Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BSC-2009-06

Bachelorarbeit

im Studiengang "Angewandte Informatik"

Documentation Generation for TTCN-3

Stefan Kirchner

am Institut für Informatik

Gruppe Softwaretechnik für Verteilte Systeme

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

17. August 2009

Georg-August-Universität Göttingen
Zentrum für Informatik

Goldschmidtstraße 7
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 72000
Fax +49 (5 51) 39-1 44 03
Email office@cs.uni-goettingen.de
WWW www.ifi.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 17. August 2009

Bachelor's Thesis

Documentation Generation for TTCN-3

Stefan Kirchner

August 17, 2009

Supervised by Prof. Dr. Jens Grabowski
Software Engineering for Distributed Systems Group
Institute for Computer Science
Georg-August University of Göttingen, Germany

Abstract

Software documentation is a very important part of software engineering. It is written text that helps to understand the documented software. This thesis presents an approach for automatic documentation generation from TTCN-3 source code using a standardized documentation comment specification. XSL Transformations to the output formats HTML and PDF that use T3DocML — an intermediate XML format that abstracts from TTCN-3 for documentation purposes which was developed in the course of this thesis — are described. A prototypical implementation of the proposed approach, that was successfully tested on several TTCN-3 test suites by ETSI, is presented in this thesis.

Zusammenfassung

Softwaredokumentation ist ein wichtiger Bestandteil der Software-Technik. Sie ist geschriebener Text, der dabei hilft, die dokumentierte Software zu verstehen. Diese Arbeit präsentiert einen Ansatz zur automatischen Dokumentationsgenerierung aus TTCN-3 Source Code mit Hilfe einer standardisierten Spezifikation für Dokumentationskommentare. XSL-Transformationen in die Ausgabeformate HTML und PDF unter Benutzung von T3DocML — einem XML-Zwischenformat, dessen Struktur von TTCN-3 abstrahiert und im Verlaufe dieser Arbeit entwickelt wurde — werden beschrieben. Eine prototypische Implementierung des vorgestellten Ansatzes, welche erfolgreich an diversen TTCN-3 Testsuiten von ETSI getestet wurde, wird in dieser Arbeit präsentiert.

Keywords: TTCN-3, T3Doc, T3DocML, Documentation, XSLT, XSL-FO

Contents

1 Introduction

Software testing is a very important part of software engineering. It provides information about the quality of the *system under test* (SUT). Software testing is conducted to find software bugs or to verify if the SUT meets predefined requirements. The *Testing and Test Control Notation Version 3* (TTCN-3) is one of several testing languages. It is internationally standardized and designed for defining test specifications. As software gets more and more complex, so do test suites. TTCN-3's wide acceptance and applicability has added to the development of very large and complex TTCN-3 test suites, which must be maintained and may need to be developed further. When developing or maintaining complex test suites, it is beneficial if they are well documented as software documentation also is an important part of software engineering. A documentation is written text that explains how to use or how to operate the documented software. Generating those documentations automatically from source code saves time and work. It also makes it faster to keep documentations up-to-date, because changes in the source code only require to run the documentation tool again.

This thesis is about an approach for the automatic generation of technical documentation from TTCN-3 source code. Technical documentations are reference manuals used in the software development process for maintenance- and modification purposes as they describe the architecture of the software. They allow developers, testers and customers to quickly look up any class or method.

Following this introduction, Chapter 2 presents the foundations of the technologies used in this thesis. These technologies are TTCN-3, T3Doc, XML, XML Schema, XSLT, and XSL-FO. In Chapter 3, the requirements for the output formats HTML and PDF of the documentation are presented. Then, the approach of this thesis is described and T3DocML is introduced. It is an XML format that is used to store the information about TTCN-3 modules needed for the documentation. At last, transformations from T3DocML to the output formats are described using XSL stylesheets. Chapter 4 is about building T3DocML from TTCN-3 source code and the implementation of the aforementioned transformations in Java using Saxon and Apache FOP. Finally, the conclusion of this thesis is presented in Chapter 5.

