catch (NullPointerException e) {
            // no xml specified
        }
        Document doc = parser.parse(inputSource);
        Element rootEl = (Element) doc.getElementsByTagName(ROOT_TAG).item(0);
        Element graphEl = (Element) rootEl.getElementsByTagName(GRAPH_TAG).item(
0);

        if (!version.valid(rootEl.getAttribute(ROOT_VER_ATTR))) {
            throw new IOException("moses: wrong repository version!");
        }

        // load the graph
        graph = getGraph(graphEl);

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (SAXParseException e) {
        System.err.println("SystemId:" + e.getSystemId());
        System.err.println("PublicId: " + e.getPublicId());
        System.err.println("Line: " + e.getLineNumber()
                           + " Col: " + e.getColumnNumber());
        if (e.getException() != null) {
```

```
            System.err.println("inner: " + e.getException());
            System.err.println("message: " + e.getMessage());
        }
        e.printStackTrace();
    } catch (SAXException e) {
        if (e.getException() != null) {
            System.err.println("inner: " + e.getException());
            System.err.println("message: " + e.getMessage());
        }
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (ClassCastException e) {
        e.printStackTrace();
    } catch (PropertyVetoException e) {
        e.printStackTrace();
    }

    return graph;
}

/**
 * Reconstruct a graph using the data from the DOM element el.
 * This method is prepared for recursive calls.
 *
 * @param el GRAPH_TAG element.
 * @return The reconstructed graph.
 *
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws PropertyVetoException
 */

Graph getGraph(Element el)
        throws IOException, ClassNotFoundException, PropertyVetoException {
    // mapping ids and objects
    Map ids = new HashMap();
    Graph graph = new Graph();
    NodeList nl;

    // get Values for the graph
    getValues(el, graph);
    getSubGraphs(el, graph);

    // vertices
    nl = el.getChildNodes();
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        // only vertex tags
        if (n instanceof Element && ((Element) n).getTagName().equals(VERTEX_TAG
)) {
            Element vEl = (Element) n;
            Vertex v = new Vertex();
            String id = vEl.getAttribute(VERTEX_ID_ATTR);
            ids.put(id, v);
            graph.addVertex(v);

            getValues(vEl, v);
            getSubGraphs(vEl, v);
        }
    }

    // edges
    nl = el.getChildNodes();
    for (int i = 0; i < nl.getLength(); i++) {
        Node n = nl.item(i);
        // only edge tags
        if (n instanceof Element && ((Element) n).getTagName().equals(EDGE_TAG))
```

```
 {
            Element eEl = (Element) n;
            String A = eEl.getAttribute(EDGE_A_ATTR);
            String B = eEl.getAttribute(EDGE_B_ATTR);
            Edge e = new Edge((Vertex) ids.get(A), (Vertex) ids.get(B));
            graph.addEdge(e);

            getValues(eEl, e);
            getSubGraphs(eEl, e);
        }
    }

    // make derived attributes
    updateObjects(graph);

    return graph;
}


/**
 * Processes the VAL_TAG elements of the DOM tree. The data is added to the
 * AttributeCarrier.
 * The different data types are constructed with the type information form
 * the xml file.
 *
 * @param el GRAPH_TAG element.
 * @param n Add element to this AttributeCarrier.
 *
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws PropertyVetoException
 */
void getValues(Element el, AttributeCarrier n)
        throws IOException, ClassNotFoundException, PropertyVetoException {
    NodeList nl = el.getChildNodes();
    for(int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        // only value tags
        if (node instanceof Element && ((Element) node).getTagName().equals(VAL_
TAG)) {
            Element v = (Element) node;
            String name = v.getAttribute(VAL_NAME_ATTR);
            String type = v.getAttribute(VAL_TYPE_ATTR);
            String s;
            if (v.hasChildNodes()) {
                s = v.getFirstChild().getNodeValue();
            } else {
                s = "";
            }
            if (type.equals(TYPE_INTEGER)) {
                n.putAttribute(name, new Integer(s));
            } else if (type.equals(TYPE_REAL)) {
                n.putAttribute(name, new Float(s));
            } else if (type.equals(TYPE_STRING)) {
                n.putAttribute(name, s);
            } else if (type.equals(TYPE_TEXT)) {
                n.putAttribute(name, s);
            } else if (type.equals(TYPE_COLOR)) {
                // do nothing
            } else if (type.equals(TYPE_TYPE)) {
                n.putAttribute(name, s);
            } else if (type.equals(TYPE_EXPRESSIONFRAME)) {
                n.putAttribute(name + GraphConstants.sourceSuffix, s);
                ElanExpressionFrameParser parser = new ElanExpressionFrameParser
();
                try {
                    n.putAttribute(name, parser.parse(s));
                } catch (ParseException e) {
                    e.printStackTrace();
                }
```

