KSM Datenmodell

Contents

1 Einleitung

Mit der Entwicklung einer KSM Anwendung auf Eclipse-RCP Basis war es nötig ein Datenmodell zu entwerfen welches zum MVC-ähnlichen Muster des Eclipse Graphical Editor Framework (GEF) kompatibel ist.

Grundsätzlich lassen sich alle Datenmodelle mit GEF abbilden, jedoch existierte in KSM/Swing nur ein stark mit der GUI und Logik verflochtenes und fehlerhaftes Datenmodell, dessen Serialisierung mit einer XML-Bibliothek erfolgte die nicht mehr weiter verwendet werden soll.

Da es nicht beabsichtigt ist die KSM/Swing Anwendung auf kurze Zeit abzulösen liegt es nahe, dass das Datenmodell universell einsetzbar sein sollte. Mit der Umstellung von KSM/Swing auf ein neues Datenmodell ist einerseits die Daten-Kompatibiltät zwischen KSM/Swing und KSM/RCP gegeben und das Ziel die XML-Serialisierung in KSM/Swing zu überarbeiten kann einfacher erreicht werden.

Dieses Dokument beschreibt die Implementation einer API zum Zugriff auf das in XML-Schema beschriebene Datenformat.

Dieses Datenformat erlaubt es darüberhinaus freie Felder zu definieren welche ebenfalls in diesem Dokument spezifiert werden.

2 Das XML-Schema

Die Namespace URL für das XML-Schema lautet:

`http://www.ba-horb.de/~ksm/xml/ksm-1`

die Schemadatei ist abrufbar unter:

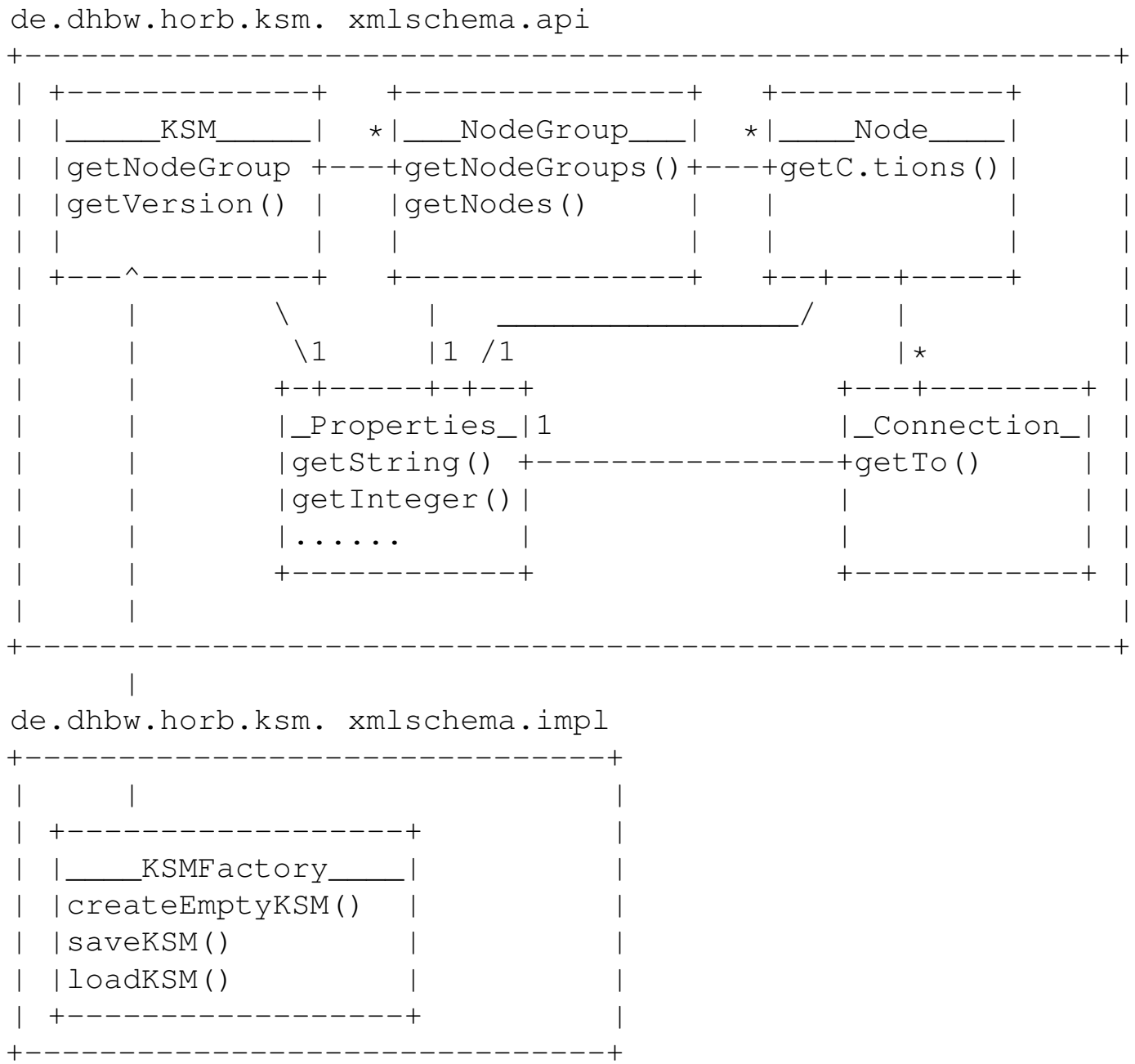
`http://www.ba-horb.de/~ksm/xml/ksm-1.xsd`

