Z Family on the Web with Their UML Photos

J. Sun, J. S. Dong, J. Liu and H. Wang
School of Computing, National University of Singapore
sunjing,dongjs,liujing1,wanghai@comp.nus.edu.sg

ABSTRACT

Recent effort and success in formal methods have been concentrated on building 'heavy' tools support, such as theorem provers and model checkers. Although those tools are essential and important for applications of formal methods, in order to achieve wider acceptance, it's necessary to develop 'light' weight tools, such as web browser environment for formal specifications and projection/translation tools from formal specifications to popular industry graphical design notations. In this paper, we firstly develop web environment and browsing facilities (based on XML) for Z family languages, and then introduce techniques (based on XMI) for projecting (object-oriented) Z models to UML diagrams.

Keywords

Z/Object-Z/TCOZ, XML/XSL/XMI, UML

1 Introduction

One reason for the slow industrial take-up of formal methods (FM) is the lack of tools support and connections to the current industrial practice. Recent effort and success in FM have been concentrated on building 'heavy' tools, such as theorem provers (HOL, Isabelle, PVS ...) and model checkers (SMV, SPIN, FDR ...). Although those tools are essential and important for applications of formal methods, in order to achieve wider acceptance, it's necessary to develop 'light' weight tools, such as easy-access browsers for formal specifications and projection/translation tools from formal specifications to industry popular graphical notations. World Wide Web (WWW) is a promising environment for software specification and design because it allows sharing design models and providing hypertextual links among the models [16]. Unified Modeling Language (UML) [18] is commonly regarded as one of the dominate graphical design notation for industrial software systems modeling. It's important to develop links and tools from FM to WWW and UML so that FM technology transfer can be successful.

Z [21, 25] is a state-based formal specification language and has an active research community. Object-Z [6, 10, 19] is an object-oriented extension to Z. The integration of Object-Z with process algebra like CSP has been a recent research focus [12, 20, 17]. In this paper, using eXtensible Markup Language (XML) [23], we develop web environment and browsing facilities for Z family languages. Then, we introduce techniques for projecting (object-oriented) Z models to UML diagrams, which are based XML Metadata Interchange (XMI) [1].

The remainder of the paper is organised as follows. Section 2 briefly outlines the main approach and techniques of the paper, discusses related work and introduces the informal requirement of an example system, the light control example. Section 3 presents the design issues of the web environment and browsing facilities for Z family languages. Section 4 presents the framework and design issues of the projections from object-oriented Z models to UML class/statechart/collaboration diagrams. Section 5 concludes the paper.

2 The main approach and related background

Formal method like CafeOBJ system [14] has included an environment supporting formal specifications over networks. Pure Z notation on the web based on HTML and Java applet has also been investigated by Bowen and Chippington [3] and Cinancarini, Mascolo and Vitali [7]. HTML has been successful in presenting information on the Internet, however the lack of content information and overburdened of all kinds of tags have made the retrieval and exchange of resource become more and more difficult to perform. This paper presents a new approach of using the latest technology