2 Foundations

In this chapter, the foundations of the technologies used in this thesis are presented. Section 2.1 explains TTCN-3 [1]. The following section presents the foundations of T3Doc, which provides additional information for the documentation specification within TTCN-3. In Section 2.3, XML [2], XML Schema [3], and the XML-based technologies XSLT [4] and XSL-FO [5] are described.

2.1 Testing and Test Control Notation Version 3 (TTCN-3)

The *Testing and Test Control Notation Version 3* (TTCN-3) [1, 6] was developed and published by the *European Telecommunications Standards Institute* (ETSI). It is an internationally standardized programming language designed for defining test specifications. Compared to its predecessor, the *Tree and Tabular Combined Notation Version 2* (TTCN-2) [7], it looks much more like a regular programming language. While TTCN-2 was mostly used for testing communication systems, TTCN-3 can be used for a wide range of systems.

TTCN-3 code is organized in TTCN-3 modules. A module can contain definitions of `groups`, `functions`, `testcases`, `altsteps`, `signatures`, `module parameters`, `templates`, `type definitions`, and `constants`. These definitions can be imported by other modules. Modules can also contain an optional `control` part that contains information on how to execute a test suite.

Groups are syntactically similar to modules and are used to structure TTCN-3 definitions. Groups can be imported from other modules, which allows for more selective importing. Other than that, they have little semantical significance.

Functions in TTCN-3 are used to store test code fragments that are, for example, often executed, in a reusable form. They work similar to functions in other programming languages like Java and can be called by testcases, other functions, and the control part of a module.

Testcases are descriptions of how a SUT is to be stimulated and what reactions are expected. Based on those reactions a `verdict` (for example `pass` or `fail`) can be assigned.

Altsteps are named alternatives of `alt` statements that can be referred to in order to avoid code duplication. They are essentially functions for alt statements. Alt statements contain alternatives that are executed based on responses of the SUT. For example, if there is no response after a certain amount of time, the verdict of a testcase is set to `fail`, and if there is an expected response, the verdict is set to `pass`.

Signatures represent methods of the SUT that can be remotely called by a TTCN-3 test suite. They can have parameters, a return value and specify exceptions. Signatures are used to test procedure-based systems (for example, web services).

Module parameters are parameters given to a module to allow for a test suite to work in different environments — for example, by defining the address of a SUT as shown in Listing 2.1. They can be used to provide external parameters at execution time.

```
0  modulepar {charstring sut_ipAddress}
```

Listing 2.1: TTCN-3 Example: Module Parameter

Templates are used for defining messages that are sent and received from the SUT and for matching incoming values against their definition. They define values of a specific type and can use wild cards. The template in Listing 2.2 defines the `Bit` values of the `Byte` type. Templates are used to test message-based, asynchronous systems.

```
0  template Byte seven := {
1          bit1 := 0,
2          // ...
3          bit6 := 1,
4          bit7 := 1,
5          bit8 := 1
6  }
```

Listing 2.2: TTCN-3 Example: Template

Types are defined by restrictions, sets, records, unions, or enumerations of built-in data types (for example, `integer` or `charstring`) or other user-defined types. Listing 2.3 shows an example of using the built-in data type `integer` to define a new type (`Bit`) that is used in the definition of the `Byte` type — a record of `Bits`.

```
0  type integer Bit (0 .. 1);
1
2  type record Byte {
3          Bit bit1,
4          Bit bit2,
5          // ...
6          Bit bit8
7  }
```

Listing 2.3: TTCN-3 Example: Types

Constants are named values. Once defined, the value cannot be changed. Naming values can help to make TTCN-3 code easier to understand. Constants also provide a single point of change if values need to be adjusted.