```
            } else if (type.equals(TYPE_EXPRESSION)) {
                n.putAttribute(name + GraphConstants.sourceSuffix, s);
                ElanExpressionParser parser = new ElanExpressionParser();
                try {
                    n.putAttribute(name, parser.parse(s));
                } catch (ParseException e) {
                    e.printStackTrace();
                }
            } else if (type.equals(TYPE_PARAMETERS)) {
                n.putAttribute(name + GraphConstants.sourceSuffix, s);
                ElanParameterDeclParser parser = new ElanParameterDeclParser();
                try {
                    n.putAttribute(name, parser.parse(s));
                } catch (ParseException e) {
                    e.printStackTrace();
                }
            } else if (type.equals(TYPE_EXTRAPARSER)) {
                String parserName = v.getAttribute(VAL_PARSER_ATTR);
                n.putAttribute(name + GraphConstants.sourceSuffix, s);
                n.putAttribute(name + GraphConstants.parserSuffix, parserName);
                try {
                    graph.util.Parser parser = (graph.util.Parser)
                        Class.forName(parserName).newInstance();
                    n.putAttribute(name, parser.parse(s));
                } catch (ParseException e) {
                    e.printStackTrace();
                } catch (InstantiationException e) {
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    e.printStackTrace();
                } catch (ClassCastException e) {
                    e.printStackTrace();
                }
            } else if (type.equals(TYPE_BOOLEAN)) {
                n.putAttribute(name, new Boolean(s.equals(TRUE)));
            } else if (type.equals(TYPE_DOUBLE)) {
                n.putAttribute(name, new Double(s.trim()));
            } else if (type.equals(TYPE_POINT)) {
                Vector vect = new Vector();
                try {
                    for (StringTokenizer tok = new StringTokenizer(s, POINT_DELI
M);
                                tok.hasMoreTokens(); ) {
                        int x = Integer.parseInt(tok.nextToken().trim());
                        int y = Integer.parseInt(tok.nextToken().trim());
                        vect.add(new Point(x, y));
                    }
                } catch (NoSuchElementException e) {
                    System.err.println("wrong point format");
                    e.printStackTrace();
                } catch (NumberFormatException e) {
                    System.err.println("wrong point format");
                    e.printStackTrace();
                } finally {
                    n.putAttribute(name, vect);
                }
            } else {
                System.err.println("unknown value: " + name + " " + s);
            }
        } else {
            // wrong nodes
        }
    }
}

/**
 * Looks for nested GRAPH_TAG elements in the DOM tree. For each sub graph
 * make a recursive call. Add the sub graphs to AttributeCarrier.
 *
 * @param el GRAPH_TAG element.
```

```
 * @param n Add sub graph to this AttributeCarrier.
 *
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws PropertyVetoException
 */
void getSubGraphs(Element el, AttributeCarrier n)
        throws IOException, ClassNotFoundException, PropertyVetoException {
    NodeList nl = el.getChildNodes();
    for(int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        if (node instanceof Element && ((Element) node).getTagName().equals(GRAP
H_TAG)) {
            Element gEl = (Element) node;
            String name = gEl.getAttribute(GRAPH_NAME_ATTR);
            Graph graph = getGraph(gEl);
            n.putAttribute(name, graph);
        }
    }
}

/**
 * Update all attributes of the graph and make e.g. derived attributes.
 *
 * @param graph Update this graph.
 *
 * @throws IOException
 * @throws ClassNotFoundException
 * @throws PropertyVetoException
 */
void updateObjects(Graph graph)
    throws IOException, ClassNotFoundException, PropertyVetoException {
    GraphType graphType = null;
    try {
        String graphTypeName = (String) graph.getType();
        graphType = graphTypeFinder.getGraphType(graphTypeName);