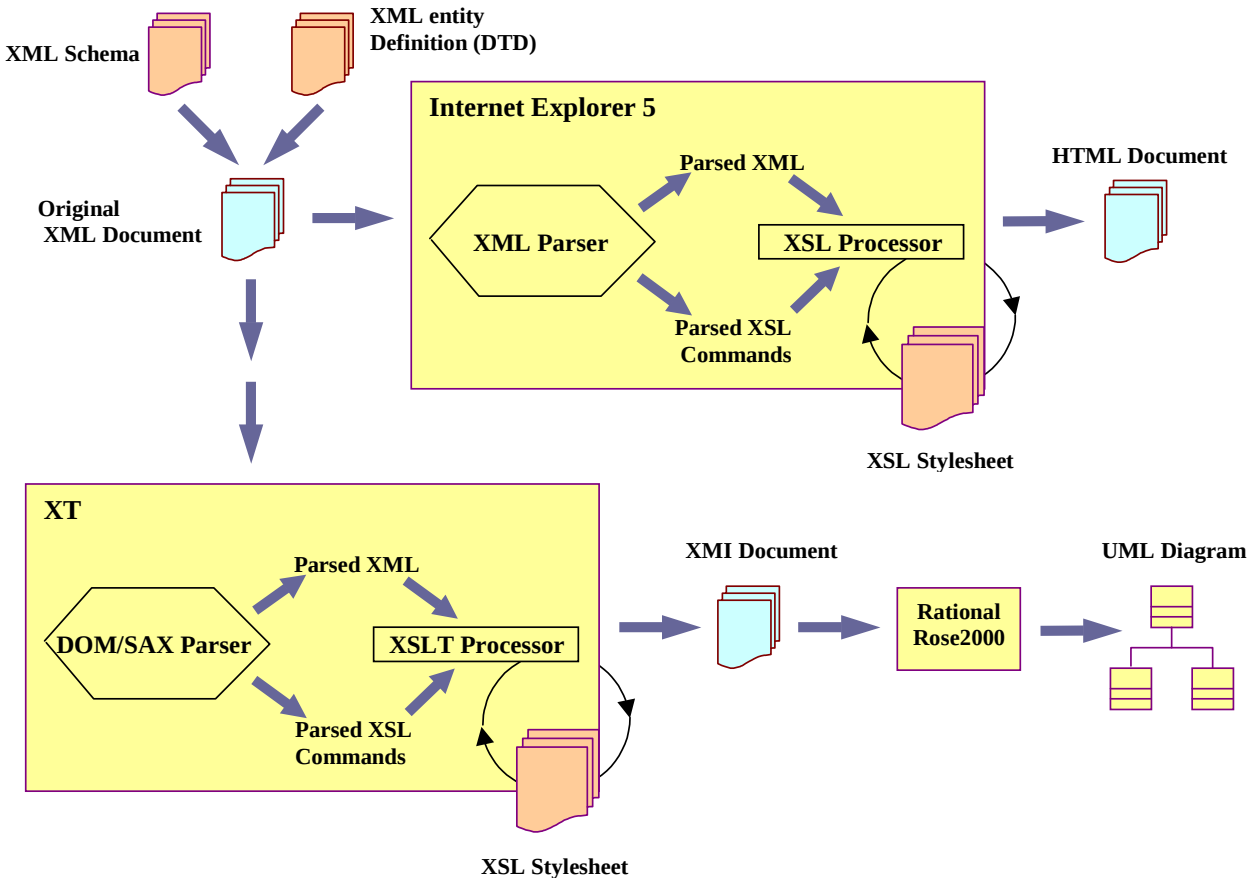

Figure 1: overall diagram

of XML and eXtensible Stylesheet Language (XSL) [24] for displaying and transforming Z family languages on the web. The users only need to follow the defined syntax in writing the XML document, the layout part is user transparent. Our XML format is inspired by the work (Java applet) of Cinancarini at el [7] and covers not only the pure Z notation but also other Z family members such as Object-Z and TCOZ. Furthermore, Z schema calculus and Object-Z/TCOZ inheritance expansions can be supported by our developed XSL environment.

The second major contribution of this work is the investigation of the semantic links and transformation environment between Object-Z/TCOZ with UML diagrams. Semantically, UML class diagrams can be identified with the signatures of the Object-Z/TCOZ classes. The states of the UML statechart can be identified with the Object-Z/TCOZ operations and processes, and the state transition links are identified with Object-Z/TCOZ pre-conditions, events and guards. Effectively, different kinds of UML diagrams can be seen as the different views of a formal model. Linking formal and informal methods has been an active area in recent years [5]. From a technical point of view, our approach lies in this area. Others have studied Z & Fusion [13] and OMT & Object-Z [11]. The main difference is that our approach further addresses the process algebra aspects (processes/events) in the projection. A new feature of UML tools—Import/Export XMI creates the potential for easy projection from formal specification (in XML) to UML (in XMI). XMI identifies standard XML Document Type Definition (DTD) to allow the exchange of UML and other object-oriented modeling technology. Basically, the XMI that Rational Rose supports is a customized XML which is consistent with the predefined "uml.dtd". We have developed an XSL environment to transform Object-Z/TCOZ (in XML) to UML XMI code.

The main process and techniques for developing Z family web environment and projections to UML are depicted by Figure 1. The following example, light control system (LCS), further facilitates the detailed discussion of our approaches.

Informal description of light control system

In most existing light control systems, all lights are controlled manually. Electrical energy is wasted by lighting rooms that are not occupied and by not adjusting light levels relative to need and daylight. LCS is an intelligent control system. It can detect the occupation of the building, then turn on or turn off the lights automatically. It is able to tune illumination in the building according to the outside light level. It gains input from sensors and actuators. The LCS presented here is a simplified version of a complex light control system [2].

The most essential functionalities of this system are when occupants enter or exit a room. For example, when a user enters a room: a motion detector senses the presence of the person, the room controller reacts by the retrieving current daylight level and turning on the light group with appropriate illumination setting. When a user leaves a room (leaving it empty): the detector senses no movement, the room controller waits "absent" time units and then turns off the light group. Finally, the occupant can directly turn on/off the light by pushing the button.