Das XML-Schema ist angereichert um JAXB-Annotationen die den XML-Schema-Java-Compiler bei der Klassenerzeugung steuert.

3 Typen des Datenmodell

Die Bibltiohekt besteht aus einem Package in dem Interfaces die API-Beschreiben und einem Package mit Implementationen dieser Interfaces die jedoch für den Benutzer unsichtbar und daher austauschbar sind.

Das folgende Diagram zeigt das Interface das dem Benutzer zur Verfügung steht:



Ein Diagramm wird erstellt indem ein neues (*createEmptyKSM()*) erstellt wird oder eines geladen wird (*loadKSM()*). Die weitere Navigation erfolgt durch Absteigen in dem Objekt-Baum der durch NodeGroup- und schliesslich Node-Objekte gebildet wird.

3.1 Datentypen der Eigenschaften

Im Modell gibt es die Datentypen KSM, NodeGroup, Node und Connection (siehe Klassen-Diagramm). Zusätzlich gibt es die Klasse Properties mit der an die Datentypen zusätzliche Eigenschaften mit einem Text-Bezeichner angehängt werden können.

Für die Eigenschaften stehen folgende Typen zur Verfügung: *'string'*, *'integer'*, *'boolean'*, *'integerList'*, *'decimalList'*, *'stringList'*. Der Datentyp erschliesst sich aus dem Namen.

Prinzipiell kann jeder Bezeichner für jeden Datentyp einmal verwendet werden, ein Bezeichner sollte jedoch wegen der Übersichtlichkeit nur einmal verwendet werden.

4 Verwendung der ChangeListener

Die Klassen *NodeGroup*, *Node* und *Properties* implementieren Change-Listener auf die sich Listener-Klassen registrieren können um über Änderungen am Datenmodell informiert zu werden.

Dies wird beispielsweise benötigt, wenn zwei Programmteile wie ein Eigenschaften-Editor in Tabellenform und ein grafischer Editor voneinander unabhängig auf das Datenmodell zugreifen und beispielsweise die Farbe eines Knoten ändern.

Die Spalte *Index* zeigt an ob das Event eine Index-Eigenschaft hat die andeutet welches Element in einer Liste geändert wurde.

Die Spalte *Version* zeigt an, ab welcher Version des Datenformat (=Version dieses Dokumentes, Version Attribut in <ksm> Element) dieser Event unterstützt wird.

