Model Querying with Graphical Notation of QVT Relations

Dan Li, Xiaoshan Li
Faculty of Science and Technology, University of Macau

Volker Stolz
University of Oslo, Norway
Agenda

- Motivation
- QVT Relations and QVTR-XSLT Tool
- Definition of the Graphical Query
- Mapping Queries to XPath
- Tool Implementation and Case Study
- Conclusion
(Meta-) Models (usually) represented as diagrams

Most model query languages use textual notation like OCL

Shortcomings:

- Textual notation does not work well in navigating two-dimensional diagrams of models and metamodels;
- Standard OCL is too verbose, even defining simple queries quickly leads to complex OCL expressions.
Notation Styles of Modeling Languages

- **Graphical notation**: visually attractive, easier and faster to understand(?)

- **Textual notation**: more expressiveness, compact and has less scalability problems.

- **Better choice**: combine advantage of both notations, and keep a balance between them:
  - Graphically specify structural matching;
  - Textually specify constraints, calculation, value assignment, … etc.
Contribution

- Graphical model query facility based on QVT Relations:
  - use an extension of graphical notation;
  - adopt checking semantics and pattern matching mechanism.

- Tool support for the query facility:
  - graphical editor for queries specified in graphical syntax;
  - code generator that can generate executable XSLT programs for the queries.

- on the basis of QVTR-XSLT tool (UML&FM’10).
Agenda

- Motivation
- QVT Relations and QVTR-XSLT Tool
- Definition of the Graphical Query
- Mapping Queries to XPath
- Tool Implementation and Case Study
- Conclusion
QVT Relations (QVT-R)

- One of the OMG’s standard model transformation languages (QVT: MOF 2.0 Query, View, and Transformation language).

- A declarative language with textual and graphical notation.

- The graphical notation provides a concise, intuitive way to specify transformations.

- Uses OCL to query models.
A Relation in QVT-R Graphical Notation

```
when PackageToSchema(p, s);
where prefix = cn; AttributeToColumn(c, t, prefix);
```
QVTR-XSLT Tool

- Supports the graphical notation of QVT-R.
- Two parts:
  - **Graphical Editor**: specify QVT-R transformations using the graphical notation.
  - **Code generator**: generate executable XSLT programs for transformations
- Features supported include:
  - transformation *parameters*
  - transformation *inheritance* through rule overriding,
  - *traceability* of transformation executions,
  - multiple input and output models,
  - *in-place* transformations.
Agenda

- Motivation
- QVT Relations and QVTR-XSLT Tool
- Definition of the Graphical Query
- Mapping Queries to XPath
- Tool Implementation and Case Study
- Conclusion
Abstract Syntax of the Query Facility
- **Domain** defined by a *domain pattern* (object diagram) with free *variables*, each *bound* to an element of the pattern or a property of an element.
- **where clause** as a set of predicates. Each defines a variable in the form: `<var> = <exp>`.
- **when clause** that also consists of a set of predicates in the form of `<var><op><exp>`.
- Set of *parameters* used to pass values to the query
- Dedicated variable “**result**“ for returning result of query
A query example which acquires a class that has the name given by the parameter.
A query is used to find parts of the model that fulfill given constraints specified in the query.

Queries can be called from QVT-R relations, other queries and functions.

The semantics of a query is defined as:

- for each valid binding of variables of the domain and the parameters…
- … which satisfy the when condition, there exists a valid binding of the result variable that satisfies the domain pattern and the where condition.
Agenda

- Motivation
- QVT Relations and QVTR-XSLT Tool
- Definition of the Graphical Query
- Mapping Queries to XPath
- Tool Implementation and Case Study
- Conclusion
Extensible Stylesheet Language for Transformations (XSLT) is one of the W3C standards.

A declarative rule-based programming language for transforming XML documents.

Widely used in data-intensive applications.

An XSLT stylesheet consists of a set of rule templates, each of them matches elements in source model, and produces output to the target model.

