An XML-based Cross-Language Framework

Arno Puder

San Francisco State University
Computer Science Department
1600 Holloway Avenue
San Francisco, CA 94132
arno@sfsu.edu

Abstract. We introduce XMLVM, a Turing complete XML-based programming language based on a stack-based, virtual machine. We show how XMLVM can automatically be created from Java class-files and .NET's Intermediate Language. While the programmer is never directly exposed to XMLVM, we provide tools based on XMLVM for tasks such as cross-language functional testing or code migration.

Overview

XMLVM is a Turing complete, fine-granular XML-based programming language. XMLVM uses a generalized virtual machine model that allows us to translate Java byte code as well as .NET executables (which are also based on a virtual machine concept) to XMLVM. We therefore manage to embrace both the Java world as well as the .NET world with one XML-based language. Every instruction understood by the virtual machine, there is one XML-tag that represents this instruction. XMLVM is based on a fine-granular syntax which means that the complete syntax of XMLVM is accessible to an XML-parser.

To facilitate the generation of XMLVM, we have implemented various translators. The idea is to hide the complexities of XMLVM from the programmer who will only "see" his or her high-level programming language. We have written two translators: the first one converts a Java class file to XMLVM and the second one converts a .NET Intermediate Language program to XMLVM (see Figure 1).

XMLVM programs can readily be mapped to other high-level programming languages. This translation can easily be done by an XSL-stylesheet that maps XMLVM-instructions one-to-one to the target language. Since XMLVM is based on a simple stack-based machine, we simply mimic a stack-machine in the target language. For those high-level languages that do not support a goto-statement (such as JavaScript), we can remove those goto-statements and replace them with a combination of a loop- and multi-level exit-statements.

Applications

Middleware defines a software layer between the network and the application to facilitate the development of distributed applications. Any middleware technol-

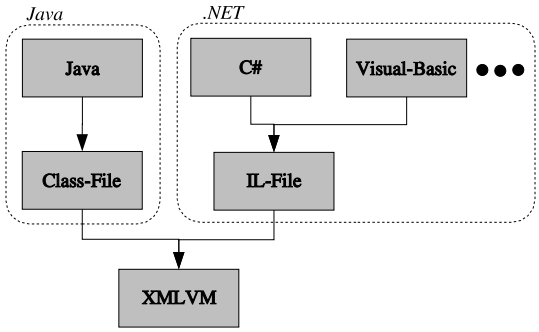

Fig. 1. Generating XMLVM.

ogy supports different programming languages. The way XMLVM is used here is to translate functional tests written in Java to XMLVM and then mapping XMLVM to C++. Two XMLVM transformations – both written as XSL-stylesheets – are applied to change the API. The first transformation adapts the CORBA-API. The second transformation does the same except for the JUnit API. The resulting XMLVM program is then translated to C++ by a third stylesheet.

The transformation chain explained above works well for many functional tests that ordinarily would have to be written for all languages supported by CORBA. The resulting C++ functional tests are not efficient from a runtime perspective (due to the overhead of mimicking a stack-machine), but performance is not important for functional testing. Further details can be found in [2].

We have used XMLVM in another project called XML11, which is an abstract windowing protocol inspired by the X11 protocol developed by MIT. In order to reduce latencies for interactive applications, XML11 also includes a code migration framework that allows to migrate part of the business logic to the end-device. We use XMLVM to translate the business logic to a language that is supported by the end-device, which is done by applying an appropriate XSL-stylesheet to the XMLVM program. E.g., for web browsers the business logic is translated to JavaScript which is universally supported by all major browsers including Microsoft's Internet Explorer. Further details can be found in [1].

References

1. Arno Puder. XML11 - An Abstract Windowing Protocol. *PPPJ*, 2005.
2. Arno Puder and Limei Wang. Cross-Language Functional Testing for Middleware. In *TestCom*, LNCS, Montreal, 2005. Springer.