Lightweight Checking for UML Based Software Development

Clare Gryce, Anthony Finkelstein and Christian Nentwich
Department of Computer Science
University College London, London WC1E 6BT
{c.gryce,a.finkelstein,c.nentwich}@cs.ucl.ac.uk

1 Introduction

This paper describes the application of xlinkit, a generic tool for managing the consistency of distributed documents, to consistency checking in the context of UML modelling and software development. In particular, we report how we have developed an application of xlinkit to the UML Core package Well-Formedness Rules [9]. We first give an overview of how xlinkit works. We then expand on the specific application of xlinkit to the UML, and discuss our report generator 'Pulitzer'. Simple examples are used for illustration.

2 xlinkit

xlinkit [5] is a framework for checking the consistency of distributed heterogeneous documents. It comprises a language, based on first order logic, for expressing constraints between documents, a document management system and an engine that checks the documents against the constraints.

xlinkit leverages XML and related technologies, notably XPath [2], XLink [3] and the DOM [1] to provide a flexible service that can be deployed as a standalone program, a web service or a distributed checker. It makes use of hyperlinks as a diagnostic to pinpoint inconsistent elements by linking them.

In order to explain xlinkit's operation, we take as our example the case of two developers working independently on the same system, with their data stored on different machines. One is specifying a UML model and the other working on a Java implementation. We wish to check the simple constraint that each Class in the UML model has to be implemented as a Java class. This constraint must be satisfied if the model and the implementation are to be consistent.

To make documents and constraints available for checking, xlinkit provides 'document sets' for including documents and 'rule sets' for selecting constraints. Figure 1 shows the document set for our example, it contains Document elements that instruct xlinkit to load the documents directly from the given URL. The Set element allows a further document set to be included. Constraints are similarly assembled into 'rule sets', which may contain further rule sets. These sets can be arbitrarily distributed, hence it is possible to assemble useful rule sets from third parties. You can add or remove rule sets in a check to vary the 'strictness' or emphasis of a check. The 'fetcher' attribute of the set is used to deal with input that is not in XML format.

```
<DocumentSet name="UMLandJava">
    <Description>
        A UML model and some Java files
    </Description>

    <Document href="http://host1/UMLmodel.xml"/>
    <Document href="http://host2/Main.java" fetcher="JavaFetcher"/>
    <Set href="http://host2/ClassSet.xml"/>
</DocumentSet>
```

Figure 1: Example document set

In this case, a Java class is loaded and translated into a DOM tree using a grammar annotated with tree production rules.

```
<consistencyrule id="r1">
    <forall var="c" in="//UML:Class">
        <exists var="j" in="/java/class">
            <equal op1="$c/@name" op2="$j/@name"/>
        </exists>
    </forall>
</consistencyrule>
```

Figure 2: Example constraint

xlinkit uses a language based on first order logic that has been adapted for use with XML and has been restricted to make it decidable. Our example constraint can be written in the XML encoding of the xlinkit language as shown in Figure 2. The consistency rule makes no reference to where the elements indicated in the constraint should be retrieved from – the language is location transparent, and data sources could be arbitrarily distributed. At run time, xlinkit will apply the XPath expressions in the constraint to all the documents in the document set, and thus build up a set of nodes to be checked.

```
<xlinkit:LinkBase docSet="DocSet.xml" ruleSet="RuleSet.xml">
    <xlinkit:ConsistencyLink ruleid="rule.xml#id('r1')">
        <xlinkit:State>consistent</xlinkit:State>
        <xlinkit:Locator xlink:href="http://host1/UMLmodel.xml#//UML:Class[1]"/>
        <xlinkit:Locator xlink:href="http://host2/Main.java#/java/class" fetcher="JavaFetcher"/>
    </xlinkit:ConsistencyLink>
    <xlinkit:ConsistencyLink ruleid="rule.xml#id('r2')">
        <xlinkit:State>inconsistent</xlinkit:State>
        <xlinkit:Locator xlink:href="http://host1/UMLmodel.xml#//UML:Class[2]"/>
    </xlinkit:ConsistencyLink>
</xlinkit:LinkBase>
```

Figure 3: Resulting hyperlinks

An important contribution of xlinkit is the definition of a new semantics for this restricted form of first order logic. We go beyond the boolean evaluation of such formulae that return 'true' or 'false' and specify a new semantics in terms of hyperlinks that link consistent or inconsistent elements. The evaluation strategy is designed to determine which parts of a formula to 'blame' depending upon whether the formula returns true or false for a given element. It discards irrelevant information and links together only those nodes that have directly contributed to the boolean result of the formula. In this way, xlinkit is able to provide strong diagnostic information

to the user, who can see which combination of elements causes an inconsistency.