3 Z family on the Web

It is sufficient to develop a web environment for TCOZ to cover Z family languages because Z and Object-Z are subsets of TCOZ. Firstly, we define a customized XML document for TCOZ. This document is used for checking the well formedness and validity of the user input specifications in XML format. The World Wide Web Consortium (W3C) has provided two mechanisms for describing XML structures: Document Type Definition (DTD) and XML Schema. The former originates from SGML Recommendation and uses a total different syntax. XML Schema is a kind of XML file itself and is going to play the role of DTD in defining customized XML structure in the future. It has the advantages of consistency with XML syntax, data type extensibility and easy to write over DTD. We use XML Schema to define our XML structure syntax for Z family members. Part of the Z XML Schema is defined as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-
 data" xmlns:dt="urn:schemas-microsoft-com:
 datatypes">

<!-- some definition omitted -->
<ElementType name="op" content="eltOnly"
 order="seq">
  <element type="name" minOccurs="1"
   maxOccurs="1"/>
  <element type="delta" minOccurs="0"
   maxOccurs="1"/>
  <element type="decl" minOccurs="0"
   maxOccurs="*"/>
  <element type="st" minOccurs="0"
   maxOccurs="1"/>
  <element type="predicate" minOccurs="0"
   maxOccurs="*"/>
  <AttributeType name="layout" dt:type=
   "enumeration" dt:values="simpl calc"
    default="simpl"/>
  <attribute type="layout"/>
</ElementType>
<ElementType name="classdef" content=
 "eltOnly">
  <!-- some definition omitted -->
  <element type="op" minOccurs="0"
   maxOccurs="*"/>
  <!-- some definition omitted -->
</ElementType>

<!-- some definition omitted -->
</Schema>
```

From the above `op` structure example, we can see that `op` tag is an element of `classdef` which defines an operation inside the class definition. It consists of one `name` for defining the operation name, a `delta` list, a number of declarations `decl`, an horizontal line `st` and some `predicate` definitions. An attribute `layout` is defined to distinguish between vertical layout schemas `simpl` and horizontal layout schemas `calc`.

Z family languages consist of a rich set of mathematical symbols. Those symbols can be presented directly in Unicode which is supported by XML. We have defined all entities in the DTD so that users do not have to memorize all the Unicode numbers when authoring their XML documents. Part of the entity declaration DTD is defined as follows.

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

<!-- some definition omitted here -->
<!ENTITY emptyset "&#x2205;">
<!ENTITY mem "&#x2208;">
<!ENTITY nem "&#x2209;">
<!ENTITY pset "&#x2119;">
<!ENTITY fset "F">
<!ENTITY uni "&#x222a;">
<!ENTITY int "&#x2229;">
<!ENTITY subset "&#x2282;">
<!ENTITY prod "&#x2715;">

<!-- some definition omitted here -->
```

For instance, if a user wants to write a power set symbol in the specification, `&pset` can be used instead of `&#x2209`. As most existing Z specifications were constructed in LaTeX translating them to our format can