        // a graph can't be updated

        // vertices
        for (Iterator it = graph.vertices().iterator(); it.hasNext(); ) {
            Vertex v = (Vertex) it.next();
            graphType.updateObject(graph, v);
        }

        // edges
        for (Iterator it = graph.edges().iterator(); it.hasNext(); ) {
            Edge e = (Edge) it.next();
            graphType.updateObject(graph, e);
        }
    } catch (NullPointerException e) {
        // the graph don't have a type
        return;
    }
}

/**
 * Get type information as specified in GraphType.
 * For debugging.
 *
 * @param atType Type constant.
 * @return Type name as string.
 */
String getAttrType(int atType) {
    switch (atType) {
        case GraphType.atColor : return "Color";
        case GraphType.atDerived : return "Derived";
        case GraphType.atExpression : return "Expression";
        case GraphType.atExpressionFrame : return "ExpressionFrame";
        case GraphType.atGraph : return "Graph";
```

```
        case GraphType.atInteger : return "Integer";
        case GraphType.atOther : return "Other";
        case GraphType.atParameters : return "Parameters";
        case GraphType.atParsed : return "Parsed";
        case GraphType.atReal : return "Real";
        case GraphType.atString : return "String";
        case GraphType.atText : return "Text";
        case GraphType.atType : return "Type";
        case GraphType.atUndef : return "Undef";
        default: return "unknown type";
    }
}

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.repository.objecttypes.mosescomponent;

import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import javax.swing.*;

import graph.*;
import graph.util.*;
import graph.graphtype.*;
import moses.repository.objecttypes.*;
import moses.util.*;
import moses.text.*;
import moses.xml.*;
import moses.editor.*;
import moses.repository.structure.*;
import moses.models.translator.*;
import moses.launcher.LauncherFrame;



/**
 * User action to store component graphs.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 *
 * @see UserAction
 * @see Action
 * @see AdapterObjectInterface
 * @see GraphTypeFinder
 */
public  class ExportGraphXMLAction
        extends GraphUserAction
{
  /**
   * Create ExportGraphXMLAction.
   *
   * @param desc Descriptor of the object which is to be exported
   * @param applications Dictionary of running repository applications
   * @param launcher LauncherFrame of the top-level application
```

```
   */
public ExportGraphXMLAction(Descriptor desc, Dictionary applications,
        LauncherFrame launcher) {
    super("Export GraphXML", desc, applications, launcher);
}

/**
 * Handler which asks a file name and then stores the component graph.
 *
 * @param event ActionEvent
 */
public void actionPerformed(ActionEvent event){
    try {
        Graph graph = getGraph();

        //makeTestData(graph);

        // get a file
        File file = null;
        final JFileChooser fc = new JFileChooser();
        fc.addChoosableFileFilter(new XMLFileFilter());
        int returnVal = fc.showSaveDialog(launcher);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            file = fc.getSelectedFile();
            if (file.exists()) {
                int returnVal2 = JOptionPane.showConfirmDialog(null,
                    "File already exists!\nOverwrite?",
                    "Warning", JOptionPane.YES_NO_OPTION,
                    JOptionPane.WARNING_MESSAGE);
                if (returnVal2 != JOptionPane.YES_OPTION) {
                    // do nothing
                    return;
                }
            }
        } else {
            // cancel
            return;
        }

        // store graph
        ObjectXMLStorageStrategy storage = new GraphXMLStorage(gtf);
        OutputStream outputStream = new FileOutputStream(file);
        storage.store(outputStream, graph);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Export of file failed.\n" + e,
            "Error", JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}


/**
 * Overrides the parent class method to decide whether
 * the Action is enabled. This is the case if
 * the descriptor is not locked and
 * HAS_CONTENT is true.
 *
 * @return true = allow action
 */
public boolean checkEnabled() {
    try {
        LockManager lm = (LockManager) applications.get("LockManager");
        String content = (String) desc.get(MosesComponentAdapter.HAS_CONTENT);
        return checkDisableOptions() && !lm.isLocked(desc)
                && content != null && content.equals("true");
    } catch(Exception e) {
        System.err.println("Couldn't check whether ExportAction may be enabled.\n" );
        e.printStackTrace();
        return false;
    }
```