Figure 3 shows two hyperlinks in an xlink linkbase that may have been generated from our sample rule – the XPaths have been abbreviated for clarity. In this case, as by default, xlinkit has generated links between consistent elements, 'consistent links' and between inconsistent elements, 'inconsistent links'. It can be seen that the consistent UML class has been linked to the Java class it conforms to and that the inconsistent UML class has been identified, but has not been linked to anything – because there is no matching Java class. In both cases, the 'ruleid' attribute of the consistency link can be used to see which rule generated the link.

3 Application of xlinkit to the UML

We have described how xlinkit can be used to check the consistency of distributed, heterogeneous documents. In this section, using further examples, we illustrate how xlinkit may be applied to the UML. We show how the service can be used to check UML models against the Well-Formedness Rules, and then how the results are presented to the user by our report generator, 'Pulitzer'.


```
<UML:Association xmi.id="G.0" name="{Store-Manager}">
  <UML:AssociationEnd xmi.id="G.2" name="manages">
    <UML:AssociationEnd.type><UML:Class xmi.idref="S.7"/></UML:AssociationEnd.type>
  </UML:AssociationEnd>
  <UML:AssociationEnd xmi.id="G.3" name="store Mgr">
    <UML:AssociationEnd.type><UML:Class xmi.idref="S.8"/></UML:AssociationEnd.type>
  </UML:AssociationEnd>
</UML:Association>
<UML:Association xmi.id="G.1" name="{Store-Warehouse}">
  <UML:AssociationEnd xmi.id="G.4" name="supplied By">
    <UML:AssociationEnd.type><UML:Class xmi.idref="S.7"/></UML:AssociationEnd.type>
  </UML:AssociationEnd>
  <UML:AssociationEnd xmi.id="G.5" name="supplies">
    <UML:AssociationEnd.type><UML:Class xmi.idref="S.9"/></UML:AssociationEnd.type>
  </UML:AssociationEnd>
</UML:Association>
<UML:Class xmi.id="S.8" name="Manager"/>
<UML:Class xmi.id="S.7" name="Store"/>
<UML:Class xmi.id="S.9" name="Warehouse"/>
```

Figure 4: Sample UML fragment and XMI encoding

UML models are MOF (Meta-Object Facility) [8] compliant, and so can be mapped to XML documents using the XMI (XML Metadata Interchange) [10]. The XMI specification includes a DTD that describes the syntax of this mapping. XMI was designed as a standard to facilitate the distribution and exchange of UML documents between developers and CASE tools. The XMI exporters included in such CASE tools can be exploited as a convenient means of transforming UML models to XMI files, that may then be submitted to xlinkit for checking.

The flexibility of xlinkit enables checks to be performed for various purposes. 'Rules' might describe local constraints affecting a system under development, or conformance with a public domain standard. Document sets (including UML documents) and rule sets can be assembled accordingly to meet these varying user needs.

Figure 4 shows a screenshot of a very basic UML model of three related Classes and two Associations and a fragment of the exported XMI file. For the sake of clarity, only the basic elements representing the model have been included in this fragment. Some of the properties of the model elements are represented as mark-up. The relationship between the elements is described using the 'xmi.id' and 'xmi.idref' attributes and the Well-Formedness Rules place constraints on the nature of this relationship. For example, the Well-Formedness Rule in the Core package 2.5.3.8 Classifier [3] states: *"No opposite Association Ends may have the same name within a Classifier"*. This constraint can be readily expressed using the xlinkit rule language, as shown in Figure 5.

```
<globalset id="classifiers" xpath="http://UML:Classifier.feature/.. |
 /XMI/XMI.content/UML:Model//UML:DataType"/>
<globalset id="associations" xpath="http://UML:Association[@xmi.id]"/>
<consistencyrule id="cs3">
    <forall var="c" in="$classifiers">
        <forall var="x" in="($associations/UML:Association.connection/
        UML:AssociationEnd[UML:AssociationEnd.type/*[1]/@xmi.idref=$c/@xmi.id])/../
        UML:AssociationEnd[UML:AssociationEnd.type/*[1]/@xmi.idref!=$c/@xmi.id]">
             <forall var="y" in="($associations/UML:Association.connection/
             UML:AssociationEnd[UML:AssociationEnd.type/*[1]/@xmi.idref=$c/@xmi.id])/../
             UML:AssociationEnd[UML:AssociationEnd.type/*[1]/@xmi.idref!=$c/@xmi.id]">
                  <implies>
                       <not>
                            <same op1="$x" op2="$y"/>
                       </not>
                       <notequal op1="$x/@name" op2="$y/@name"/>
                  </implies>
             </forall>
         </forall>
    </forall>
</consistencyrule>
```

