

XML Encoded Reverse Engineering of Java to UML

C. R. Russell and R.G. Dewar

Department of Computer Science, School of Mathematical and Computer Sciences, Heriot-Watt University, Riccarton, Edinburgh, EH14 4AS, Scotland, UK.

{c.r.russell | r.g.dewar} @hw.ac.uk

Abstract

This paper introduces an XML encoded reverse engineering transformation from Java to the Unified Modeling Language (UML). We explore the relationship between an XML based representation of Java, namely JavaML, and an XML based representation of UML, XMI. A series of XSLT templates are then described that reverse engineer Java to UML Class diagrams. By exploiting XML technologies, this approach demonstrates the opportunities for simple, standardised and adaptable conversions between code and design information, within a software development and maintenance environment.

1. Introduction

Today's corporate world is dynamic and software artefacts have to be adaptable to keep pace with business needs. Hence there is a need for tools to automate the modification and development of existing systems.

An essential procedure involved with such software maintenance efforts is code understanding. However this process can be time-consuming and tedious possibly made more difficult by a lack of good documentation. It is often the case that maintainers of code are not the original designers [1]. As such, programmers spend large amounts of time deciphering other people's code. In fact, it is estimated that 50% of a software engineer's time is spent on maintenance tasks involving information searching and program understanding [2].

Reverse engineering techniques allow analysts to produce design models, which can be used to discover the underlying program architecture and behaviour. These abstractions aid the program understanding process by allowing the software engineer to quickly gain a picture of the overall system. Additionally, reverse engineering plays an important role in facilitating code reengineering and evolutionary development [1, 3].

Source code representation is an important issue in software analysis, it influences how easily analysis and maintenance tools can extract, process and exchange the program data. Traditional representation has been through Abstract Syntax Trees (AST) [4]. However XML based representations seem to present a more useful form

of source code. Mamas and Kontogiannis [5] argue that this format allows for more open data exchange and exploits the abundance of XML tools and technologies, which make it easier to traverse, edit and analyse code. McArthur et al [6] complement this work with an extensible tool for XML-encoded source code representation.

Since the Unified Modeling Language (UML) [7] has been accepted as a standard, by the Object Management Group (OMG), it has become established as a principle means for modelling software systems. Although many tools exist that reverse engineer code into UML models, Gogolla and Kollmann [8] observe that, "a uniform standardized process for the redocumentation of Java programs with UML diagrams that furthermore aims at an extensive coverage of the notational features of the UML is still missing".

This paper highlights the opportunities to standardise the reverse engineering process that result from adopting XML based techniques. In addition, the work emphasises the power and simplicity of XML translation schemes in a software development and maintenance environment.

The paper is organised as follows: in sections 2 and 3 the technologies involved in our approach are discussed; section 4 describes the implementation of the XML encoded reverse engineering tool; and finally we draw conclusion and highlight future directions for this research.

2. Java Markup Language (JavaML)

An XML based representation of Java source code, the Java Markup Language (JavaML), is presented by Badros [9, 10]. He has developed a converter using the Jikes Java compiler that transforms Java source code into the Java Markup Language (JavaML). The result is a Java program delimited by tags that describe the elements of the code. Badros maintains that this representation offers a number of benefits over classical text representation. With respect to reverse engineering, it is easier to parse and analyse by utilising the mass of existing XML related tools.

The rules that structure Java source code are inherently mirrored in the hierarchy of JavaML elements. As a result the structure of a program is presented

directly through the way in which JavaML elements are nested.