```
}

/**
 * Get the graph declared in the descriptor.
 *
 * @return Graph form descriptor.
 */
private Graph getGraph() {
    try {
        ObjectPersistenceManager pm = (ObjectPersistenceManager)
            desc.get(ObjectAdapter.PERSISTENCE_STRATEGY);
        return (Graph) pm.load(desc);
    } catch(Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * Insert test data in the graph in order to test attributes
 * with neighbour attributes with sourceSuffix and parserSuffix.
 *
 * @param graph Graph to manipulate.
 */
private void makeTestData(Graph graph) {
    // Testdaten fuer Graphen mit _Parser und _Source
    // fuegt neue Attribute beim 1. Knoten im Graphen ein
    Vertex v = (Vertex) graph.vertices().iterator().next();
    // ganz leer
    v.putAttribute("Test1", "Data1");
    // nur Source
    v.putAttribute("Test2", "Data2");
    v.putAttribute("Test2" + GraphConstants.sourceSuffix, "Source2");
    // Source und Parser
    // parser muss vorhanden sein
    //v.putAttribute("Test3", "Data3");
    //v.putAttribute("Test3" + GraphConstants.sourceSuffix, "Source3");
    //v.putAttribute("Test3" + GraphConstants.parserSuffix, "Parser3");

    // richtiger parser
    v.putAttribute("Test4", "schrott");
    v.putAttribute("Test4" + GraphConstants.sourceSuffix, "pos = s(0),\nmsg = [id, s(0)]");
    v.putAttribute("Test4" + GraphConstants.parserSuffix, "graph.util.ElanExpressionFrame
Parser");
}

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.repository.objecttypes.mosescomponent;

import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
import java.beans.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import graph.*;
import moses.xml.*;
import moses.repository.structure.*;
import moses.repository.objecttypes.*;
import moses.repository.objecttypes.mosescomponent.*;
import moses.launcher.LauncherFrame;


/**
 * User action to import a Moses component graph
 * from a xml file.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 *
 * @see ObjectPersistenceManager
 * @see UserAction
 * @see Action
 * @see GraphTypeFinder
 */
public  class ImportGraphXMLAction
    extends GraphUserAction
{

/**
 * Create a ImportGraphXMLAction.
 *
 * @param desc Descriptor of the object which is  to be imported.
 * @param applications Dictionary of running repository applications.
 * @param launcher LauncherFrame of the top-level application.
 */
public ImportGraphXMLAction(Descriptor desc, Dictionary applications,
        LauncherFrame launcher) {
    super("Import GraphXML", desc, applications, launcher);
```

```
}

/**
 * Handler which asks a file name and then loads the component graph.
 * The descriptor is locked during the loading.
 * The descriptor will be updated with the new informations.
 *
 * @param event ActionEvent
 */
public void actionPerformed(ActionEvent event) {
    LockManager lm = null;
    try {
        lm = (LockManager) applications.get("LockManager");
        // lock descriptor
        lm.lock(desc);

        // get a file
        File file = null;
        final JFileChooser fc = new JFileChooser();
        fc.addChoosableFileFilter(new XMLFileFilter());
        int returnVal = fc.showOpenDialog(launcher);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            file = fc.getSelectedFile();
            if (!file.exists()) {
                throw new IOException("File don't exist!");
            }
        } else {
            // cancel
            return;
        }

        // load graph
        URL xmlFile = file.toURL();
        ObjectXMLStorageStrategy storage = new GraphXMLStorage(gtf, xmlFile);
        InputStream inputStream = new FileInputStream(file);
        Graph graph = (Graph) storage.load(inputStream);

        // store graph within repository
        ObjectPersistenceManager opm = (ObjectPersistenceManager)
            desc.get(ObjectAdapter.PERSISTENCE_STRATEGY);
        opm.store(desc, graph);
        // update descriptor TYPE
        String type = (String) graph.getAttribute(GraphConstants.typeName);
        desc.set(MosesComponentAdapter.TYPE, type);
        // update descriptor HAS_CONTENT
        desc.set(MosesComponentAdapter.HAS_CONTENT, "true");
        //update descriptor MODIFICATION_TIME
        DateFormat dateFormatter =
            DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
            DateFormat.MEDIUM, Locale.getDefault());
        desc.set(ObjectAdapter.MODIFICATION_TIME,
            dateFormatter.format(new Date()));