Figure 5: Well-formedness rule in xlinkit

The example demonstrates how xlinkit uses XPath to extract the elements referred to by the rule. It also shows how the xlinkit rule language can be used to express more complex constraints. Figure 6 shows a link generated by the evaluation of our sample constraint against an XMI document that violates it. In this case, xlinkit has been set to return only inconsistent links to the linkbase. The link includes 3 locators. These locators include hyperlinks back to the offending document, pointing out the classifier, and the opposite association ends that have violated the rule.

Over 35 of the Core Well-Formedness Rules have been successfully written using the xlinkit rule language. These rules can be seen at `http://www.xlinkit.com`. The rules have been tested individually and in combination, using UML models to generate appropriate XMI test documents.

```
<xlinkit:LinkBase docSet="DocumentSet.xml" ruleSet="RuleSet.xml">
    <xlinkit:ConsistencyLink ruleid="Classifier.xml#/consistencyruleset/consistencyrule[3]">
        <xlinkit:State>inconsistent</xlinkit:State>
        <xlinkit:Locator number="1" xlink:href="TestXMIDocs/cs3Inconsistent.xml#/
             XMI/XMI.content[1]/UML:Model[1]/UML:Namespace.ownedElement[1]/UML:Package[1]/
             UML:Namespace.ownedElement[1]/UML:Class[10]"/>
        <xlinkit:Locator number="2" xlink:href="TestXMIDocs/cs3Inconsistent.xml#/
             XMI/XMI.content[1]/UML:Model[1]/UML:Namespace.ownedElement[1]/
             UML:Package[1]/UML:Namespace.ownedElement[1]/UML:Association[1]/
             UML:Association.connection[1]/UML:AssociationEnd[1]"/>
        <xlinkit:Locator number="3" xlink:href="TestXMIDocs/cs3Inconsistent.xml#/
             XMI/XMI.content[1]/UML:Model[1]/UML:Namespace.ownedElement[1]/
             UML:Package[1]/UML:Namespace.ownedElement[1]/UML:Association[5]/
             UML:Association.connection[1]/UML:AssociationEnd[2]"/>
     </xlinkit:ConsistencyLink>
</xlinkit:LinkBase>
```

Figure 6: Hyperlink generated by xlinkit

4 Report Generation

Our report generator, 'Pulitzer' creates a diagnostic report for the user. A report contains information that explains and details the inconsistencies detected. The user can then make an informed decision about what action to take in order to remove, or accommodate the inconsistencies. This 'tolerant' approach to inconsistency is a deliberate feature of xlinkit. In the context of UML modelling, distributed development might mean, for example, that different models represent a system at different stages of development. The developer may want to know about any inconsistencies in their model at any time, but it may not always be convenient or appropriate to have their resolution enforced.

```
<report:fragment ruleid="NewClassifier1.3.xml#/consistencyruleset/consistencyrule[3]" id="cs3">
     <report:linkdescription state="inconsistent">
          <report:define var="classifierType" path="name($locator[1])"/>
          <report:define var="classifierName" path="$locator[1]/@name"/>
          <report:define var="association1Name" path="$locator[2]/../../@name"/>
          <report:define var="association2Name" path="$locator[3]/../../@name"/>
          <report:define var="associationEnd1Name" path="$locator[2]/@name"/>
          <report:define var="associationEnd2Name" path="$locator[3]/@name"/>
          <li/>
          The <b>%@=gettext(substring-after($classifierType,"UML:"))%</b>
         '<b>%@=gettext($classifierName)%</b>' has more than one opposite Association End called
         '<b>%@=gettext($associationEnd1Name)%</b>'; in the Association
         '<b>%@=gettext($association1Name)%</b>', and in the Association
         '<b>%@=gettext($association2Name)%</b>'.
     </report:linkdescription>
</report:fragment>
```

Figure 7: Report-fragment used by Pulitzer

Pulitzer reads the linkbase and generates an HTML report. It uses the hyperlinks contained in the locators as the initial context for XPaths that access selected elements in the original document. References to elements that are most useful for diagnostic reporting can therefore be included in the report. The XPath expressions that select the desired elements are included in 'report-fragments'. Report-fragments are XHTML [11] documents that combine standard

HTML elements with elements defined in the Pulitzer report language. Figure 7 shows the report-fragment for our current example.