It is worthwhile noting that the JavaML elements embody all the existing source information. Hence anything else, required for another format, must be either generated by the transformation or deduced from this source information. The following provides a simple example of some Java code (Figure 1) and its equivalent representation in JavaML (Figure 2).

```
class Person
{
    private String firstName;
    private String lastName;
    private Car myCar;

    public void setName(String n){
        firstName=n;
    }
}
```

Figure 1. Simple Java Class (Code 1).

```
<java-source-program name="Person.java">
<class name="Person">
<field name="firstName" visibility="private">
    <type name="String"/>
</field>
<field name="lastName" visibility="private">
    <type name="String"/>
</field>
<field name="myCar" visibility="private">
    <type name="Car"/>
</field>
<method name="setName" visibility="public">
    <type name="void" primitive="true"/>
    <formal-arguments>
        <formal-argument name="n" id="frmarg-20">
            <type name="String"/>
        </formal-argument>
    </formal-arguments>
    <statements>
        <assignment-expr>
            <lvalue>
                <var-set name="firstName"/>
            </lvalue>
            <var-ref name="n" idref="frmarg-20"/>
        </assignment-expr>
    </statements>
</method>
</class>
</java-source-program>
```

Figure 2. JavaML Representation of Figure 1.

3. UML, XMI, PGML and ArgoUML

UML seems an obvious choice as a form of abstract representation for Java reverse engineering given that both: a tight mapping exists between UML and OO languages; and it is now familiar to many software engineers [11].

XML Metadata Interchange (XMI) is a standard produced by the OMG [12]. It merges the UML standard

with the accepted form of information exchange, XML. The result creates a standard format for the open interchange of design information [13]. Relevant to this work, it offers a standardised method to store and transfer the information that is displayed through UML diagrams.

Alternatively, a translation scheme could be developed to convert into other exchange formats, for instance, GXL [14]. However, this work focuses on demonstrating the benefits of establishing translation schemes, between XML-encoded software artefacts.

ArgoUML is an object-oriented design tool that supports the use of UML for software development [15]. It is also an Open Source Development Project that stores UML models with the XMI standard. The Precision Graphics Markup Language (PGML) [16] is used to save the graphical representation of these models.

The Ophelia project [17] is working to integrate ArgoUML into a suite of software engineering tools. At the moment, the environment includes a parse-oriented reverse engineering facility. We have augmented this functionality with a transformation-based reverse engineering method.

4. Reverse Engineering Transformation

The need for automated software maintenance tools and the role that reverse engineering techniques play in program understanding have been highlighted. In addition, UML has been presented as a suitable modelling language for representing design abstractions. Indeed, UML has the major benefits of being both an industry standard and supported by XMI. Additionally, the advantages of XML based program representation, in particular JavaML, have been introduced.

The key observation is that a missing link exists between JavaML representation of source code and XMI representation of design information. This work concentrates on exploring the relationship and mappings between these formats. In essence, this mapping is the reverse engineering process. JavaML represents the marked up source code (i.e. no abstraction or loss of detail) while XMI should contain only the marked up UML design information (i.e. an abstraction). To achieve standardized reverse engineering, a complete translation scheme that re-documents the JavaML in XMI must be developed.

Initially the project has only been concerned with creating a class model in order to concentrate on and demonstrate the XML techniques involved.

A reverse engineering tool has been developed that extracts design and structure information from Java source files and generates the relevant XMI file. The tool

allows the class model to be visualised, in ArgoUML, by generating a PGML file that relates to the XMI model.

4.1. Transformation Language

Two main methods to transform XML exist; these are XSL Transformations [18] and the Document Object Model (DOM) [19]. While XSLT describes the state of the transformed document in relation to the original document, DOM allows manipulation of the tree structure.

Since the XMI and JavaML vocabularies are completely different it is sensible to declare the state of the output XMI, using XSLT, as opposed to implementing a procedure to modify every element.

The basis of a mapping exists between these two formats. To implement this with DOM would require the development of procedures that reconstruct the tree according to the mapping rules. Alternatively, with XSLT we can directly describe the translation, leaving the tree construction to the XSL processor.

Finally, the main motivation to use XSLT instead of DOM is that by using XSLT the translation becomes adaptable. Changing the process merely requires altering the stylesheet, no recompilation is necessary.

4.2. XMI Class Diagram Components

While designing, it was important to understand the structure of these two formats (JavaML and XMI) and their relationship. Significantly, however, only a subset of the JavaML and XMI schemas are relevant to the project problem.