2.2 TTCN-3 Documentation Comment Specification

The TTCN-3 Documentation Comment Specification [8] (also known and subsequently referred to as T3Doc) specifies special documentation comments to be used in TTCN-3 source code. These comments can be automatically processed to create documentation for the code. They are organized in documentation blocks and contain information about the TTCN-3 element, module, or group they precede, for example the version of a testcase. A T3Doc comment is a valid TTCN-3 comment that starts with the "*"-character. Tagged paragraphs are used to make it easier for documentation tools to identify what kind of information is conveyed. The tagged paragraph given in Listing 2.4 contains information about the author of the following TTCN-3 code by using the `@author` tag.

```
0 // * @author John Doe
```

Listing 2.4: T3Doc Example: Tagged Paragraph

Listing 2.5 shows an example of a T3Doc documentation block using block comments and tagged paragraphs. It contains information about the author(s) (`@author` tag), version (`@version` tag), and a short description (`@desc` tag) of the `Example()` testcase. Tagged paragraphs can extend over several lines as they end with the next tagged paragraph or with the end of the documentation block. A list of tags that can be used in T3Doc comments is given in Table 2.1, along with information on their usage, taken from the T3Doc standard [8]. With the help of these tags, additional information about the TTCN-3 code can be specified.

```
0  // ...
1
2  /*
3   * @author John Doe
4   * @version v0.1.4
5   * @desc This is an example of a testcase in ttcn-3
6   * this is line 2 of the description paragraph
7   */
8
9  testcase Example() runs on Examplecomponent{
10                 // TTCN-3 testcase code
11 }
12 // ...
```

Listing 2.5: T3Doc Example: A Documentation Block

Tag	Specifies	Occurrence (max., location)
@author	Name(s) of the author(s)	Any, preceding any element
@config	Test configuration	Once, preceding testcases
@desc	Intention and functionality	Any, preceding any element
@exception	Information on exception types	Any, preceding signatures
@member	Name(s) of member(s)	Any, preceding certain type-, modulepar- and template definitions
@param	Parameters	Any, preceding parameterized elements
@purpose	Test purposes	Once, preceding testcases
@remark	Extra information	Any, preceding any element
@return	Information on returned values	Once, preceding signatures and functions
@see	Reference to other elements	Any, preceding any element or embedded in any T3Doc tag
@since	Module version when added	Once, preceding any element
@status	Status	Once, preceding any element
@url	Reference to external sources	Any, preceding any element or embedded in any T3Doc tag
@verdict	Circumstances of verdicts	Any, preceding testcases, functions, and altsteps
@version	Version	Once, preceding any element

Table 2.1: The T3Doc Tags

2.3 Extensible Markup Language (XML)

The *Extensible Markup Language* (XML) [2] is an open standard designed to store and carry semistructured data and was recommended by the *World Wide Web Consortium* (W3C). It is a markup language without predefined tags that derived from the *Standard Generalized Markup Language* (SGML) [9]. The XML text representation stores data as plain text, making it hard- and software independent. This allows for easy exchange of data between heterogeneous systems. By predefining the tags, attributes, and structure (the schema) of an XML document, new languages like XHTML [10], RSS [11], RDF [12], and OWL [13] can be created.

Listing 2.6 shows how data of a mailing address consisting of country, city, postal code, street, and street number could be stored in XML textual representation.

```
0 <!-- [...] -->
1 <address>
2  <street number="7">Goldschmidtstrasse</street>
3  <city zip="37077">Goettingen</city>
4  <country>Germany</country>
5 </address>
6 <!-- [...] -->
```

Listing 2.6: XML Example: Address

A well-formed XML document consists of an XML declaration, one or more elements, and possibly comments, processing instructions and white spaces. An XML declaration specifies the XML version that is used in the document. Elements are either empty — consisting of an empty-element tag and possibly attributes (for example `<anemptyelement/>`) — or not empty — consisting of a start tag, a end tag, and possibly attributes and content. An example for a non-empty element would be `<city zip="37077">Goettingen</city>` as given in Listing 2.6, where `zip="37077"` is an attribute and the character string "Goettingen" is the content of the `<city>` element. Non-empty elements can also contain other elements.