Figure 8: Report page generated by Pulitzer

The rule used in our example is a simple one. However, some of the Well-Formedness Rules are rather more 'wordy', and an inexperienced UML modeller may find them hard to interpret. As an example, consider the rule 2.5.3.24 Method [5], *"If the realized Operation has been overridden one or more times in the ancestors of the owner of the Method, then the Method must realize the latest overriding. (That is, all other Operations with the same signature must be owned by ancestors of the owner of the realized Operation"*. This is rather cumbersome to read. Figure 8 shows how Pulitzer presents information about an inconsistency that refers to this rule. By using natural language and naming particular elements in the document containing the inconsistency, Pulitzer offers the user a diagnostic tool that does not assume an in-depth knowledge of the semantic concepts underpinning the UML. In this way, Pulitzer benefits the novice UML modeller by increasing their understanding of the semantics of the notation, as well as promoting confidence in the consistency of their models.

5 Model Interchange

Recent work on xlinkit as applied to the UML has used the popular CASE tool TogetherSoft 6.0 for UML modelling, and for the export of XMI 1.1 documents. This work has revealed various practical limitations of the modelling tool and the exporter. These seemingly trivial limitations place severe restrictions on the consistency checking that can be supported.

The limitations of TogetherSoft 6.0 as a tool for UML modelling are concerned with the repre-

sentation of the'higher level' and the less frequently used semantic constructs of the UML, for example, the 'discriminator' attribute used to assign a meta-type classification to a Classifier. Although such constructs are set out in the UML 1.4 specification, TogetherSoft 6.0 does not always support them.

Much more significant in context of xlinkit, are the limitations of TogetherSoft 6.0 as a tool for the export of XMI 1.1 documents. Whether the limitations lie with the underlying repository or with the actual exporter has not been established. However, the implication for consistency checking is the same. In practice the user might reasonably expect that any document generated through the transformation or export of their UML model will be consistent with that model.

TogetherSoft 6.0 exhibits two kinds of limitation in this respect. Firstly, certain model elements are represented in the exported file in a way that is not consistent with the source diagram, or the semantics of a model element. Secondly, other model elements are not represented at all in the exported file. Whilst some such model elements might be considered as falling into the 'high level' or 'seldom used' category described in the previous paragraph, others are commonly used parts of the UML. The lack of representation of these model elements in the exported file would severely hinder its further processing or use as part of the development lifecycle.

6 Conclusions and Future Work

We have demonstrated how xlinkit currently supports the checking of UML documents against the Well-Formedness Rules for the static elements of the UML. Future work will extend the service, to include the rules for the dynamic elements of the metamodel. This will enable consistency checking between different model types. In this way, xlinkit will facilitate the checking of models that represent multiple viewpoints of a system. We also plan to use SVG [4] to increase the usability of 'Pulitzer', by including simple graphics in the report to illustrate inconsistent elements in a way familiar to developers. The successful presentation of UML models using SVG has already been demonstrated [7].

Other current related work has shown the application of xlinkit to the UML based development of systems based on Enterprise JavaBeans [6].

Finally, we are investigating the interactive repair of documents as rules get violated. By statically creating repair actions from formulae, it is possible to derive the minimum set of actions that can remove an inconsistency. Using our consistency checker and this repair action generator, it will be possible to provide comprehensive consistency management support for distributed software development. We already have a working prototype of such a system.

Acknowledgements

We would like to thank Jim Arlow whose ideas have informed the development of our inconsistency reports.

References

[1] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification. W3C Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium, October 1998.

[2] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. Recommendation http://www.w3.org/TR/1999/REC-xpath-19991116, World Wide Web Consortium, November 1999.

[3] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0. W3C Recommendation http://www.w3.org/TR/xlink/, World Wide Web Consortium, June 2001.

[4] J. Ferraiolo et al. Scalable Vector Graphics (SVG) 1.0. Candidate Recommendation http://www.w3.org/TR/2000/CR-SVG-20001102, World Wide Web Consortium, November 2000.

[5] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2(2):151–185, May 2002.

[6] C. Nentwich, W. Emmerich, and A. Finkelstein. Flexible Consistency Checking. Research note, University College London, Dept. of Computer Science, 2001. Submitted for Publication.

[7] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman. BOX: Browsing Objects in XML. *Software Practice and Experience*, 30(15):1661–1676, 2000.

[8] Object Management Group. *The Meta Object Facility 1.3*. Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, March 2000.

[9] Object Management Group. *Unified Modeling Language Specification*, March 2000.

[10] Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA. *XML Metadata Interchange (XMI) Specification 1.1*, November 2000.

[11] XHTML Working Group. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). Technical report, World Wide Web Consortium, August 2002.