Table 1: *NodeGroup*-Events

Property-Name	**Version**	**Index?**	**Beschreibung**
nodes	1+	✓	Eine *Node* wurde dieser *NodeGroup* hinzugefügt oder entfernt

Table 2: *Node*-Events

Property-Name	**Version**	**Index?**	**Beschreibung**
connections	1+	✓	Eine *Connection* wurde erstellt oder gelöscht

Table 3: *Properties*-Events

Property-Name	**Version**	**Index?**	**Beschreibung**
string: *X*	1+	✗	Eine Zeichenketten Eigenschaft mit Name *X* wurde geändert
decimal: *X*	1+	✗	Eine Fliesskommazahl Eigenschaft mit Name *X* wurde geändert
integer: *X*	1+	✗	Eine ganzzahlige Eigenschaft mit Name *X* wurde geändert
boolean: *X*	1+	✗	Eine boolsche Eigenschaft mit Name *X* wurde geändert
integerList: *X*	1+	✓	Eine Ganzzahl Liste mit Name *X* wurde manipuliert
decimalList: *X*	1+	✓	Eine Fliesskommazahl Liste mit Name *X* wurde manipuliert
stringList: *X*	1+	✓	Eine Zeichenketten Liste mit Name *X* wurde manipuliert

5 Eigenschaften (Properties)

Allen Elementen im Datenmodell lassen sich dynamische, dass heist nicht in einem Schema festgelegte, Eigenschaften zuwiesen. Dies hat zur Folge, dass eine Anwendung sowohl den Fall handhaben muss, dass eine erwartete Eigenschaft nicht vorhanden ist als auch, dass Eigenschaften vorhanden sind die unbekannt sind und ignoriert werden müssen. Dazu steht der Anwendung jedoch das Attribut *Version* im Schema (siehe KSM#getVersion()) zur Verfügung, welches auf eine Version von diesem Dokument zeigt.

Dieser Ansatz wurde gewählt, da sich gezeigt hat, dass die Studienarbeiten im KSM-Projekt einen begrenzten Fokus haben und es daher für einen einzelnen Studenten schwer möglich ist, alle benötigten Datenfelder zu definieren. Das bisherige KSM-Datenformat handhabt dies, indem das XML-Schema beliebig verändert wurde und damit sinnlos wurde. Da dies unvermeidbar ist wird es mit diesem Ansatz aktiv unterstützt.

Eine alternative Herangehensweise wäre die Verwendung von verschiedenen XML-Schemas gewesen wobei mit jeder Erweiterung ein zusätzlicher Namensraum eingeführt wird. Dies schien jedoch sehr viel umständlicher und unnötig kompliziert.

5.1 Eigenschaften von Modellen (KSM)

Die Spalte *Typ* zeigt den Datentyp der Eigenschaft an. Die Spalte *Schlüssel* den Bezeichner welcher in Kombination mit dem Typ eindeutig ist.

Die Spalte *Version* zeigt an, ab welcher Version des Datenformat (=Version dieses Dokumentes, Version Attribut in <ksm> Element) dieser Event unterstützt wird.

Table 4: KSM Properties

Typ	**Schlüssel**	**Version**	**Beschreibung**
nichts	ist	definiert	-

5.2 Eigenschaften von Knoten (Node)

Table 5: Node Properties

Typ	**Schlüssel**	**Ver.**	**Beschreibung**
string	visual.caption	1+	Titel der Node im Editor.
string	visual.color	1+	Farbliches Merkmal der Node Hexadezimal im 8-Bit RGB Format wie folgt: `#RRGGBB`.
integer	visual.location.x	1+	X-Position relativ zur übergeordneten NodeGroup
integer	visual.location.y	1+	Y-Position relativ zur übergeordneten NodeGroup
decimal	data.user_value	1+	User Value (?)