An example of a well-formed XML document is given in Listing 2.7. The `<contacts>` element is called root element which contains all other elements of the XML document. For an XML document to be not only well-formed but also valid, it needs an associated *Document Type Definition* (DTD) [2], which is part of the XML standard, or an XML schema which also can be used for validation. Listing 2.8 shows a DTD defining the structure of a `contacts` XML document such as the one given in Listing 2.7.

```
0  <?xml version="1.0" ?>
1  <contacts>
2   <contact>
3    <name>John Doe</name>
4    <address>
5     <street number="7">Goldschmidtstrasse</street>
6     <city zip="37077">Goettingen</city>
7     <country>Germany</country>
8    </address>
9   </contact>
10  <contact>
11   <name>Jane Doe</name>
12   <address>
13    <street number="3">Bunsenstrasse</street>
14    <city zip="37077">Goettingen</city>
15    <country>Germany</country>
16   </address>
17  </contact>
18 </contacts>
```

Listing 2.7: XML Example: Contacts

```
0  <!ELEMENT contacts (contact*)>
1  <!ELEMENT contact (name, address)>
2  <!ELEMENT name (#PCDATA)>
3  <!ELEMENT address (street, city, country)>
4  <!ELEMENT street (#PCDATA)>
5   <!ATTLIST street number CDATA #REQUIRED>
6  <!ELEMENT city (#PCDATA)>
7   <!ATTLIST city zip CDATA #REQUIRED>
8  <!ELEMENT country (#PCDATA)>
```

Listing 2.8: DTD Example: Contacts

An XML element is declared via an Element type declaration, which defines its name and content as `<!ELEMENT name (content)>`. The content is either a sequence of other elements (including `#PCDATA` for text), `EMPTY` for no content, or `ANY` for any content. In this sequence, elements can either be in an ordered list (`element1, element2, element1`) or in a list of alternatives where only one element is chosen from (`element1 | element2 | element3`). A sequence can be treated as an element by using brackets — for example, `element1, (element2 | element3), element4`. By adding quantifiers like in regular expressions behind elements, the number of occurrences can be specified. In Line 0, the `<contacts>` element is defined as a sequence of any number of `<contact>` elements, which are defined in Line 1 as consisting of exactly one `<name>`- and one `<address>` element. The attributes of elements are defined with `<!ATTLIST elementname list_of_attributes>` where an attribute consists of a name, a type and a default value as shown in Lines 5 and 7.

2.3.1 XML Schema Definition

The *XML Schema Definition* (XSD) [3] is an XML schema language recommended by the W3C. It is also known as XML Schema. Like a DTD, an XML schema defines the structure of an XML document, but has several advantages. For example, it supports data types, which makes it possible to check if the data is correct. XML Schema comes with several primitive datatypes (e.g., `integer` or `boolean`), with which new data types can be created by using restrictions, lists, and unions of them. Also, XML schemas themselves are written in XML, therefore they have all the advantages of XML text representations.

```
0  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
1   <xs:element name="address">
2    <xs:complexType>
3     <xs:sequence>
4      <xs:element ref="street" />
5      <xs:element ref="city" />
6      <xs:element name="country" type="xs:string" />
7     </xs:sequence>
8    </xs:complexType>
9   </xs:element>
10  <xs:element name="city">
11   <xs:complexType mixed="true">
12    <xs:attribute name="zip" type="xs:string" use="required" />
13   </xs:complexType>
14  </xs:element>
15  <xs:element name="contact">
16   <xs:complexType>
17    <xs:sequence>
18     <xs:element name="name" type="xs:string" />
19     <xs:element ref="address" />
20    </xs:sequence>
21   </xs:complexType>
22  </xs:element>
23  <xs:element name="contacts">
24   <xs:complexType>
25    <xs:sequence>
26     <xs:element ref="contact" minOccurs="0" maxOccurs="unbounded" />
27    </xs:sequence>
28   </xs:complexType>
29  </xs:element>
30  <xs:element name="street">
31   <xs:complexType mixed="true">
32    <xs:attribute name="number" type="xs:string" use="required" />
33   </xs:complexType>
34  </xs:element>
35 </xs:schema>
```

Listing 2.9: XSD Example: Contacts

Listing 2.9 shows an XML Schema equivalent of the DTD in Listing 2.8. Elements that can contain other elements or have attributes are defined as complex types such as the `<address>` element or the `<city>` element. The `<address>` element is defined in Lines 2-10, consisting of a sequence of three other elements (`<street>`, `<city>`, and `<country>`). In Line 14 the `zip` attribute of the `<city>` element is defined as a character string — a primitive data type of XML Schema.

