QVT Based Model Transformation from Sequence Diagram to CSP

Li Dan
Faculty of Science and Technology
University of Macau
Motivation

- UML sequence diagrams need to be verified and analyzed formally. One of the choices is to translate to CSP through model transformation.

- Graphical notation of QVT provides a concise, intuitive way to specify model transformations.

- XSLT is the most common and powerful language for XML transformation. We implement the QVT transformation rules using XSLT for their automatic execution.
Model Transformation

Meta-model A

Define Transformation

Conforms To

Model A

Transformation

Conforms To

Model B

Meta-model B

Apply Transformation
What is QVT

- MOF 2.0 Query, View, and Transformation language
- A standard for model transformation proposed by the Object Management Group (OMG)
- QVT is composed of 3 sub-languages
  - QVT-Operational
  - QVT-Relations
  - QVT-Core
- QVT Relations has a graphical notation.
QVT Relations

- Bidirectional, declarative language

- A transformation is specified as a set of relations (rules) between model elements of the source and the target domains.

- A relation is a transformation declare constraints that must be satisfied by the elements of the candidate models.

- The kernel technique to implement QVT Relations is the *pattern matching*. 
UML people might expect to continue the graphical tradition of class diagrams and favor a graphical notation.

Graphical specification is a higher-level view that is easier to understand and communicate than the lexical counterpart.

Can be served as good software design documentations.

* A picture is worth a thousand words *
Extensible Stylesheet Language for Transformations (XSLT) is one of the W3C standards. Provide powerful capacity that enable the rule declaration, transformation, navigation, and create of XML content. Widely used in developing web application. Supported by many commercial and open source tools, can be embedded in Java. XSLT has strong support to complex pattern matching.
XML Metadata Interchange (XMI)
- An OMG standard that specify how to produce XML documents from MOF model.
- EMF XMI is an XMI implementation, supported by many major modeling tools, such as MagicDraw and Topcased

XML Schema (XSD)
- used to define the syntactic structure of XML documents
- XMI also gives rules to produce an XML Schema from a MOF model
Our Transformation Approach

Get the best of both worlds:

- Define the transformation using QVT relations language in graphical notation

- Implement the transformation using XSLT
Steps in the Approach (1)

1. Define metamodels for both SeqD and CSP. The SeqD metamodel is in EMF XMI format.

2. Generate XSDs from SeqD and CSP metamodels

3. Specify the transformation relations (rules) using QVT graphical notation

4. Implement these QVT transformation relations as XSLT rule-based templates
5. Prepare the SeqD in CASE tools, and output it as an EMF XMI file.

6. Validate the SeqD XMI against the SeqD XSD.

7. Perform the transformation by executing the XSLT templates in an XSLT processor, output result as a CSP model.

8. Validate the CSP model using the CSP XSD.
Overall Transformation Process
SeqD Metamodel (with XMI Support)
Semantics of SeqD

- Defined as an union of order relations on the set of all the message sending and receiving actions.

- Every object lifeline has its own flow of control, performs its sequence of actions along the lifeline.

- Objects are only synchronized on the sending and receiving actions of same message.
CSP Metamodel
<table>
<thead>
<tr>
<th>Concepts of SeqD and CSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqD</td>
</tr>
<tr>
<td>- Interaction</td>
</tr>
<tr>
<td>- Lifeline</td>
</tr>
<tr>
<td>- InteractionOperand</td>
</tr>
<tr>
<td>- CombinedFragment</td>
</tr>
<tr>
<td>- opt, alt, loop, break</td>
</tr>
<tr>
<td>- Message</td>
</tr>
<tr>
<td>- MessageOccurrence</td>
</tr>
<tr>
<td>- sendEvent</td>
</tr>
<tr>
<td>- receiveEvent</td>
</tr>
<tr>
<td>CSP</td>
</tr>
<tr>
<td>- CSP</td>
</tr>
<tr>
<td>- Process</td>
</tr>
<tr>
<td>- SubProcess</td>
</tr>
<tr>
<td>- Prefix</td>
</tr>
<tr>
<td>- IFTThenElse</td>
</tr>
<tr>
<td>- ExternalChoice</td>
</tr>
<tr>
<td>- SKIP/STOP</td>
</tr>
<tr>
<td>- Event</td>
</tr>
<tr>
<td>- Channel</td>
</tr>
<tr>
<td>- Parallel Composition</td>
</tr>
</tbody>
</table>
QVT Rules for SeqD to CSP Transformation

- Interaction to CSP (*Parallel Composition of Processes*)
- Lifeline to Process
- InteractionOperand to SubProcess
- MessageOccurrence to Event
- CombinedFragment to IfThenElse
- Alt (without guard) to ExternalChoice
- Message to Channel
Rule: Interaction to CSP

```
InteractionToCSP

<<domain>>
seq : Interaction

name = sn

ownedBehavior

owner

:Collaboration

<<domain>>
csp : CSP

name = sn

where
MessageToChannel(seq,csp);
LifeLineToProcess(seq,csp);
```
Rule : Lifeline to Process

```plaintext
where
  pn = if (m = "") then cn else rn endif;
  OperandToSub(seq,pro,lfid,"SKIP");
```
Rule: MessageOccurrence to Event