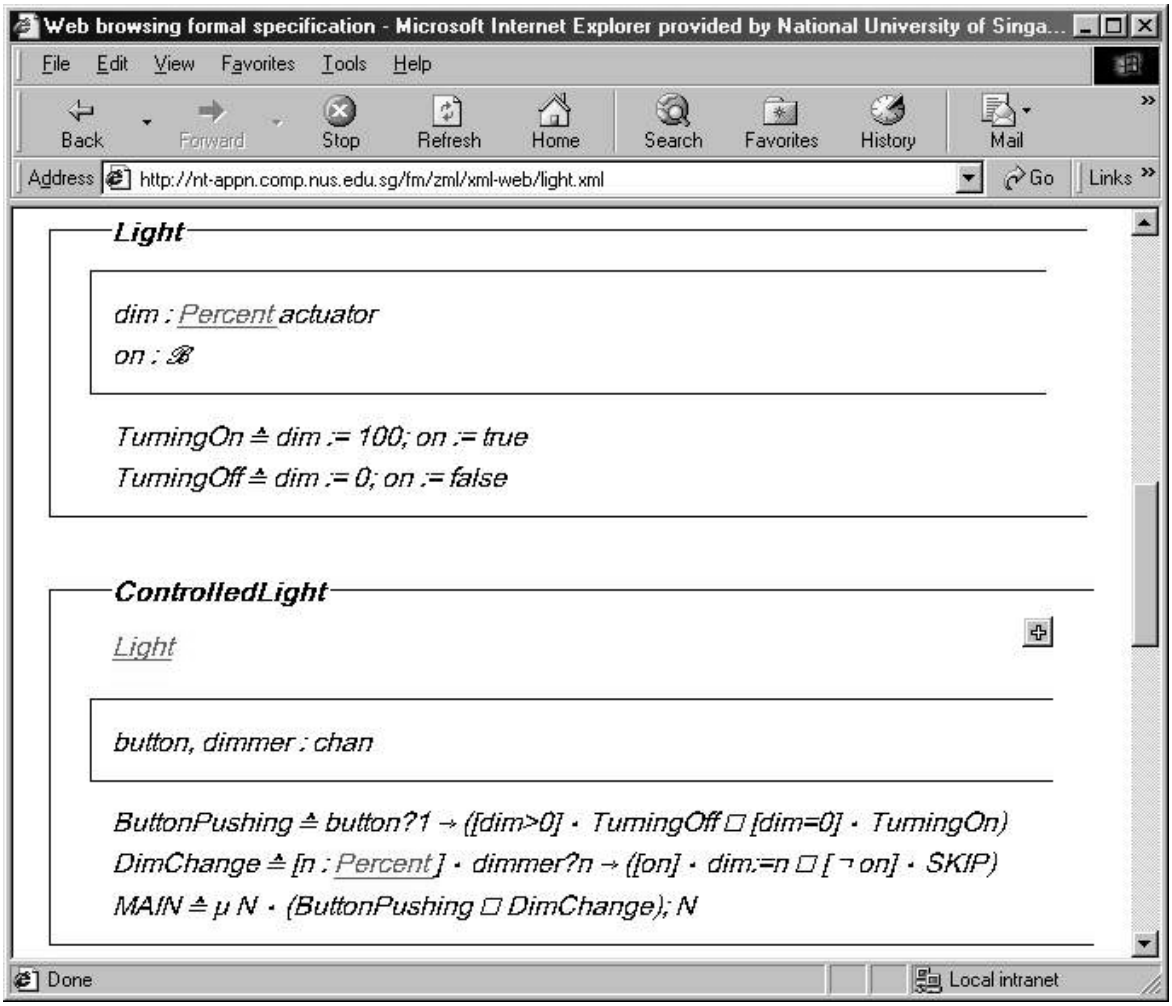

Figure 2: light on web

be a trivial task due to that each entity is given a LaTeX compatible name. DTD is chosen to define our entity declaration because XML Schema does not support entity declaration at the moment. When authoring XML files the user simply declares the name space of the XML schema and Entity DTD file as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<! --some definition omitted here -->
<!DOCTYPE unicode SYSTEM
  "http://nt-appn.comp.nus.edu.sg/fm/zml/
          unicode.dtd">
<objectZnotation xmlns="x-schema:
  http://nt-appn.comp.nus.edu.sg/fm/zml/
         objectZschema.xml"
  xmlns:HTML="http://www.w3.org/Profiles/
  XHTML-transitional">

<!-- some definition omitted here -->
</objectZnotation>
```

With the above name space links, the XML editing tools can check the validity of the user file via XML Schema definition and the DTD entity declarations. Any unspecified structures and entity symbols would be reported as a syntax error. The following is the LCS `Light` class specification in our XML format.

```
<classdef layout="simpl" align="left">
  <name>Light</name>
  <state>
    <decl>
      <name>dim</name>
      <dtype>
        <type ty="predefine">Percent
        </type>
        <type ty="basic">Actuator</type>
      </dtype>
    </decl>
    <decl>
      <name>on</name>
```

```
      <dtype>
        <type ty="basic">&bool;</type>
      </dtype>
    </decl>
  </state>
  <op layout="calc">
    <name>TurningOn</name>
    <predicate>dim := 100; on := true
    </predicate>
  </op>
  <op layout="calc">
    <name>TurningOff</name>
    <predicate>dim := 0; on := false
    </predicate>
  </op>
</classdef>
```

With a valid XML file in hand, the next step is to transform the XML file into HTML format and display it on the web. XSL is a stylesheet language to describe rules for matching and transforming XML documents. An XSL file is also in XML format and it can perform the transformation between XML to HTML, XML to XML, XSL to XSL and so on. This kind of transformation can be done on the server side or the client side. Since Internet Explorer 5 (IE5) already supports XSL technology, our environment is based on client side (browser) transformation. A partial XSL stylesheet segment for displaying operation `op` and class definition `classdef` are defined as below.

```
<xsl:template match="op[@layout='simpl']">
<html>
  <tr>
    <!-- some definition omitted here -->
    <td height="24" valign="middle" align=
    "left" nowrap="true">
      <i><xsl:value-of select="name"/></i>
      <!-- some definition omitted here -->
    </td>
    <!-- some definition omitted here -->
  </tr>
  <xsl:for-each select="delta | decl">
    <xsl:apply-templates select="."/>
  </xsl:for-each>
  <xsl:apply-templates select="st"/>
  <xsl:for-each select="predicate">
    <xsl:apply-templates select="."/>
  </xsl:for-each>
  <tr>
    <!-- some definition omitted here -->
  </tr>
</html>
</xsl:template>

