An XML based intermediate language for a compiler infrastructure

Stephan Bange

December 2, 2005

1 Introduction

In this work a XML based intermediate language for a compiler frontend will be designed and implemented. It will be integrated into a compiler and will be based on an existing form of intermediate language called ICode. This will be done to grant a clean separation between two of the compilers phases and to make the internal structures of the ICode more obvious for performance measurements and testing.
The practical result of this work will be an interface that will be able to export the datastructure ICode into XML and to reimport this XML files back to ICode. XML was chosen because of its ability to represent treelike structures like the ICode. To find inconsistencies we will use XML Schema to check the structure of the exported ICode against a schema.

2 Basics

This chapter will describe the basics that will be used in this work. I will start with a short description of XML and a more specific description of the main elements of XML Schema. With XML Schema I will go into more detail for elements that I will need in my work. I will describe the general purpose of intermediate languages and I will give a short description two other forms of these languages. As last part of this chapter I will roughly describe the concepts of the ICode that will be exported from and imported into XML.

2.1 XML and XML Schema

- Short introduction into XML
- XML Schema
- XML as intermediate language

2.2 Common forms of intermediate languages

- Purpose of intermediate languages
- N-Tuple Notation
- Abstract Syntax Trees

2.3 The ICode intermediate language

- High level elements (Application, SubApplication, CodeUnit, Files)
- Element in a file (Functions, Operations, Macros ...)
- Multi phase optimization
- Elements that are not included in the current implementation

3 Concepts

The purpose of this work is the design of a XML based intermediate language that is capable of representing the ICode in such completeness that it can be regenerated from the XML data. Therefore the general concepts of how things will be represented in XML will be described here. The integration into an existing compiler will be done by an software component called XMLICodeInterface. This component is separated into two subcomponents namely Import and Export. The concepts of these two subcomponents will be described here. The phase of the compiler that deals with the ICode is split into 3 subphases with each of them having its own specialized version of the ICode. While a general schema can be used to check the correctness of all three ICode versions special schemes will be used to check inconsistencies between the three different versions.

3.1 XML Representation of the ICode

- What will be represented in the intermediate language? Why?
- What will not be represented? Why?
- How will things be represented? Why? (attribute vs. content; IDRef vs. element of; ...)

3.2 The schemes

- The general scheme
- Schemes for multi phase optimization

3.3 Concepts of the Interface

- Concepts of the import
- Concepts of the export

4 Implementation

This chapter contains the details of the implementation and therefore a description of the structure of the software. The implementation and integration of the two parts of the software will be explained.

4.1 Structure

Here the class structure of the software will be described. Therefore I will explain the concept behind each class and which functions it should perform. A general overview will show where and how the Interface will be inserted in the existing compiler.

4.2 Import

Before the import of the XML file is done it will be checked against a schema. This will be done by the Xerces API and here I will describe how. After the XML Input passed the test it will be imported via a Sax event parser. I will explain certain issues that may occur with some of the imported ICode elements and how they will be solved. The last topic of this section will be the integration into the existing compiler. The generation of the ICode elements will be done by the Factory methods which will be explained here too.

4.3 Export

To iterate over the treelike structure of the ICode two common design patterns will be used: the Iterator patter and the Visitor patter. I will describe their implementation in this chapter. Another topic will be how the XML Output will be realized and how the Export will be integrated into the existing compiler.

5 Results

The first result of this work will be to show that the integration of an XML based intermediate language is possible. Further on measurements of the size of the generated XML documents and of the performance of the import and export operations will be of importance. Since the purpose of this work is to generate XML for testing purposes some of the results of this testing will be described here.

5.1 Performance of the implementation

- Feasibility
- Size of the output
- Performance of the import and export operations

5.2 Results drawn from the output

6 Outlook

Since the implementation of the work does not handle the whole ICode I will describe the elements that are missing and how they should be converted into XML and back. Further I will give a short overview over other phases of the compiler and explain where other Interfaces could be inserted.

6.1 Implementation for the whole ICode

- Which elements are missing up to now
- Additional difficulties with the import or export of these elements

6.2 XML Interfaces for other phases

- Rough structure of the compiler
- Where could the additional interfaces be inserted

7 Timeline

GANTT project
Dezember 2005
Januar 2006
Februar 2006
März 2006
Woche 49
Woche 50
Woche 51
Woche 52
Woche 1
Woche 2
Woche 3
Woche 4
Woche 5
Woche 6
Woche 7
Woche 8
Woche 9
Woche 10
Document
Write Introduction
Write Basics
Write Concepts
Write Implementation
Write Results
Write Outlook
Implementation