Adopting a pragmatic approach, initial analysis involved generating UML models with ArgoUML and examining the equivalent XMI document. This identified three distinct components required in the XMI model of a UML Class Diagram:

- class
- association
- data type

For each component, it is essential to categorise its elements depending on whether they must be **mapped**, **deduced** or **generated**. Mapped, indicates a transfer of information directly from JavaML to XMI. Deduced, refers to information that is implicit in JavaML but must be made explicit in XMI during the translation. Finally generated, specifies elements that have no connection to the JavaML input and must be generated. Such a categorisation could form the basis of a framework for creating translation schemes between XML-encoded software artefacts.

4.3. XSLT Templates

Several XSLT templates have been designed that implement these JavaML to XMI translations, which are all combined within an overall stylesheet.

In order to use the Argo environment to display the reverse engineered model, a PGML file must be generated that relates to the XMI model. To display a model with PGML, every class, association and association end must be referenced by a group element. Another style sheet implements this and generates a PGML file for a given XMI file.

Figure 3 highlights how the prototype utilises the specified style sheets and generates an Argo project.

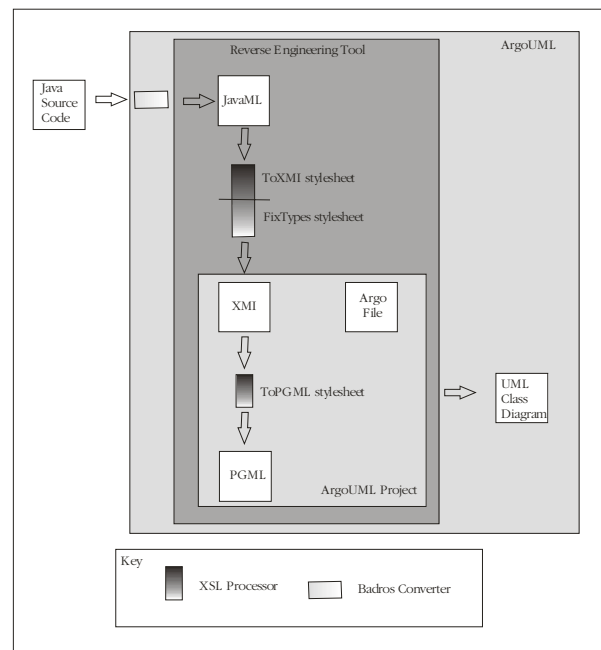


Figure 3. Prototype System View.

Unfortunately, space limitations prevent us showing actual transformations graphically realised in modeling applications.

5. Conclusions

This paper introduces an XML based transformation of marked up Java source code to marked up design information. Our initial work aims to highlight the benefits and opportunities that could result from extending XML encoding throughout the automated software environment.

A key difference from existing reverse engineering techniques, that must be emphasised, lies in the data gathering operation. Traditional methods depend upon parsing, which involves grammar rules, whereas our approach simply requires transforming. With XML

encoding, grammar rules are already implicit within the input document structure thus allowing focus to be placed on the desired transformation.

It is worth reiterating that the XSLT style sheets used in the transformation are declarative. As a result, our approach is adaptable and extensible; any alterations or additions to the process simply require modified or new templates, no recompilation.

Ultimately, as XML based source code representations become more prevalent, a library of adaptable XSLT style sheets could prove useful in the automation of the software design process.

This work should serve as a proof of concept for future translations between XML-encoded software artefacts. In fact, Alves-Foss *et al.* [20] have recently highlighted such a notion in their future work, yet our investigations have already shown its feasibility.

6. Future Work

Possibly the most significant extension to this work would result from applying our approach to the development of automated round-trip engineering capabilities. A library of XSLT style sheets could be implemented that describe the possible mappings to and from design information and source code. Also, this approach presents opportunities to overcome the key problem of consistency, involved with round-trip engineering, by exploiting X-Diff tools [21].

Indeed, incorporating XML tools with our approach would offer considerable advantages. Tools could determine and compare design and code metrics to check consistency. Additionally, XML technologies could allow elements of software that are affected by design changes to be easily referenced and modified.