<xsl:template match="classdef[@layout=
'simpl'] | classdef[@layout='gen']">
<html>
  <!-- some definition omitted here -->
  <a><xsl:attribute name="name"><xsl:value
  -of select="name"/></xsl:attribute></a>
  <!-- some definition omitted here -->
  <xsl:apply-templates select="tydef"/>
  <xsl:apply-templates select="state"/>
  <xsl:apply-templates select="init"/>
  <xsl:apply-templates select="op"/>
  <!-- some definition omitted here -->
</html>
</xsl:template>
```

XSL stylesheet file defines `match` method for each customized tag in the XML structure and describes the corresponding HTML codes. From the example above, in matching the "op" tag the XSL will display the operation name, delta list, declaration and predicates accordingly; in matching the "classdef" tag the XSL will first convert the class name into a HTML bookmark for the type reference usage and then apply the templates of drawing local type definition, state schema, initiation schema, operations and so on. To apply a template in XML is like to make a function call in programming, and each template will perform its own transformation. The users only need to write their XML files and add an URL to the defined XSL stylesheet location as below.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="http://nt-appn.comp.nus.edu.sg/fm/zml/
            objectZschema.xml"?>
```

With this link, the browser (IE5) will automatically transform XML document into desired HTML output. This process is totally user transparent and much faster than those Java applet approaches [3, 7]. For example, the `Light` class in XML format specified previously is transformed into HTML as in Figure 2.

A full demonstration of the light control system specification is available at

```
href="http://nt-appn.comp.nus.edu.sg/fm/zml/
       xml-web/light.xml.
```

From the above output we can see that the *ControlledLight* class inherits the *Light* class. This is treated as a HTML link in the presentation.

Inheritance in Object-Z/TCOZ is a mechanism for incremental specification, whereby new classes may be derived from one or more existing classes. Essentially, all definitions are pooled with the following provisions. Inherited type and constant definitions and those declared in the derived class are merged. The state and initialisation schemas of inherited classes and those declared

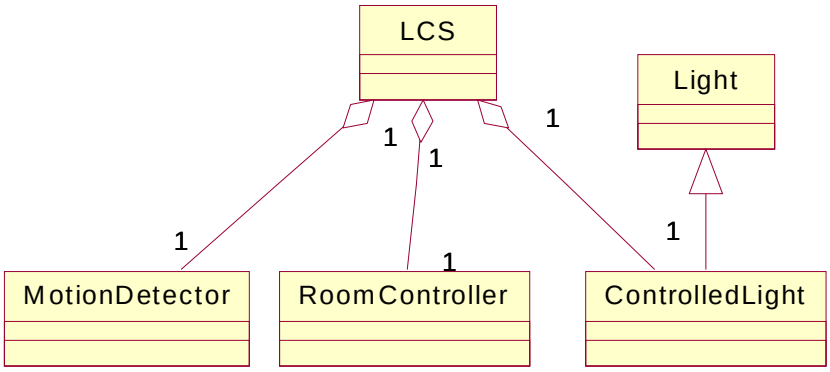

Figure 3: class diagram

in the derived class are conjoined. Operations with the same name are conjoined. Additional inheritance rules for *active class* can be found in TCOZ[9].

The aim of class inheritance and schema calculus expansion is to allow user to view the full definition of the definition. In the *ControlledLight* class case when user click the '⊞' link (opposite to the inherited class *Light* in Figure 2), the full definition of class of "ControlledLight" will be shown following the inheritance rules stated above ('⊟' link for going back to the original).

The core part of the expansion techniques uses the match facilities provided by XSL to find the corresponding definitions in the parent class and merge them in the derived class. Part of the XSL for merging the declarations in the state schema of a class is as follow.

```
<!-- some definition omitted here -->
<xsl:for-each select="//classdef[name=
context(-1)/inherit/type]/state/decl">
<!-- some definition omitted here -->
</xsl:for-each>
<xsl:for-each select="state/decl">
<!-- some definition omitted here -->
</xsl:for-each>
<!-- some definition omitted here -->
```

As we can see from above, the *select* constraint will restrict a search through the entire XML document for a match of same named class definition corresponding to the name in the *inherit* list of the current class. Then the state declaration of super class will be merged with the state schema of the present class. Thus the whole definition of *state* declarations in the derived class is displayed. Z Schema calculus can also be treated in a similar way. The next section is focused on projecting Object-Z models (in XML) to UML diagrams (in XMI).

4 UML photos

For better understanding of the Object-Z/TCOZ models, we use UML to visualize them. As introduced earlier, the Object-Z/TCOZ models can be constructed in XML format. The textual specifications of UML models are in XMI format. Based on XSL Transformations (XSLT) [22] technology, we define an XSL file to capture all translation rules from Object-Z/TCOZ XML to UML XMI. XT [8] is chosen as the XSLT processor and Rational Rose 2000 is used as the UML tool. By now we have fully implemented the visualization of static parts with UML class diagrams and are looking into the dynamic parts with UML statecharts. In our approach, all elements from the static view, such as attributes, operations, classes and their relationships (inheritance and aggregation) can be successfully captured through the transformation process.