        // store changes to the repository
        Repository rep = (Repository) applications.get("Repository");
        rep.store();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Import of file failed.\n" + e,
            "Error", JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    } finally {
        // unlock descriptor
        try {
            lm.unlock(desc);
        } catch(Exception e) {
            JOptionPane.showMessageDialog(null, "Unlock of descriptor failed.\n" + e,
                "Error", JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
    }
```

```
}

/**
 * Overrides the parent class method to decide whether
 * the Action is enabled. This is the case if
 * the descriptor is not locked.
 *
 * @return true = allow action
 */
public boolean checkEnabled() {
    try {
        LockManager lm = (LockManager) applications.get("LockManager");
        return checkDisableOptions() && !lm.isLocked(desc);
    } catch(Exception e) {
        System.err.println("Couldn't check whether ImportAction may be enabled.\n" + e);
        e.printStackTrace();
        return false;
    }
}

}
```

```
package graph;

/**
 * Defines some of those nasty constants used everywhere by graphs
 * @author Rob Esser
 *
 * @version 14-01-00 (rek) added parserSuffix
 * @version 22-01-99 (jwj) moved to graph package
 * @version 24-11-98
 */

public interface GraphConstants extends Version {

  static final long serialVersionUID = versionGraph;

  final static String typeName = "TypeName";
  final static String instantiatedShape = "InstantiatedShape"; //MN 12.1.99

  final static String name = "Name";
  final static String parameters = "Parameters";
  final static String packageName = "PackageName";
  final static String signature = "Signature";
  final static String importedPackages = "importedPackages";
  final static String superClass = "SuperClass";
  final static String declarations = "Declarations";
  final static String componentClass = "ComponentClass";

  final static String modelInterface = "ModelInterface";
  final static String compiler = "Compiler";
  final static String interpreter = "Interpreter";
  final static String animationAdapter = "AnimationAdapter";
  //MN 6.1.99

  final static String graphTypeDeclarations = "GraphTypeDeclarations";

  final static String info = "Info";
  final static String documentation = "Documentation";
  final static String documentationType = "DocumentationType";


  final static String editorPlugInName = "EditorPlugInName";
  final static String editorPropertiesName = "EditorProperties";

  final static String connectorA = "ConnectorA";
  final static String connectorB = "ConnectorB";

  final static String sourceSuffix = "_Source";
  final static String parserSuffix = "_Parser";   //REK 14.1.2000

  //
  //  signature defs
  //

  static final String       sigInputs = "Inputs";
  static final String       sigOutputs = "Outputs";
  static final String       sigInterfaces = "Interfaces";
  static final String       sigInvInterfaces = "InvInterfaces";

  //
  // basic structural elements
  //

  static final String vertexA = "#VertexA";
  static final String vertexB = "#VertexB";
  static final String vertices = "#Vertices";
  static final String edges = "#Edges";

  //
  // graph type constants
  //
```

```
  static final String vertexTypes = "#VertexTypes";
  static final String edgeTypes = "#EdgeTypes";
  static final String predicates = "#Predicates";