Finally, as more languages become XML encoded, our approach could be applied to translate among languages as well as between alternative design information representations.

7. Acknowledgements

Thanks to the members of the Ophelia team, in particular Mike Smith and Alan Smith, for their support and advice. Thanks also to Hunter Davis for his feedback. Lastly we greatly appreciate the work of Greg Badros.

8. References

[1] E. J. Chikofsky and J. H. Cross II. "Reverse Engineering and Design Recovery: A Taxonomy". *IEEE Software*, 7(1):13-17, Jan 1990.

[2] R. S. Pressman. *Software Engineering, A Practitioner's Approach*. McGraw Hill, 1997.

[3] J. Bowen, P. Breuer, and K. Lano. "A compendium of formal techniques for software maintenance". In *Software Engineering Journal*, September, 1993.

[4] A. V. Aho, R. Sethi and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[5] E. Mamas and K. Kontogiannis. "Towards Portable Source Code Representations Using XML". In *Proceedings of 7th Working Conference on Reverse Engineering (WCRE 2000)*, pages 172-182. IEEE, Brisbane, Australia, November 2000.

[6] G. McArthur, J. Mylopoulos and S. K. K. Ng. "An Extensible Tool for Source Code Representation Using XML". In *Proceedings of 9th Working Conference on Reverse Engineering (WCRE 2002)*, pages 199-210. IEEE, Richmond, Virginia, USA, October 2002.

[7] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.3, June 1999*. Object Management Group, <http://www.omg.org>, 1999.

[8] M. Gogolla and R. Kollmann. "Application of UML and their Adornments in Design Recovery". In *Proceedings of 8th Working Conference on Reverse Engineering (WCRE 2001)*, pages 81-90. IEEE, Los Alamitos, 2001.

[9] G. J. Badros. JavaML Home Page. <http://www.cs.washington.edu/homes/gjb/JavaML>.

[10] G. J. Badros. "JavaML: A Markup Language for Java Source Code". In *Proceedings of 9th International World Wide Web Conference (WWW9)*. Amsterdam, Netherlands. May 2000.

[11] M. Gogolla and R. Kollmann. "Re-Documentation of Java with UML Class Diagrams". In E. Chikofsky, editor, *Proceedings of 7th Reengineering Forum, Reengineering Week 2000 Zurich*, pages REF 41-REF 48. Reengineering Forum, Burlington, Massachusetts, 2000.

[12] OMG: XML Metadata Interchange (XMI) Version 1.1. November 2000, Object Management Group. <http://www.w3.org/TR/xmi>

[13] S. Brodsky. "XMI Opens Application Interchange". IBM, March 1999.

[14] R. C. Holt, A. Winter and A. Schurr. "GXL: Towards a Standard Exchange Format". In *Proceedings of 7th Working Conference on Reverse Engineering (WCRE 2000)*, pages 162-171. IEEE, Brisbane, Australia, November 2000.

[15] J. Robbins. "Cognitive Support Features for Software Development Tools". 1999. http://argouml-tigris.org/docs/robbins_dissertation

[16] N. Al-Shamma and others. Precision Graphics Markup Language (PGML). W3C Note, April 1998. <http://www.w3.org/TR/1998/NOTE-PGML>

[17] Ophelia Project Home Page. <http://www.cee.hw.ac.uk/ophelia>

- [18] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999.
<http://www.w3.org/TR/xslt>
- [19] A. Le Hors and others. Document Object Model (DOM) Level 2 Core Specification Version 1.0. W3C Recommendation, November 2000.
<http://www.w3.org/TR/DOM-Level-2-Core>.
- [20] J. Alves-Foss, D. Conte de Leon and P. Oman.
“Experiments in the Use of XML to Enhance Traceability Between Object-Oriented Design Specifications and Source Code”. In *Proceedings of 35th Hawaii International Conference on System Sciences (HICSS 2002)*, pages 276-284. IEEE, Big Island, Hawaii, 2002.
- [21] XML Diff and Merge Tool.
<http://www.alphaworks.ibm.com/tech/xmldiffmerge>