

An XML-based Framework for Language Neutral Program Representation and Generic Analysis

Raihan Al-Ekram and Kostas Kontogiannis
Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
Email: {rekram | kostas}@swen.uwaterloo.ca

1. Introduction

XML applications [1] are becoming increasingly popular to define constrained data in XML format for special application areas by means of DTD or XML Schema definition. In pursuit there is a growing momentum of activities related to XML representation of source code in the area of program comprehension and software re-engineering. The source and the artifacts extracted from a program are necessarily structured information that needs to be stored and exchanged among different tools. This makes XML to be a natural choice to be used as the external representation formats for program representations.

Various XML applications are proposed to represent the source code of different programming languages. But most of these representations are at the Abstract Syntax Tree (AST) level, which is tightly coupled with the language grammar and requires development of different tools for different languages to perform the similar type of analysis. Moreover AST abstracts the program at a fine level of granularity and hence not suitable to be used directly for sophisticated analysis. Representations at higher level abstractions like intra-procedural and inter-procedural flow and dependence graphs are required for such analysis.

As such, in this paper we propose XML applications for language neutral representation of programs at different levels of abstractions and based on them we present a program representation framework in order to facilitate the development of generic program analysis tools.

2. Source Code Representation using XML

Simic and Tolnik [2] explores the prospects of representing source code using XML in place of classical plain text format. There is a spectrum of levels of granularity at which programs are represented. AST is the source code representation formalism at the finest level of details. An AST is a

hierarchical representation of the derivations of the source code from its grammar. Such tree-like structures with information in the nodes and edges can very easily be represented in XML.

Badros [3] proposes the Java Markup Language (JavaML) to represent the ASTs of Java programs. The JavaML is defined by an XML DTD, where the elements represent the structure of the AST and most if the source code information are stored as attributes on the element tags. Collard et. al. [4] describes a technique to convert the C++ source code into an XML representation called the Source Code Markup Language (scrML). In this technique the tags are superimposed on the source code to explicitly describe the syntax structure of the code, keeping the original code as it is. As a result the non syntactic elements like comments and formatting informations are preserved. McArthur et. al. [5] presents the XMLizer tool to transform source code of several programming languages into their respective XML. The PLIXML, PascalML and the JavaML are defined to represent PL/IX, Pascal and Java source code respectively. The XMLizer uses a technique to selectively parse the code and generate partial ASTs.

3. Generic AST Representations

To enable portability of the representations and building generic tools to perform similar type of analysis on the ASTs of different programming languages, the AST representations should be decoupled from the grammars of the particular programming languages. This can be achieved by developing generic domain models for ASTs of programming languages that share similar features.

Mamas and Kontogiannis [6] proposed the Object-Oriented Markup Language (OOML) as an aggregated and generic AST representation for the object-oriented language family. They provided the mappings to generate OOML from JavaML and CppML representations instead of directly manipulating the source code. Zou and Kontogiannis [7] in their work

proposed a generic domain model for representing the ASTs of procedural languages in XML.

4. Higher Level Representations

While the AST level representations are useful for some type of analysis, they are not directly usable for sophisticated analysis like data flow analysis, program slicing, architecture recovery etc. Representations of higher level program abstractions like intra-procedural and inter-procedural flow and dependence graphs are required for such analysis. These graph data structures are abstractions in terms of the control flow and data flow of the program. We define XML sub-languages for these higher level representations.

The CFGML is defined by means of a DTD to represent a Control Flow Graph (CFG) in XML. The CFGML consists of two types of elements – one to describe the basic blocks and the other for the flows of control from one basic block to another. Each basic block element consists of a sequence of elements for the statements that comprises the block.

The PDGML is defined by means of another DTD to represent a Program Dependence Graph (PDG). The PDGML also consists of two types of elements – one to describe the program statements and the other for the dependencies among them. There can be two kinds of dependency element – one to represent the control flow, labeled true or false and the other for the data flow, labeled with the variable causing the dependency.

The CGML, defined to describe Call Graphs (CG), also consists two types of elements. The first element consists of elements to describe the functions or procedures, abstract data types, files or modules and the other element consists of elements to represent the caller-callee relationships among the functions, usages of abstract data types by functions, containment of functions in the files etc. These relationships are augmented with additional attributes.

The basic program constructs that are shared by all these representations are described by another DTD called the FactML, stored in a separate XML file and then referred from the XML files for the CFG, PDG or CG using XLink. These basic constructs are data types, variables, statements, functions and the relationships among them. These higher level ML representations can be generated by manipulating of the AST level ML representations using DOM API or XQuery.

5. Representation Framework

We propose a framework, presented in Figure 1, to represent program artifacts at different levels of abstractions that facilitate language neutral and generic way of building program analysis tools.

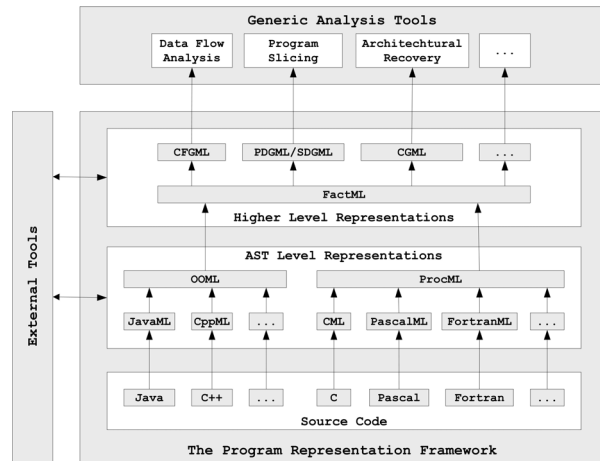


Figure 1: Program Representation Framework

The framework consists of three layers of abstractions – the source code, the AST level and the higher level representations. A generic tool can be built on top of the framework to perform a particular type of analysis on the source code written in any programming language. For example a generic data flow analysis tool can be written to work on the CFGML or a single slicing tool can be written to use the PDGML to perform slicing on source code of any language. All the representations being XML can easily be transformed to and from any other format using XSLT or XQuery technology in order to facilitate import/export with other external tools.

6. References

- [1] <http://xml.coverpages.org/xmlApplications.html>
- [2] Hrvoje Simic and Marko Topolnik. Prospects of Encoding Java Source Code in XML. Conference of Telecommunications, 2003.
- [3] Greg J. Badros. JavaML: A Markup Language for Java Source Code. International World Wide Web Conference, 2000.
- [4] Michael L. Collard, Huzefa H. Kagdi and Jonathan I. Maletic. An XML-based Lightweight C++ Fact Extractor. International Workshop on Program Comprehension, 2003.
- [5] Gregory McArthur, John Mylopoulos and Siu Ng. An Extensible Tool for Source Code Representation Using XML. Working Conference on Reverse Engineering, 2002.
- [6] Evan Mamas and Kostas Kontogiannis. Towards Portable Source Code Representations using XML. Working Conference on Reverse Engineering, 2000.
- [7] Ying Zou and Kostas Kontogiannis. Incremental Transformation of Procedural Systems to Object Oriented Platforms. Computer Software and Applications Conference, 2003.