Table 5: (continued)

Typ	Schlüssel	Ver.	Beschreibung
decimal	data.min_value	1+	Minimal Value (?)
decimal	data.max_value	1+	Maximal Value (?)
decimal	data.extern	1+	Extern Value (?)

Als Erbe aus dem KSM/Swing Projekt kann *'visual.color'* die folgenden Werte annehmen, diese sollen auf den folgenden RGB-Wert übertragen werden:

- `White` → `#ffffff`
- `Light Yellow` → `#faffa2`
- `Medium Yellow` → `#f4ff4b`
- `Yellow` → `#edfc00`
- `Light Blue` → `#d4d5e9`
- `Medium Blue` → `#7678ff`
- `Blue` → `#0002f8`
- `Light Green` → `#c8f8c9`
- `Medium Green` → `#7afa7e`
- `Green` → `#1af520`
- `Light Red` → `#fdcccc`
- `Medium Red` → `#f95959`
- `Red` → `#f62020`

5.3 Eigenschaften von Verbindungen (Connection)

Table 6: Connection Properties

Typ	Schlüssel	Ver.	Beschreibung
string	visual.caption	1+	Titel der Connection im Editor.
string	visual.color	1+	Farbliches Merkmal der Connection
string	data.functionType	1+	Ein Funktionstyp von: *straight-line*, *individual*, *parable*, *parabolic-sections*.
decimalList	data.function	1+	KSM-Simulator Funktionsparameter dieses Knoten

'data.function' enthält eine Liste von Argumenten für die verwendete, durch *'data.functionType'* festgelegte Funktion.

5.4 Eigenschaften von Gruppen (NodeGroup)

Table 7: NodeGroup Properties

Typ	**Schlüssel**	**Ver.**	**Beschreibung**
string	visual.caption	1+	Titel der Group im Editor.
string	visual.color	1+	Farbliches Merkmal der Group
integer	visual.location.x	1+	X-Position relativ zu übergeordneten NodeGroup
integer	visual.location.y	1+	Y-Position relativ zu übergeordneten NodeGroup

6 Dokumentation der Implementierung

Das am Anfang vorgestellte Objektmodell wird durch Interfaces im Package *de.dhbw.horb.ksm. xmlschema.api* umgesetzt.

Eine Implementierung dieser Interfaces findet sich im Package *de.dhbw.horb.ksm. xmlschema.imp* Von dieser Implementierung ist für äusseren Zugriff nur die Klasse *KSMFactory* vorgesehen, welche Methoden zum Laden und Speichern von KSM-Modellen zur Verfügung stellt.

Die Implementierung benutzt vom XML-Schema-Compiler (xjc) von JAXB generierte Klassen die im Package *de.dhbw.horb.ksm. xmlschema.generated* liegen. Das generieren wird vom Ant-Task *compile-xjc* und Annotationen im Schema gesteuert.

Der Zugriff auf das geladene Modell erfolgt auschliesslich über die in den Interfaces vorgesehen Methoden, ein direkter Zugriff ist nicht möglich.

Neben dem Quellcode in src/ gibt es im Projektverzeichniss noch das Verzeichnis test/ welches JUnit-4 Tests enthält die die Implementierung nahezu 100% abdecken. Die Tests sind dabei zum Teil im Stil des Behavior Driven Development (BDD) geschrieben unter Zuhilfename der Bibliothek mockito.

7 Vorgehensweise bei Änderungen

1. Eintragung der Änderung in diesem Dokument
2. Erhöhen der Versionszahl
3. Erstellen eines Eintrags in der Revisions Historie in diesem Dokument
4. Anpassen der Versionszahl in build.xml *project.version*
5. Erstellen einer HTML- und PDF Version von diesem Dokument (asciidoc/a2x).

8 Revisions Historie

Table 8: Revisions

Ver.	**Datum**	**Person**	**Änderung**
1	2011-03-24	Yves Fischer	Beginn der Historie, KSM Version 1
1	2011-04-26	Fischer, Dreher	Erläuterung Datentypen. Festlegung Funktionsname/Parameter von Connection's.