Designing and Building Static Analysis Tools for C Programs Represented as XML

Hayato Kawashima (010035)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 2000

Keywords: XML, CASE tool platforms, data integration, DTD design.

1 Background

In software development processes, there are many products, which are all documents to complete software. For example, they are specification diagrams, design documents, programs, memorandums, manuals and so on. In software development and maintenance, such products are repeatedly referred to and modified.

As products are getting more large-scale and complicated, software development and maintenance are difficult for hand works. Therefore, support by computers, that is CASE(Computer Aided Software Engineering) tools, is required. But the cost of developing CASE tools is very high. One reason is because internal data in CASE tools is usually not available to other tools, although such data can be shared among CASE tools. As a result, most CASE tools have their own individual parsers and analyzers, resulting in low maintainability. It is a key issue to find or develop some technology to facilitate data sharing, or exchanging among CASE tools in an elegant and cost-effective manner.

2 Purpose

To solve this issue, CASE tool platforms which are basic software systems to offer common data and common functions for CASE tools, have been proposed and built(i.e.,Sapid). In this paper, our aim is to show that our approach of applying XML(Extensible Markup Language) to CASE tool platforms is useful in the sense that a well-designed DTD(Document Type Definition) for some products as well as XML technologies including DOM(Document Object Model), XSLT(Extensible Stylesheet Language Transformations) and XML parsers will greatly cut the cost of developing CASE tools.

Our goal is to build a flexible, useful and uniform data interchange format for CASE tools, which is a key issue to make it much easier to develop CASE tools. XML has a great potential for such data interchange formats, since XML has both advantages of plain text and a structured format.

Especially we think data integration among downstream parts is important for software development, since most of the cost of software development is the cost of software testing and maintenance, and there are still few applications of XML for downstream parts.

3 Advantages of XML for CASE Tools Development

The idea of data integration for CASE tools is not new. For example, CDIF (CASE Data Interchange Format) and PCTE (Portable Common Tool Environments) have already been proposed. But, unfortunately, these technologies have not yet come into wide use in CASE tools. Thus, the quest for the ideal format is still continuing.

XML can be an effective tool for our goal, since XML has the following positive characteristics.

- Program structure can be represented as natural nesting of XML tags. Relationships among programs can be represented as ID/IDREF links.
- Even incomplete data can be stored as an XML document as long as the data is represented as plain text, although the XML document might not be valid.
- XMI(XML Metadata Interchange Format) allows us to handle UML diagrams as XML documents, so XML could bridge the great gap between the upstream and downstream parts of software development.

4 Experimental Implementation of CASE Tool Platform for ANSI C

As a first step to our goal, we targeted only ANSI C programs. We selected ANSI C from full-fledged programming languages to verify that XML could be practically useful for CASE tools development. The implemented CASE tool platform consists of the following three components.

- **ACML : ANSI C Markup Language**
  A schema describing the syntactic structure and static semantics for ANSI C code.
- **XCI : Experimental C Interpreter**
  A translator from ANSI C codes to ACML-tagged documents.
- **Developed CASE tools with XML technologies**
  An unparser, a program slicer and a cross-referencer as experimental applications.

At first, an ANSI C program is translated to an ACML-tagged document by XCI. Then the ACML-tagged document is handled to compute program slices and display source code, using DOM or XSLT through the XML parser.

On DTD designing, it was not quite trivial to determine the degree of abstraction or the granularity of the representation. We took a fine-grained approach for ACML; not only functions and statements, but also all language constructs including literals and variables

are tagged with ACML. This is because most advanced program slicers or cross-referencers require the syntactic details.

On the other hand, JavaML(Java Markup Language) developed by Badros is not based on underlying grammar, which is used for parsing. That JavaML's approach is not a fine-grained one. Depending on software products, modeling on coarse granularity such as JavaML may be suitable. But for supporting downstream parts, it is lack of ability.

By the way, at the beginning of XCI development, we modified the design of the internal data in XCI, which became the cause to define ACML. XCI was designed to have the API to allow programmers to directly access the internal AST(Abstract Syntax Tree). This is the similar way as used in Sapid. In our view, such API costs much for programmers to learn and depends on XCI's internal implementation. Therefore, to solve these problems, ACML was introduced, and changed to output ACML-tagged programs.

5 Experiment on Implementation of CASE Tools

We conducted two experiments on implementation of CASE tools based on the CASE tool platform using XML. One was to implement a program slicer based on Weiser's algorithm, the other was to implement a cross-referencer using Web browsers. These CASE tools are appropriate examples to measure effectiveness of our CASE tool platform, for the following reasons:

- These CASE tools are much important and necessary for all of CASE tools.
- Both of them handle relationships among program elements.

In the experiments, it took only 0.5 month each by a programmer to implement the program slicer and the cross-referencer based on the CASE tool platform, whereas it took about 2 months by a programmer to implement an ANSI C parser and static semantics analyzer of XCI. Thus, roughly speaking, we saved 2 months each in implementation of the program slicer and the cross-referencer, since we did not have to reimplement the ANSI C parser and static semantics analyzer by using ACML and XCI.

6 Discussions

We had valuable experiences through the experiments. First, we again recognized that we need to elaborate ACML repeatedly through development of many applications like the implementation of the program slicer and the cross-referencer. For example, we found ACML lacks an attribute context that tells how many sibling nodes there are before the node, which is very convenient for tree walking, especially for tree ascending.

Second, we found that we need to build class libraries for CASE tools that can be commonly used, which include

- To extract all statements with a given identifier.
- To find the innermost statement including a given identifier.

although DOM and XSLT are quite useful.

Third, we anew recognized that the DTD is quite useful for CASE tools development, since the development could be completed with little interaction between developers, due to the function of ACML as a specification for data interchange. Surely, in our experiments, the XCI developer and the CASE tools developer required usually the DTD to define ACML.

7 Conclusion and Future Works

In this paper, we have considered a CASE tool platform for ANSI C based on XML. To investigate how useful XML is for the CASE tool platform, first we defined ACML, which can describe the AST and static semantics such as types, symbols, and so on. Then we experimentally implemented a program slicer and a cross-referencer. We had a good result; it took only 0.5 month each by a programmer to develop them.

Roughly speaking, we saved 2 months, since it took 2 months to develop an ANSI C parser and static analyzer for XCI and we did not have to reimplement the ANSI C parser and static semantics analyzer in implementing the program slicer and the cross-referencer.

We will extend ACML to support a lot of kinds of information. Especially, we have a plan to study the following:

- To extend ACML for dynamic program slicers.
- To extend ACML to support other similar languages like C++ or Java.
- To develop other CASE tools(e.g., test case generators) using ACML.
- To extend ACML to describe preprocessor directives, lexical information (e.g., comments, coding styles), system calls, and inline assemblers.
- To relate ACML and XMI to bridge the upstream and downstream processes in software developments.
- To define the proper representation of incomplete(buggy) programs, and the intention of programmers.