FragmentToSub

fr : MessageOccurrence
  xmi : id = xid
  receivedEvent
  message
  event
  ms : Message
    name = fn
    sendEvent
  : Event
  : MessageOccurrence
  covered

Ifid : String
nextP : String

pro : SubProcess
  : Prefix
    name = pf
    target = nextP
  : Operation
    name = on
    event
    operation
  : Lifeline
    xmi : id = If2id
    name = mn
    direction = "in"
    passBy = cn

where
pn = getLifelineName(lf2id);
p2n = getLifelineName(lf2id);
mn = if (ev.operation -> isEEmpty()) then fn else onm endif;
pfnn = pn + "_" + xid.substring(1,3);
cn = lowercase(p2n)"_"lowercase(pn)"_"fn;
Rule: \texttt{Alt (without guard)} to \texttt{ExternalChoice}

\texttt{fr} : \texttt{CombinedFragment}
interactionOperator = "alt"
xmi : id = xid

\texttt{llfd} : \texttt{String}
nextP : \texttt{String}

\texttt{pro} : \texttt{SubProcess}

\texttt{sp} : \texttt{ExternalChoice}
name = snm

\texttt{sp 1} : \texttt{EmptyProcess}
target = nextP

\texttt{sp 2} : \texttt{EmptyProcess}
target = nextP

\texttt{where}
\texttt{snm} = pn + "._" + xid.substring(1,3);
OperandToSub(op1,sp1,llfd,nextP);
OperandToSub(op2,sp2,llfd,nextP);
A QVT rule is implemented as a pair of XSLT templates: a matching-template and a constructing-template.

The source domain pattern of a QVT rule is implemented as a matching-template to play the searching task in the source model.

The target domain pattern of a QVT rule is implemented as a constructing-template to create the elements of the pattern in the target model.
<xsl:template mode="InteractionToCSP"
    match="//packagedElement[@xmi:type='uml:Collaboration']
        /ownedBehavior[@xmi:type='uml:Interaction']">
    <xsl:call-template name="InteractionToCSP">
        <xsl:with-param name="sn" select="@name"/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="InteractionToCSP">
    <xsl:param name="sn"/>
    <xsl:element name="CSP">
        <xsl:attribute name="name" select="$sn"/>
        <xsl:apply-templates mode="MessageToChannel"/>
        <xsl:apply-templates mode="LifelineToProcess"/>
    </xsl:element>
</xsl:template>
An Example of SeqD
<xml version="1.0" encoding="UTF-8"/>
<CSP name="Example">
  <Channel name="o1_o2_m1" from="O1" to="O2"/>
  <Channel name="o2_o3_m2_create" from="O2" to="O3"/>
  ......  
  <Process name="O1" target="SKIP">
    <Prefix name="O1-582" target="O1-603">
      <Event name="m1" direction="!" passBy="o1_o2_m1"/>
    </Prefix>
    <IfThenElse name="O1-603">
      <bExp>c1</bExp>
      <then target="O1-603">
        <IfThenElse name="O1-643">
          <bExp>c2</bExp>
          <then target="O1-603">
            <Prefix name="O1-422" target="O1-603">
            ......
          </then>
        </then>
      </then>
    </IfThenElse>
  </Process>
</CSP>
channel o1_o2_m1, o2_o3_m2_create, o1_o2_m3, o2_o3_m4, o2_o1_m1_rt
channel o3_o2_m4_rt, o1_o2_m6, o2_o1_m1_rt

O1=o1_o2_m1!->O1-603
O1-603=if (c1) then O1-643 else O1-378
O1-643=if (c2) then o1_o2_m3!->O1-603 else o1_o2_m6!->O1-603
O1-378=o2_o1_m1_rt?->SKIP

O2 = o1_o2_m1?->O2-603
O2-603 = if (c1) then (o2_o3_m2_create!->O2-643) else O2-242
O2-643 = if (c2) then (o1_o2_m3?->O2-684)
else (o1_o2_m6?->O2-603)
O2-684 = if (c3) then (o2_o3_m4!->o3_o2_m4_rt?->O2-603)
else O2-603
O2-242 = o2_o1_m1_rt!->SKIP
CSP Generated from the SeqD (con.)

O3 = O3-603
O3-603 = if (c1) then (o2_o3_m2_create?->O3-643) else SKIP
O3-643 = if (c2) then O3-684 else O3-603
O3-684 = if (c3) then (o2_o3_m4?->o3_o2_m4_rt!->O3-603) else O3-603

O1-O2 = O1 [||{o1_o2_m1, o1_o2_m3, o1_o2_m6, o2_o1_m1 rt}||] O2

CSP = O1-O2 [||{o2_o3_m2_create, o2_o3_m4, o3_o2_m4 rt}||] O3
Thank You !