}
```

D.3. Hilfsklassen

Auf den nächsten Seiten ist der Programmcode abgedruckt.

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

/**
 * Service class for xml.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class Xml {

    /**
     * Normalizes the given string for XML output.
     * Replaces special characters, used by XML itself.
     *
     * @param s Normal Java string.
     * @return String ready for XML Output.
     */
    public static String normalize(String s) {
        StringBuffer str = new StringBuffer();
        int len = (s != null) ? s.length() : 0;
        for (int i = 0; i < len; i++) {
            char ch = s.charAt(i);
            switch (ch) {
                case '<': {str.append("&lt;"); break;}
                case '>': {str.append("&gt;"); break;}
                case '&': {str.append("&amp;"); break;}
                case '\u0022': {str.append("&quot;"); break;}        // "
                case '\'': {str.append("&apos;"); break;}
                default: {str.append(ch);}
            }
        }
        return str.toString();
    }

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.net.URL;
import java.io.IOException;
// xml
import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

/**
 * Encapsulates a validating DOM parser using the stratgey pattern.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 *
 * @see moses.repository.structure.RepositoryXMLDom
 * @see moses.repository.objecttypes.mosescomponent.GraphXMLStorage
 */
public interface DOMParserWrapper {

    /**
     * Parses the specified InputSource and returns a DOM document.
     *
     * @param inputSource InputSource to be parsed.
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(InputSource inputSource)
            throws IOException, SAXException;

    /**
     * Parses the specified URL and returns a DOM document.
     *
     * @param url XML filename as URL
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(URL url)
            throws IOException, SAXException;

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.io.*;
import java.net.URL;
// Sun DOM Parser
import com.sun.xml.tree.*;
// w3c Interaces
import org.w3c.dom.*;
import org.xml.sax.*;

/**
 * This class uses the validating Sun Project X Technology Release 2 parser.
 * @author Roland E. Kurmann, ETHZ
 * @version 14.1.2000 REK created
 */
public class DOMParser_Sun_XTR2 implements DOMParserWrapper {

    /**
     * Parses the specified InputSource and returns a DOM document.
     *
     * @param inputSource InputSource to be parsed.
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(InputSource inputSource)
            throws IOException, SAXException {
        //return XmlDocument.createXmlDocument(url.toExternalForm(), true);
        //System.out.println("stream test: " + url.getFile());
        //return XmlDocument.createXmlDocument(url.openStream(), true);
        return XmlDocument.createXmlDocument(inputSource, true);
    }

    /**
     * Parses the specified URL and returns a DOM document.
     *
     * @param url XML filename as URL
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(URL url)
            throws IOException, SAXException {
        return XmlDocument.createXmlDocument(url.toExternalForm(), true);
        //System.out.println("stream test: " + url.getFile());
```

```
        //return XmlDocument.createXmlDocument(url.openStream(), true);
    }
}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.io.*;
import java.net.URL;
//ibm xml parser
import com.ibm.xml.parsers.*;
//w3c
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.InputSource;

/**
 * This class uses the validating IBM XML4J Release 2 parser.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class DOMParser_IBM_XML4J2 implements DOMParserWrapper {

    /**
     * Parses the specified InputSource and returns a DOM document.
     *
     * @param inputSource InputSource to be parsed.
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(InputSource inputSource)
            throws IOException, SAXException {
        RevalidatingDOMParser parser = new RevalidatingDOMParser();
        //InputSource isource = new InputSource(url.toExternalForm());
        parser.parse(inputSource);
        Document doc = parser.getDocument();
        Element rep = doc.getDocumentElement();
        boolean valid = parser.validate(rep) == null;
        if (!valid) {
            throw new SAXException("XML file not valid");
        }
        return doc;
    }

    /**
     * Parses the specified URL and returns a DOM document.
     *
```

```
     * @param url XML filename as URL
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(URL url)
            throws IOException, SAXException {
        InputSource inputSource = new InputSource(url.toExternalForm());
        return parse(inputSource);
    }


}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.io.*;
import java.net.URL;
// Oracle DOM Parser
import oracle.xml.parser.v2.*;
// w3c Interaces
import org.w3c.dom.*;
import org.xml.sax.*;

/**
 * This class uses the validating Oracle v2 parser.
 * @author Roland E. Kurmann, ETHZ
 * @version 14.1.1999 REK, created
 */

public class DOMParser_Oracle_v2 implements DOMParserWrapper {

    /**
     * Parses the specified InputSource and returns a DOM document.
     *
     * @param inputSource InputSource to be parsed.
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(InputSource inputSource)
            throws IOException, SAXException {
        DOMParser parser = new DOMParser();
        parser.setErrorStream(System.err);
        parser.setValidationMode(true);
        parser.showWarnings(true);
        //parser.parse(url);
        parser.parse(inputSource);
        return parser.getDocument();
    }

    /**
     * Parses the specified URL and returns a DOM document.
     *
     * @param url XML filename as URL
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
```

```
    public Document parse(URL url)
            throws IOException, SAXException {
        DOMParser parser = new DOMParser();
        parser.setErrorStream(System.err);
        parser.setValidationMode(true);
        parser.showWarnings(true);
        parser.parse(url);
        return parser.getDocument();
    }

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.io.*;
import java.net.*;
import java.util.*;
// java xml api for parsers
import javax.xml.parsers.*;
// w3c Interaces
import org.w3c.dom.*;
import org.xml.sax.*;

/**
 * This class uses the DocumentBuilderFactory
 * to create a DOM Parser. The DocumentBuilderFactory gets
 * the parser from the property "javax.xml.parsers.DocumentBuilderFactory"
 * using System.getProperty().
 *
 * @author Roland E. Kurmann, ETHZ
 * @version 14.1.2000 REK created
 */
public class DOMParser_javax implements DOMParserWrapper {

    /**
     * Parses the specified InputSource and returns a DOM document.
     *
     * @param inputSource InputSource to be parsed.