The XML file for formal specifications and the XMI file for UML diagrams have similar structures. An XMI file has the structure as follows:

```
<XMI xmi.version="1.0">
  <XMI.header>
  <XMI.content>
  <XMI.extensions>
</XMI>
```

The `XMI.header` section includes some optional information about UML model. Elements in UML diagrams, such as classes in class diagrams and states in the statecharts, are specified in the `XMI.content` section, while their positions, colors and other displaying properties are specified in the `XMI.extensions` section.

The XSL file used in this section is similar to the XSL file used in displaying Z specification in the previous section. The template technology plays a key role in implementing the translation rules. Consider the implementation issues and the translation rules based on

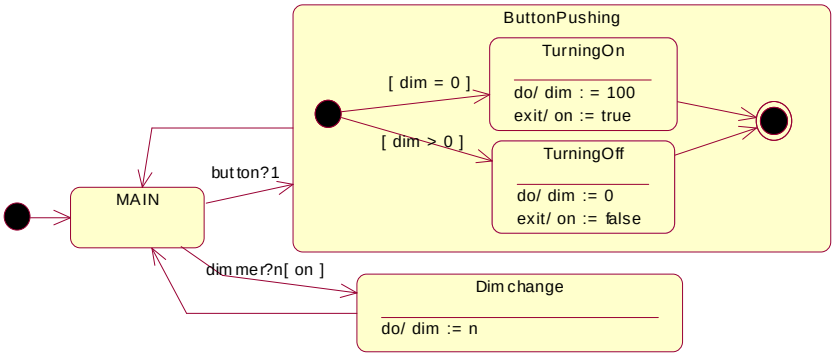

Figure 4: ControlledLight statechart

the formal model, the following guidelines are formed:

- Each class in Object-Z XML models corresponds with a class in UML XMI models. They have the same name, attributes and operations.
- If a type value in the *Inherit* part of a class matches the name of any other class in the current XML file, we regard that former class inherits the second one and illustrate the inheritance relationship between these two classes in the UML class diagram. In the case of spelling mistakes or missing reference of the *Inherit* type, we ignore the relationship.
- If a type value in the *decl* part, that is, the type of an attribute, matches the name of any class in current XML file, this is regarded as aggregation relationship between these two classes. The cardinality of the aggregation will be calculated and classified into UML aggregation ranges.

Due to the space limitation (XMI files for UML models are normally very large and complex with all details about property specifications), only the sketch of a simplified XMI unit—class **Light**, is given as an example in the paper.

```
<Class xmi. id="S.10001">
  <name>Light</name>
  <Attribute>
     <name>motion</name>
     <type>< DataType href="G.2"/>
      <!-- chan -->
     </type>
  </Attribute>
  <Attribute>
     <name>md</name>
     <type>< DataType href="G.3"/>
      <!-- (Move|NoMove) sensor -->
     </type>
  </Attribute>
  <Operation>
     <name>TurningOn</name>
  </Operation>
  <Operation>
     <name>TurningOn</name>
  </Operation>
   <DataType xmi.id="G.2">
     <name>chan</name>
  </DataType>
   <DataType xmi.id="G.3">
     <name>(Move|NoMove) sensor</name>
  </DataType>
</Class>
```

In addition to the *ControlledLight* component, LCS consists of a *MotionDetector* and a *RoomController*.

MotionDetector

...	[see details in the appendix]

RoomController

...	[see details in the appendix]

The whole light control system is defined as

LCS

$m : MotionDetector$
$l : ControlledLight$
$r : RoomCtrller$

$\text{MAIN} \mathrel{\widehat{=}} \|(m \overset{motion}{\longleftrightarrow} r \overset{dimmer}{\longleftrightarrow} l)$

The projection rules for translating TCOZ model to UML class diagrams are trivial. The UML class diagram in Figure 3 depicts the structure of the *LCS*. The relationship between *LCS* and the other three classes is aggregation. The aggregation relationship between *LCS* and *MotionDetecter* is illustrated in the following XMI segment:

```
<Association xmi.id="G.2">
  <name />
  <connection>
      <AssociationEnd xmi.id="G.3">
<name />
<type>
<xmi.idref="S.10004" />
<!--  LCS   -->
   </type>
      </AssociationEnd>
      <AssociationEnd xmi.id="G.4">
   <name />
<AssociationEnd.type>
   <xmi.idref="S.10001" />
<!--  MotionDetector   -->
      </AssociationEnd.type>
      </AssociationEnd>
  </connection>
</Association>
```

The dynamic parts of Object-Z/TCOZ model are translated into UML statechart diagram. Formal specifications for dynamic view of classes are composed of operations and events. With the statechart diagram of *ControlledLight* Figure 4, some guidelines for this translation are introduced: First of all, we consider each operation in TCOZ model as a state/substate, which may have its own actions or some fixed values for a span of time. MAIN is modeled as the state that the startstate leads to. Secondly, events and guards in TCOZ model are viewed as triggers which cause transition of states in the statechart (such as *button*?1 and $[dim = 0]$ in Figure 4), which totally match the definition of trigger in UML statechart. Thirdly, sometimes, operations can call other operations (this doesn't happen in this simple case study). In this case, the called operations serve as the substates of the former ones and they compose a composite state in the statechart.