XSLT has many industrial strength implementations, and can work with most of the programming languages.
XPath

- sub-language of XSLT for selecting parts of an XML document.
- compact syntax, operates on an abstract, logical tree-structure *data model* of XML documents.
- *path expressions* navigate through the tree structure of the data model.
- functions, tests, and expressions for manipulation of strings, numbers and booleans.
- also W3C standard
Data Model of XML Documents

- **Node**
  - name : String

- **Text**
  - text : 0..1

- **Document**
  - child : 0..* \{ordered\}

- **Element**
  - parent : 0..*
  - child : 0..* \{ordered\}
  - referred : 1..*

- **Attribute**
  - attribute : 0..*

- **ValueAttribute**
  - value : String

- **Reference**
XPath Path Expressions

\begin{align*}
\langle \text{Expr} \rangle & = \langle \text{NodeExpr} \rangle \mid \langle \text{AttributeExpr} \rangle; \\
\langle \text{NodeExpr} \rangle & = \text{'}$\text{'} \ (\langle \nu_p \rangle \mid \text{'}xmiXMI\text{'} \rangle) (\langle \text{'}/'\text{'} \mid \text{'}//\text{'} \rangle \langle \text{Step} \rangle )^*; \\
\langle \text{Step} \rangle & = \langle \text{Axis} \rangle (\langle \text{'}[\text{'} \langle \text{Predicate} \rangle \text{'}\text{']\text{'}} \rangle)^*; \\
\langle \text{Axis} \rangle & = \langle \ell \rangle \mid \text{'}*\text{'} \mid \text{'}parent\text{'} \rangle \mid \text{'}xmiRefs\text{'} \langle \langle \text{RefExp} \rangle \text{'}\rangle; \\
\langle \text{AttributeExpr} \rangle & = \langle \text{NodeExpr} \rangle \text{'}/'\text{'} \text{'}@\text{'} \langle \alpha_v \rangle; \\
\langle \text{RefExp} \rangle & = \langle \text{NodeExpr} \rangle \text{'}/'\text{'} \text{'}@\text{'} \langle \alpha_r \rangle;
\end{align*}

Denoting:

- \text{xmiXMI} \space: \space \text{the document node};
- \alpha_v \space: \space \text{a value attribute};
- \alpha_r \space: \space \text{a reference attribute};
- \nu_p \space: \space \text{a parameter of the query};
- \ell \space: \space \text{an xmi label};

Functions:

- \text{parent} \space: \space \text{get the parent node}
- \text{xmiRefs} \space: \space \text{get elements by ID}
- \text{xmiLabel} \space: \space \text{get the xmi label of an element}
- \text{OCLtoXPath} \space: \space \text{get a path expression from an OCL expression.
Mapping Domain Pattern to Path Expressions

- Mapping starts from the *root* `objectTemp`
- May bind to root variable *(depth-first search)*
- Two kinds of queries:
  - **Global model query:** If *root not bound to an input* parameter, the pattern matching will search over all the model represented by the variable `xmiXML`. Then, we have `expr="$"+"xmiXML"`.
  - **Local model query:** The query retrieves information specific for a model element given as a parameter `νp`, which is bound to the *root* `objectTemp`. Then, we have `expr="$"+ νp`. 
- **Down the containment hierarchy**: Let $\ell$ be the *xmi label* for the contained element, then a child step is added as $expr = expr \ "/" + \ell$; if its *forAllSub* feature is *true*, then $expr = expr \ "//" + \ell$ for querying all descendant elements.

- **Up the containment hierarchy**: a parent step is added as: $expr = expr + \ "/" + \"parent()\"$.

- **Through normal association**: Let asso be the name of an association end, then a reference step is added as: $expr = xmiRefs(expr + \"/@\" + asso)$. 

---

**Navigating the Domain Pattern**
Generating XSLT functions for a query

- Create a variable `var` defined in the domain pattern using `<xsl:variable name=var select=expr/>` instruction.
- If there is an OCL constraint `exp` bound to the variable in the `when` clause or the pattern, let `xexp = OCLtoXPath(exp)`, then the `expr` becomes `expr + "[" + xexp+ "]"`.
- Generate `<xsl:variable/>` instructions for variables defined in the `where` clause.
- Create an XSLT function for the query by generating XSLT parameters for the `parameters of the query, and collecting all variable definitions.`
- Use an `<xsl:sequence>` instruction to return the value of the `result` variable.
<xsl:function name="my:GetClassByName">
  <xsl:param name="nm"/>
  <xsl:variable name="result" select="$xmiXMI//packagedElement[@xmi:type='uml:Class'][@name=$nm]"/>
  <xsl:variable name="cnm" select="$result/@name"/>
  <xsl:sequence select="$result"/>
</xsl:function>
Agenda