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(InputSource inputSource)
            throws IOException, SAXException {
        try {
            System.out.println("javax");
            // get the right parser through the factory
            DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();
            factory.setNamespaceAware(false);
            factory.setValidating(true);
            factory.setLocale(Locale.getDefault());
            DocumentBuilder parser = factory.newDocumentBuilder();

            // parse
            parser.setEntityResolver(null);     //use default implementation
            parser.setErrorHandler(null);       //use default implementation
```

```
            //return parser.parse(url.toExternalForm());
            return parser.parse(inputSource);
        } catch (FactoryException e) {
            throw new SAXException(e.toString());
        }
    }

    /**
     * Parses the specified URL and returns a DOM document.
     *
     * @param url XML filename as URL
     * @throws SAXException Any XML errors.
     * @throws IOException Reading errors.
     * @return DOM Document
     */
    public Document parse(URL url)
            throws IOException, SAXException {
        try {
            System.out.println("javax");
            // get the right parser through the factory
            DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();
            factory.setNamespaceAware(false);
            factory.setValidating(true);
            factory.setLocale(Locale.getDefault());
            DocumentBuilder parser = factory.newDocumentBuilder();

            // parse
            parser.setEntityResolver(null);     //use default implementation
            parser.setErrorHandler(null);       //use default implementation
            return parser.parse(url.toExternalForm());
        } catch (FactoryException e) {
            throw new SAXException(e.toString());
        }
    }

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

/**
 * This interface encapsulates the creation of an external identifier for object
s.
 * For the same object the implementation must deliver the same identifier.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public interface IDStrategy {

    /**
     * This method returns an unique identifier for an object.
     *
     * @param o The object asking for an identifier.
     * @return Unique identifier.
     */
    String get(Object o);

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.util.*;

/**
 * This class creates unique identifiers through counting.
 * If an object appears for the first, it gets a number.
 * This number will be stored.
 *
 * This class isn't synchronized!
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class CountID implements IDStrategy {

    /** Holdes the objects and the identifiers. */
    private Map map = new HashMap();

    /** Holdes the next free number. */
    private long count;

    /**
     * This is the standart constructor.
     * The counting starts with 0.
     */
    public CountID() {
        this(0);
    }

    /**
     * You can initialize the starting number.
     *
     * @param start The first number for an object.
     */
    public CountID(long start) {
        count = start;
    }

    /**
     * Returns the identifier for the object.
     *
     * @param o The object asking for an identifier.
     * @exception IndexOutOfBoundsException No more numbers.
     * @return Unique identifier.
```

```
     */
    public String get(Object o) {
        if (!map.containsKey(o)) {
            map.put(o, new Long(count));
            count++;
            if (count == Long.MAX_VALUE) {
                throw new IndexOutOfBoundsException("MAX_VALUE");
            }
        }
        return map.get(o).toString();
    }

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.util.*;

/**
 * This class creates identifiers through the hashCode() method.
 * This way don't work on all platforms and is therefore platform dependent!
 * Use instead the class CountID.
 *
 * @see CountID
 * @see java.lang.Object#hashCode()
 */
public class HashID
        implements IDStrategy
{

    /**
     * Returns an identifier for the object.
     *
     * @param o The object asking for an identifier.
     * @return An identifier constructed with hashCode.
     */
    public String get(Object o) {
        return String.valueOf(o.hashCode());
    }

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

/**
 * Version strategy interface.
 * Versions for this strategy are numbers with one dot.
 * The different strategies pass only bigger versions, only the exact version, .
..
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public interface VersionStrategy
{
    /** Constant for the deliminator of major and minor version numbers. */
    String POINT = ".";

    /**
     * Check the version.
     *
     * @param version Version to test.
     * @return true = version is ok
     */
    boolean valid(String version);
}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

/**
 * Only the same version number as the base version can pass the test.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class ExactVersion
        implements VersionStrategy
{
    /** Test the versions against this version. */
    private String baseVersion = null;

    /**
     * Create the class with the base version.
     *
     * @param baseVersion Test the versions against this version.
     */
    public ExactVersion(String baseVersion) {
        this.baseVersion = baseVersion.trim();
    }

    /**
     * Check the version.
     *
     * @param version Version to test.
     * @return true = version is ok
     */
    public boolean valid(String version) {
        return baseVersion.equals(version.trim());
    }

}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.util.*;

/**
 * Only equal or greater versions as the base version can pass the test.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class EqualOrGreaterThanBaseVersion
        implements VersionStrategy
{

private int majorBaseVersion;
private int minorBaseVersion;

/**
 * Create the class with the base version.
 *
 * @param baseVersion Test the versions against this version.
 */
public EqualOrGreaterThanBaseVersion(String baseVersion) {
    StringTokenizer tok = new StringTokenizer(baseVersion, POINT);
    majorBaseVersion = Integer.parseInt(tok.nextToken());
    minorBaseVersion = tok.hasMoreTokens() ? Integer.parseInt(tok.nextToken()) :
 0;
}

/**
 * Check the version.
 *
 * @param version Version to test.
 * @return true = version is ok
 */
public boolean valid(String version) {
    StringTokenizer tok = new StringTokenizer(version, POINT);
    int majorVersion = Integer.parseInt(tok.nextToken());
    int minorVersion = tok.hasMoreTokens() ? Integer.parseInt(tok.nextToken()) :
 0;
    return (majorVersion > majorBaseVersion) ||
        ((majorVersion == majorBaseVersion) && (minorVersion >= minorBaseVersion
));
}
```