Brief structures of a `simplestate` and a transition (from `Dimchange` to `Main`) in the statechart in XMI are:

```
<State_Machines.SimpleState xmi.id="G.11">
   <name>Dimchange</name>
<ModelElement.name>dim := n</ModelElement.name>
 </State_Machines.SimpleState>
<State_Machines.Transition xmi.id="G.7">
  <name />
  <source>
   <SimpleState xmi.idref="G.4" />
<!--  MAIN   -->
  </source>
  <target>
  <SimpleState xmi.idref="G.11" />
  <!--  Dimchange   -->
  </target>
  <trigger>
  <SignalEvent xmi.idref="G.28" />
  <!--  dimmer?n   -->
  </trigger>
</State_Machines.Transition>
```

Other than class diagram for static view and statechart for dynamic view, collaboration diagram can be used to capture the synchronous/asynchronous communications between different parts. In TCOZ, *chan* borrowed from CSP is used for synchronous communication interface while *sensor*/*actuator* are used for asynchronous communication interfaces. These can be well presented in collaboration diagrams, such as *motion* in Figure 5. In the collaboration diagram, the direction of message passing through *chan* or *sensor*/*actuator* can be readily decided. For *sensor*/*actuator* case, the message is passed from actuator to sensor. For *chan* case, the message flow direction is simply decided by the tagged symbol ! or ?.

5 Conclusion

The first contribution of this paper is the development of new techniques of using the latest technology of XML and XSL for displaying and browsing Z family languages on the web. Extensible browsing facilities such as Z schema calculus and Object-Z/TCOZ inheritance expansions can be readily supported by our developed Z-XSL environment. Our ideas for putting Z family on the Web can be easily adopted by other formal methods, such as VDM/VDM++. In fact, since TCOZ includes most Timed CSP constructs, its web environment can be used for CSP/Timed-CSP specifications. Perhaps this may create a new culture for constructing formal specification on the web (XML) rather than in LaTeX. We hope it can be the starting point for developing a standard XML environment for all formal notations — Formal specification Markup Language (FML).

The second contribution of this work is the investigation of the semantic links and web transformation environment (XSLT) between Object-Z/TCOZ (in XML) with UML diagrams (in XMI). Although we have some ideas on Object-Z/TCOZ behaviour projections to statecharts, the development of the Web environment for systematic transformation from Object-Z/TCOZ to statechart/collaboration diagrams remains a challenge. The engineering work for developing further techniques and putting these techniques into commercial case tools perhaps requires involvement from industry partners. We are currently in contact with various UML tools vendors.

6 Acknowledgements

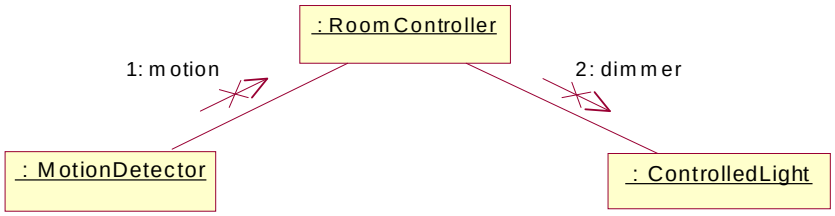

Figure 5: Collaboration diagram

This work is supported by the academic research grants, *Integrated Formal Methods* (R-252-000-050-107) and *Adding Formality to UML* (R-252-000-076-112) from National University of Singapore.

REFERENCES

[1] Xml metadata interchange. http://www-4.ibm.com/software/ad/standards/xmi.html, 2000.

[2] E. Borger and R. Gotzhein. The light control case study. J.UCS Special Issue on Requirements Engineering Volume 6 (7), 2000.

[3] J. P. Bowen and D. Chippington. Z on the Web using Java. In Bowen et al. [4], pages 66–80.

[4] J. P. Bowen, A. Fett, and M. G. Hinchey, editors. *ZUM'98: The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, 24–26 September 1998*, volume 1493 of *Lect. Notes in Comput. Sci.* Springer-Verlag, 1998.

[5] A. Bryant and L. Semmens. *Methods Integration'96*. Springer, Leeds, UK, 1996.

[6] D. Carrington, D. Duke, R. Duke, P. King, G. Rose, and G. Smith. Object-Z: An object-oriented extension to Z. In S. Vuong, editor, *Formal Description Techniques, II (FORTE'89)*, pages 281–296. North-Holland, 1990.

[7] P. Ciancarini, C. Mascolo, and F. Vitali. Visualizing Z notation in HTML documents. In Bowen et al. [4], pages 81–95.