2.3.2 Extensible Stylesheet Language Transformations (XSLT)

Extensible Stylesheet Language Transformations (XSLT) [4, 5] is a language recommended by the W3C in 1999. It is used to transform a source XML document into a result XML document — for example, into an XHTML document, to display the data in a web browser — or a different text-based format. The source document is not altered. During transformation, patterns are matched against elements of the source document and templates are used to create the result document with those elements.

By using XSLT, the data is separated from its presentation. This has several advantages, for example, multiple output formats: using the same source document, several presentations of the same data can be produced by using different XSL [5] stylesheets. On the other hand, the same stylesheet can be used for different source documents of the same structure.

```
0  <xsl:stylesheet version="2.0" <!-- [...] --> >
1  <xsl:output method="xml" <!-- [...] --> />
2   <xsl:template match="/">
3    <html>
4     <head> <title>Contacts</title> </head>
5     <body>
6      <table border="1">
7       <tr>
8        <th>Name</th>
9        <th>Country</th>
10      </tr>
11      <xsl:apply-templates select="//contact"/>
12     </table>
13    </body>
14   </html>
15  </xsl:template>
16  <xsl:template match="contact">
17   <tr>
18    <td><xsl:value-of select="name/text()"/></td>
19    <td><xsl:value-of select="address/country/text()"/></td>
20   </tr>
21  </xsl:template>
22 </xsl:stylesheet>
```

Listing 2.10: XSLT Example: Address

To illustrate how XSLT works, an example of an XSL stylesheet is given in Listing 2.10. Used on the example in Listing 2.7, it would produce the HTML [14] document shown in Listing 2.11. First, a template matching the root element (Lines 2-15) is called. The elements are matched via XPath [15], which is a navigation language for XML. The contents of a called template define the output written to the result document. In Line 11, another template that matches all `<contact>` elements of the source document is called. It is called separately for each of those elements. This template is defined in Lines 16-21 and creates a `<tr>` element containing two `<td>` elements each time it is called. The first `<td>` element contains the contents of the `<name>` element inside the `<contact>` element the template was called with. The second contains the contents of the `<country>` element inside the `<address>` element. This results in an HTML table with the names and country of a contact.

```
0  <html>
1   <head>
2    <title>Contacts</title>
3   </head>
4   <body>
5    <table border="1">
6     <tr>
7      <th>Name</th>
8      <th>Country</th>
9     </tr>
10    <tr>
11     <td>John Doe</td>
12     <td>Germany</td>
13    </tr>
14    <tr>
15     <td>Jane Doe</td>
16     <td>Germany</td>
17    </tr>
18   </table>
19  </body>
20 </html>
```

Listing 2.11: Result Document

Templates can also be named by giving them a `name` attribute and can be called with parameters using the `<with-param>` element. XSLT also has control flow operators like `if` constructs (`<xsl:if>`) or iterators (`<xsl:for-each>`).

2.3.3 Extensible Stylesheet Language Formatting Objects (XSL-FO)

The *Extensible Stylesheet Language Formatting Objects* (XSL-FO) [5] is a language recommended by the W3C for describing the formatting of XML data. XSL-FO documents are XML documents following a schema that contains information regarding the layout of their data. This information (also XML data) can be used by XSL-FO processors to generate readable/printable output formats such as the *Portable Document Format* (PDF) [16].

The root of an XSL-FO document is the `<fo:root>` element. Inside the root element there is one `<fo:layout-master-set>` element and at least one `<fo:page-sequence>` element (or `<fo:page-sequence-wrapper>`). The `<fo:layout-master-set>` element holds a list of templates defining the size and shape of pages, while the `<fo:page-sequence>` elements contain the actual content of those pages.