D. Code

```
}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.util.*;

/**
 * Only equal or less versions as the base version can pass the test.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class EqualOrLessThanBaseVersion
        implements VersionStrategy
{

private int majorBaseVersion;
private int minorBaseVersion;

/**
 * Create the class with the base version.
 *
 * @param baseVersion Test the versions against this version.
 */
public EqualOrLessThanBaseVersion(String baseVersion) {
    StringTokenizer tok = new StringTokenizer(baseVersion, POINT);
    majorBaseVersion = Integer.parseInt(tok.nextToken());
    minorBaseVersion = tok.hasMoreTokens() ? Integer.parseInt(tok.nextToken()) :
 0;
}

/**
 * Check the version.
 *
 * @param version Version to test.
 * @return true = version is ok
 */
public boolean valid(String version) {
    StringTokenizer tok = new StringTokenizer(version, POINT);
    int majorVersion = Integer.parseInt(tok.nextToken());
    int minorVersion = tok.hasMoreTokens() ? Integer.parseInt(tok.nextToken()) :
 0;
    return (majorVersion < majorBaseVersion) ||
        ((majorVersion == majorBaseVersion) && (minorVersion <= minorBaseVersion
));
}
```

D. Code

```
}
```

```
/*
 Copyright (c) 2000 Computer Engineering and Communication Networks Lab (TIK)
 Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE TIK OR THE ETH ZURICH BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE TIK OR THE ETH ZURICH HAVE BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE TIK AND THE ETH ZURICH SPECIFICALLY DISCLAIM ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND TIK AND THE ETH ZURICH
 HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.
*/

package moses.xml;

import java.io.File;
import javax.swing.filechooser.FileFilter;

/**
 * XML FileFilter for swing FileChooser.
 *
 * @author Roland E. Kurmann, ETHZ
 * @version  14.1.2000 REK created
 */
public class XMLFileFilter extends FileFilter {

    /** XML file ending. */
    public static final String XML = "xml";

    /** Accept directories and xml files. */
    public boolean accept(File f) {
        if (f != null) {
            return f.isDirectory() || getExtension(f).equals(XML);
        } else {
            return false;
        }
    }

    /** Description for this file filter. */
    public String getDescription() {
        return "XML files (*.xml)";
    }


    /**
     * Return the extension portion of the file's name .
     *
     * @see FileFilter#accept
     */
    public String getExtension(File f) {
        if (f != null) {
            String filename = f.getName();
            int i = filename.lastIndexOf('.');
            if (i > 0 && i < filename.length() - 1) {
                return filename.substring(i + 1).toLowerCase();
            };
        }
        return null;
```

```
    }
}
```