Extending CZT parsers

Leonardo Freitas

March 2006

Abstract

This article documents the process of extending the CZT parsing architecture for a new Z extension. In particular, it uses *Circus* as a target language example.

1 Introduction

In this document we define the general guidelines for extending the CZT parsing architecture. It includes various parsers, scanners, and pretty printers. The architecture allows one to use Apache ANT to generate the Java, CUP, and JFlex source files related to theses tools from various XML schema files containing the appropriate markup for each different language. Thus, we allow some reuse within the rather usually monolithic structure of parsing tools.

The main tools generated are usually related to LaTeX and Unicode. At the moment we have three extensions for Z, Object-Z and TCOZ. New languages can be added as needed. For instance, in this document we use *Circus* as the new language we are extending the CZT parsing architecture for. Furthermore, we assume the paths given are relative to the value of the `CZT_HOME` environment variable.

Although the modifications for different extensions spread across different files and directories, it is left to an irreducible minimum in order to allow extension with least effort possible. This is because parsers and scanner code are very difficult to extend through inheritance due to the very nature of its algorithm. Thus, our compromise here for reusability is using XML templates as an intermediate medium between the various versions. That is, from the XML templates, one is able to generate a set of different parsers and scanners, hence leaving room for better maintenance and extensibility.

1.1 Dependencies

The parsing architecture depends on `corejava.jar`, and `session.jar`. Therefore, to extend a parser one should be aware of the dependencies and documentation of those two packages. Moreover, the JWSP files might be necessary if one want to read/write XML files using JAXB.

During the execution of a parsing session, some warning or information messages about `clover.jar` or typechecking might appear. These can be ignored and have to do with CZT testing architecture, and session management typechecking configuration respectively.

2 Defining Unicode Characters

+ devtools/circus2charclass.xsl!

+ talk about the 13 layered scanners!

+ talk about ContextFreeScanner + Latex2Unicode!

3 Defining keywords

To instruct the parsers to recognise keywords of a language as special tokens, one need to edit the following files:

1. `./parser/lib/circus_toolkit.tex`

   The `parser/lib` directory contains the toolkits for the various languages. Even if the target language do not have particular definitions, the keywords must appear somewhere for the parser/scanners to be aware of it.

   It does not need to be in the toolkit file necessarily, neither on this directory. Nevertheless, if one puts it somewhere else, the `czt.path` environment variable must be to the appropriate directory where the file containing the keywords lie. It can be either a Unicode (`.utf8`, or `.utf16`), or a LaTeX file.

2. `./devtools/circuschar.xml`
   `./corejava/xsl/circuschar2stringclass.xsl`

   These files are the link between Unicode characters and strings into Java. The char file contains Unicode characters, whereas the string file contains Unicode strings. Usually, keywords are to be found here.

3. `./parser/templates/KeywordScanner.xml` This XML template is the place to define the keywords for the various scanners, as well as to link them to the underlying parsers.

4. `./parser/templates/Parser.xml` This XML template is where the language grammar rules are defined using the CUP parser generator.

   The Java variable name of the string constant defined in the file ***must*** be the same as the CUP terminal name used for this special token. This establishes a programmable link between the CUP parsing and other Java files and scanners.

For example, to define the **channel** keyword for declaration of channels in *Circus* one needs to update these files as follows:

1. `./parser/lib/circus_toolkit.tex`

   ```
   %%Zpreword \circchannel channel
   ```

2. `./corejava/xsl/circuschar2stringclass.xsl`

   ```
   String CIRCCHAN = ‘‘channel’’
   ```

3. `./parser/templates/KeywordScanner.xml`

   ```
   <add:circus>
   import net.sourceforge.czt.circus.util.CircusString;
   </add:circus>
   ⋮
   public class KeywordScanner implements Scanner {
   ⋮
     private Map<String, Integer> createKeywordMap() {
       Map<String, Integer> result = new HashMap<String, Integer>();
       ⋮
       <add:circus>
   ```

```
        //Maps the Java Unicode string with the Parser symbols
        result.put(CircusString.CIRCCHAN, new Integer(Sym.CIRCCHAN));
        </add:circus>
        ⋮
      }
    }
```

4. `./parser/templates/Parser.xml`

```
 //Line 1055 (v1.214) scan with {: return local_next_token(); :};
<add:circus>
  terminal CIRCCHAN;
 </add:circus>
```

Firstly, in `circus_toolkit.tex` we define the mapping between LaTeX and Unicode representation. As in this example, they do not need to be the same. Moreover, we also define which markup directive is to be used for this keyword, which in this case is a **pre**fix key**word** (*i.e.*, `%%Zpreword`). Next, in `circuschar2stringclass.xsl` we define Unicode representation for the `CIRCCHAN` terminal token for the CUP file generated from `Parser.xml`. Finally, in `KeywordScanner.xml` we define the last link between Unicode to LaTeX for the `\circchannel` LaTeX command as just the `‘‘channel’’` Unicode string.

4 Setting a debugging environment

5 Conclusion

+ Simplifies the designs and extension.
  + Demands a series of well-documented steps.