To illustrate the structure of an XSL-FO document, an example is given in Listing 2.12. In Lines 7-15, a template with the unique name "A4" is shown. That name can be used for referencing. It defines the page size and margins via attributes. Inside, the regions of the page are defined. The footer and header of the page are represented by `region-before` and `region-after`, while `region-start` and `region-end` are sidebars. They are all part of the `region-body` — which is where the output is written to.

```
0  <?xml version="1.0" encoding="UTF-8"?>
1  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
2
3   <fo:layout-master-set>
4    <fo:simple-page-master master-name="A5">
5     <!-- content -->
6    </fo:simple-page-master>
7    <fo:simple-page-master master-name="A4" page-width="210mm"
8        page-height="297mm" margin-top="1cm" margin-bottom="1cm"
9        margin-left="1cm" margin-right="1cm">
10    <fo:region-body margin="3cm"/>
11    <fo:region-before extent="2cm"/>
12    <fo:region-after extent="2cm"/>
13    <fo:region-start extent="2cm"/>
14    <fo:region-end extent="2cm"/>
15   </fo:simple-page-master>
16  </fo:layout-master-set>
17
18  <fo:page-sequence master-reference="A4">
19   <fo:flow flow-name="xsl-region-before">
20     <fo:block background-color="red">This is a block.</fo:block>
21   </fo:flow>
22  </fo:page-sequence>
23 </fo:root>
```

Listing 2.12: XSL-FO Example

An `<fo:page-sequence>` element (as defined in Lines 18-22) defines output pages. They refer to a page template — in this case, `<fo:simple-page-master master-name="A4">` — and hold a `<fo:flow>` element that contains the actual content of the page. The `flow-name` attribute decides to what part of the body region the output is written to — in this case, to the header. The output in this example is a red block (Line 20) with "This is a block." written on it. An XSL-FO processor would process this document into a single A4 page with a red header saying "This is a block."

Since XSL-FO documents are XML documents, XSLT can be used to create them from a source XML file. This makes it possible to use an XSLT document to create the same XSL-FO output for many XML files of the same structure.

3 Documentation Transformations

In this chapter, the documentation transformations are explained. In the first section, the output requirements for the HTML and PDF formats are described. In the second section, the approach of these transformations is described. The following section is about the XML Schema that is used to store information about TTCN-3 modules. It defines the structure of the source XML documents that are used in the creation of the documentation and will be called T3DocML in the remainder of this thesis. The last two sections of this chapter describe the transformations of T3DocML to HTML and XSL-FO with XSL stylesheets.

3.1 Requirements

The aim is to create a technical documentation in HTML and PDF formats, which contains all information available in the TTCN-3 modules of a project in an easily accessible and clearly arranged format. It shall feature information processed from T3Doc documentation blocks.

3.1.1 HTML Output Requirements

The HTML documentation should have

- A navigation menu,
- Information from the documentation blocks,
- Information on the elements (source code),
- Cross-references to other elements.

The HTML pages shall be divided into two parts: the navigation menu and the content. The navigation menu shall also have two parts. As seen in Figure 3.1, one part of the menu shall be a list of hyperlinks to all HTML files documenting the modules of the TTCN-3 project. The `Modules` hyperlink shall refer to the index page. The other part shall be a list of all element types, including groups. When clicked on, a list of all elements of that type should appear (for example, via Javascript [17]). These two menus should appear on every page of the HTML documentation to allow for easy navigation. The contents of the HTML pages shall contain the information extracted from the documentation blocks in the

source code and the source code of the elements itself. This source code should contain cross-references (hyperlinks) to documentation pages of other TTCN-3 elements referenced in it. This way, the documentation of all elements involved can be accessed quickly. Also, words marked with embedded `@see` tags shall be cross-references, if a corresponding element is found within the documentation.