[8] James Clark. Xt version 19991105. http://www.jclark.com/xml/xt.html, 1999.

[9] J.S. Dong and B. Mahony. Active Objects in TCOZ. In J. Staples, M. Hinchey, and S. Liu, editors, *the 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98)*, pages 16–25. IEEE Computer Society Press, December 1998.

[10] R. Duke and G. Rose. *Formal Object Oriented Specification Using Object-Z*. Cornerstones of Computing, (series editors: R. Bird and C.A.R Hoare). Macmillan, March 2000.

[11] S. Dupuy, Y. Ledru, and M. Chabre-Peccoud. Translating the OMT dynamic model into Object-Z. In Bowen et al. [4], pages 347–366.

[12] C. Fischer. *Combination and Implementation of Processes and Data: from CSP-OZ to Java*. PhD thesis, University of Oldenburg, Gemany, January 2000.

[13] R. France, J-M. Bruel, M. Larrondo-Petrie, E. Grant, and M. Saksena. Towards a Rigorous Object-Oriented Aalysis and Design Method. In Hinchey and Liu [15], pages 7–16.

[14] K. Futatsugi and A. Nakagawa. An Overview of CAFE Specification Environment. In Hinchey and Liu [15].

[15] M. Hinchey and S. Liu, editors. *the IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[16] G. Kaiser, S. Dossick, W. Jiang, and J. Yang. An Architecture for WWW-based Hypercode Environments. In R. Adrion, A. Fuggetta, and R. Taylor, editors, *The 19th International Conference on Software Engineering (ICSE'97)*, pages 3–13, Boston, USA, 1997. IEEE Press.

[17] B. Mahony and J.S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, February 2000.

[18] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Languauge Reference Manual*. Addison-Wesley, 1999.

[19] G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.

[20] G. Smith and J. Derrick. Specification, refinement and verification of concurrent systems - an integration of object-z and csp. *Formal Methods in System Design*, 2000. (To appear).

[21] J.M. Spivey. *The Z Notation: A Reference Manual*. International Series in Computer Science. Prentice-Hall, 2nd edition, 1992.

[22] World Wide Web Consortium (W3C). Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt, 1999.

[23] World Wide Web Consortium (W3C). Extensible markup language (xml). http://www.w3.org/XML, 2000.

[24] World Wide Web Consortium (W3C). Extensible stylesheet language (xsl). http://www.w3.org/Style/XSL, 2000.

[25] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International, 1996.

A LCS Formal Model (Partial)

The basic data types are defined:

$$Illumination == 1..10000\,\mathsf{lux}$$
$$Percent == \{0\} \cup 10..100$$

MotionDetector

$motion$: **chan**
md : $(Move \mid NoMove)$ **sensor** [motion sensor]

$NoUser \mathrel{\widehat=} md?Move \rightarrow motion!1 \rightarrow User \;\Box$
$\quad md?NoMove \rightarrow \textsc{Wait}\ 1\,\mathsf{s};\ NoUser$
$User \mathrel{\widehat=} md?NoMove \rightarrow motion!0 \rightarrow NoUser \;\Box$
$\quad md?Move \rightarrow \textsc{Wait}\ 1\,\mathsf{s};\ User$
$\textsc{Main} \mathrel{\widehat=} NoUser$

The purpose of the MotionDetector is to detect any movement and send appropriate signals to the room controllers. Timing behavior of the motion detector (when movement/nomovement is detected, system will be notified) is captured in the MAIN process

The satisfaction relation between dim value for light and outdoor light illumination is specified:

$satisfy : Percent \leftrightarrow Illumination$

The room controller is used to control the light. It is specified as:

RoomController

$dimmer, motion$: **chan**
$odsensor$: $Illumination$ **sensor**
$absenT : \mathbb{T}$
$olight$: $Illumination$

Adjust

$dim! : Percent$ **on** $dimmer$

$dim!$ *satisfy* $olight$

$Ready \mathrel{\widehat=} motion?1 \rightarrow On$
$Regular \mathrel{\widehat=} \mu\, R \bullet [n : Illumination] \bullet$
$\quad odsensor?n \rightarrow olight := n;\ Adjust;$
$\quad dimmer!dim \rightarrow R$
$On \mathrel{\widehat=} Regular \triangledown motion?0 \rightarrow OnAgain$
$OnAgain \mathrel{\widehat=} (motion?1 \rightarrow On) \triangleright \{absenT\}\ Off$
$Off \mathrel{\widehat=} dimmer!0 \rightarrow Ready$
$\textsc{Main} \mathrel{\widehat=} Off$

LCS

$m : MotionDetector$
$l : ControlledLight$
$r : RoomCtrller$

$\textsc{Main} \mathrel{\widehat=} \|(m \overset{motion}{\longleftrightarrow} r \overset{dimmer}{\longleftrightarrow} l)$