- Motivation
- QVT Relations and QVTR-XSLT Tool
- Definition of the Graphical Query
- Mapping Queries to XPath
- Tool Implementation and Case Study
- Conclusion
A UML Profile for the Query Facility

```
<<stereotype>>
Domain
[InstanceSpecification]

<<stereotype>>
Object
[InstanceSpecification]
+position: Integer
+forAllSub: Boolean

slot 0..*

<<stereotype>>
PropertyItem
[Slot]
+value: String

domain 0..1

Query
[Package]

<<stereotype>>
Parameter
[InstanceSpecification]
+sNumber: Integer

0..*

<<stereotype>>
When
[Constraint]

0..1

<<stereotype>>
Where
[Constraint]
```
Tool Implementation

- Tool Interface
- Diagram: GetSuperClass
  
  ```
  GetSuperClass
  {where=supers=GetSuperClass(sc);
   result=c->union(supers);
  }
  ```

  - Domain: c : Class
  - Parameter: sc : Class
  - Generalization: c --> sc
Use the model of UML Superstructure version 2.4.1 which includes 11 packages, 341 classes: ~9500 model elements stored in an XMI file of 1.7 MB in size.

A set of queries that are frequently used for model analysis.
Query Example: GetClsWithAssoNum

Obtains a list of classes with a given number of associations.

```
<<Query>>
GetClsWithAssoNum
{result=cls->select(ownedAttribute.association
->size()==assoNum);}
```

- **<<Domain>>**: Package
  - forAllSub
- **<<Parameter>>**: assoNum : Integer
- **cls : Class**
- **: Attribute**
- **: Association**
  - ownedAttribute
Get names of primitive attributes in a given class;

Selects by:
- owns an element of \textit{Type};
- not have any association links;
- visibility is \textit{public}.

Diagram:

\begin{itemize}
  \item \texttt{GetAttributeName}
  \item \texttt{<<Domain>>}
    \begin{itemize}
      \item \texttt{c : Class}
    \end{itemize}
  \item \texttt{<<Parameter>>}
    \begin{itemize}
      \item \texttt{c : Class}
    \end{itemize}
  \item \texttt{: Attribute}
    \begin{itemize}
      \item name = "result"
      \item visibility = "public"
    \end{itemize}
  \item \texttt{: Type}
  \item \texttt{{not} : Association}
\end{itemize}
### Experiments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Result</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountClass</td>
<td>count all classes of the model</td>
<td>331</td>
<td>78 ms</td>
</tr>
<tr>
<td>GetClsWithAssoNum(3)</td>
<td>classes with a given number of asso.</td>
<td>30</td>
<td>65 ms</td>
</tr>
<tr>
<td>GetMaxDepth</td>
<td>maximal depth of class inheritance</td>
<td>10</td>
<td>15 s</td>
</tr>
</tbody>
</table>

(Times incl. reading/writing the model to disk; compilation to XSLT not included)
A Relation in QVT-R Graphical Notation

```
<<domain>>
c : Class
  kind = "Persistant"
  name = cn

<<domain>>
t : Table
  name = cn

cl : Column
  type = "NUMBER"
  name = cn+"_tid"
  column
  key

k : Key
  name = cn+"_pk"

namespace
p : Package

when
  PackageToSchema(p,s);
where
  prefix=cn;
  AttributeToColumn(c,t,prefix);
```

source Domain
source DomainPattern
Property
Variable
when clause
where clause
RelationCallExp
ObjectTemplateExp
PropertyTemplateItem
rootVariable
target Domain
target DomainPattern
Conclusion

- Graphical query facility using adapted graphical notation and pattern matching mechanism of QVT Relations.
- Map selection criteria of queries into XPath expressions. (Still quite a bit of OCL left.)
- Queries are implemented as XSLT functions.
- Tool supports editing and execution of queries.
- Applied in model transformations (FACS’11, TTC’11&14).

http://rcos.iist.unu.edu