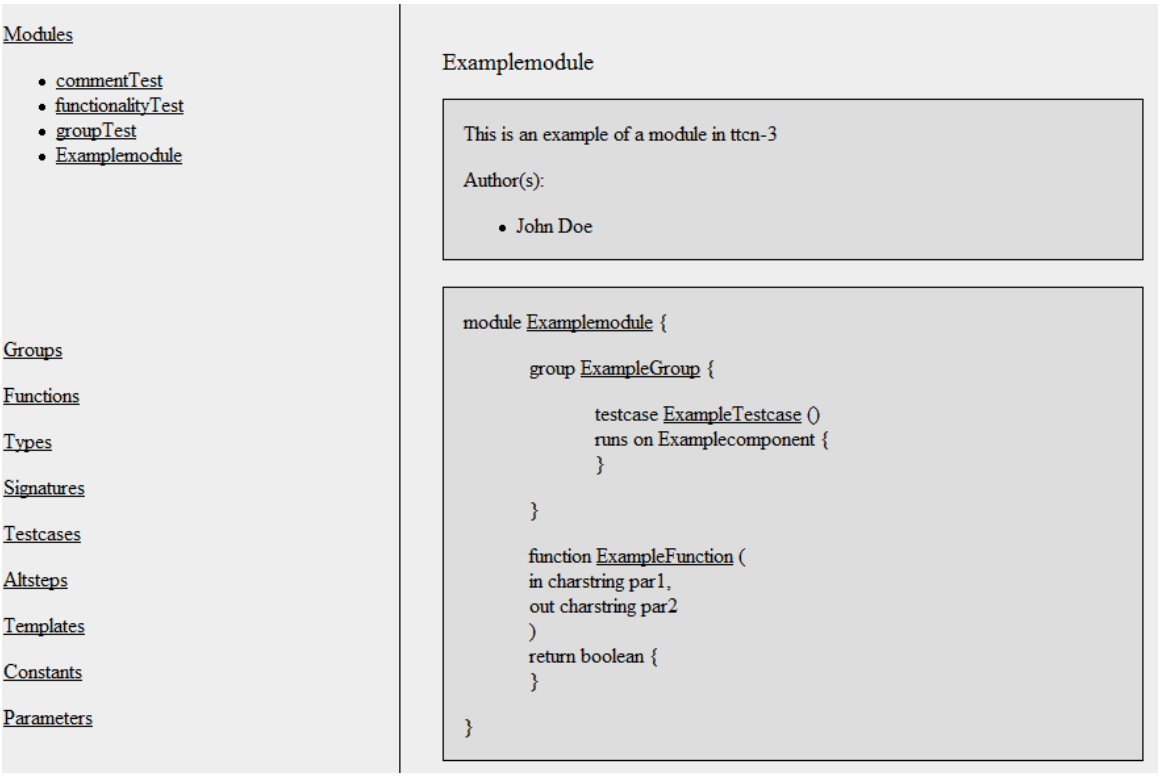

Figure 3.1: A Screenshot of the HTML Documentation

3.1.2 PDF Output Requirements

Every module shall have its own block in the PDF document. Illustrated in Figure 3.2 is the block that the module shown in Listing 3.1 should generate. Every element, including groups, shall have its own block inside the module blocks, with its signature[1] as a headline. The source code of the elements should not be displayed to make the documentation more compact. Information from the T3Doc paragraphs shall be displayed in a way similar to the HTML documentation.

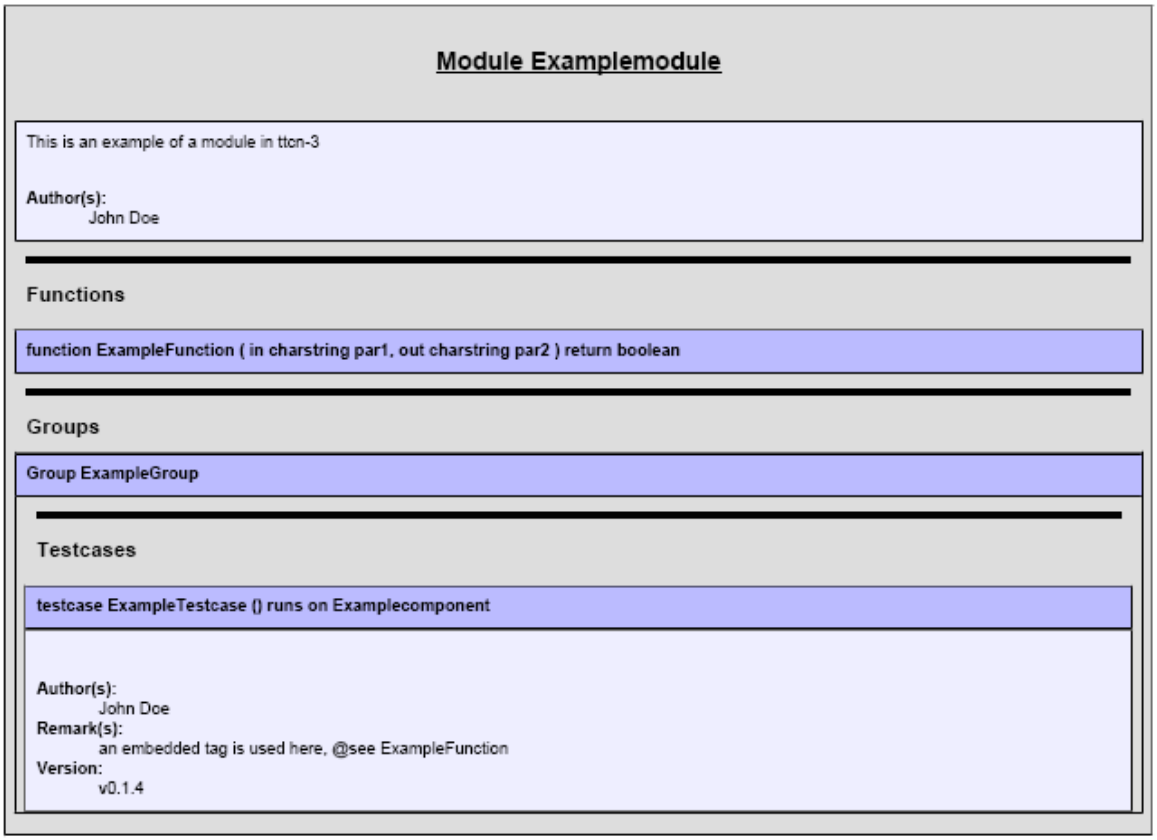

Figure 3.2: A Screenshot of the PDF Documentation

[1] Element signatures are not to be confused with TTCN-3 signatures

3.2 Approach

The goal is to create documentation for TTCN-3 projects in HTML and PDF formats. To achieve this, the necessary data from the TTCN-3 code needed for the documentation will be used to create an XML document in the T3DocML schema. After that, XSLT will be used to transform T3DocML into HTML and an XSL-FO that is used to create the documentation in PDF format. This approach is shown in Figure 3.3.

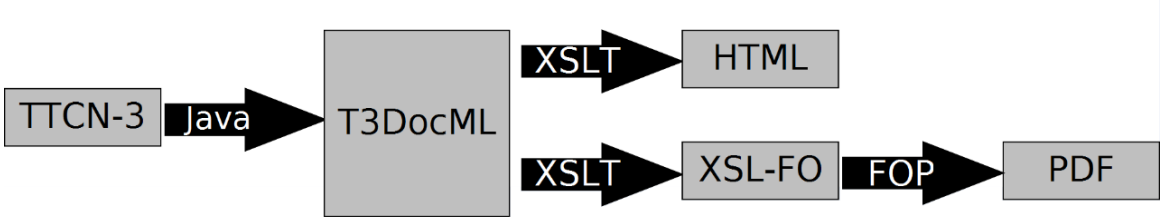

Figure 3.3: Approach

3.3 The T3DocML Schema

First, we need to find an appropriate XML representation of the TTCN-3 source code. It must contain all data that is needed to create the documentation. We call the XML Schema used in this thesis T3Doc Markup Language (T3DocML). As shown in Listing 7.1 and Figure 3.4 in a graphical representation, the structure of T3DocML is abstracted from the structure of TTCN-3, limited to modules, element definitions, and information from T3Doc documentation blocks.

For the HTML documentation, the name, location of the associated HTML file, T3Doc comments, and source code of all TTCN-3 elements are needed. The PDF documentation will need the name, T3Doc comments, and the signatures of these elements. The XML document must also contain information on where the TTCN-3 elements are located inside the TTCN-3 project that is to be documented. Listing 3.1 shows a TTCN-3 module containing a group with a testcase, and a function. It also contains two T3Doc documentation blocks belonging to the module and the testcase. Listing 3.2 shows where the TTCN-3 elements of Listing 3.1 will be located inside the XML code. They are stored inside the `module` element associated with the TTCN-3 module to which they belong. The original group structure is maintained.

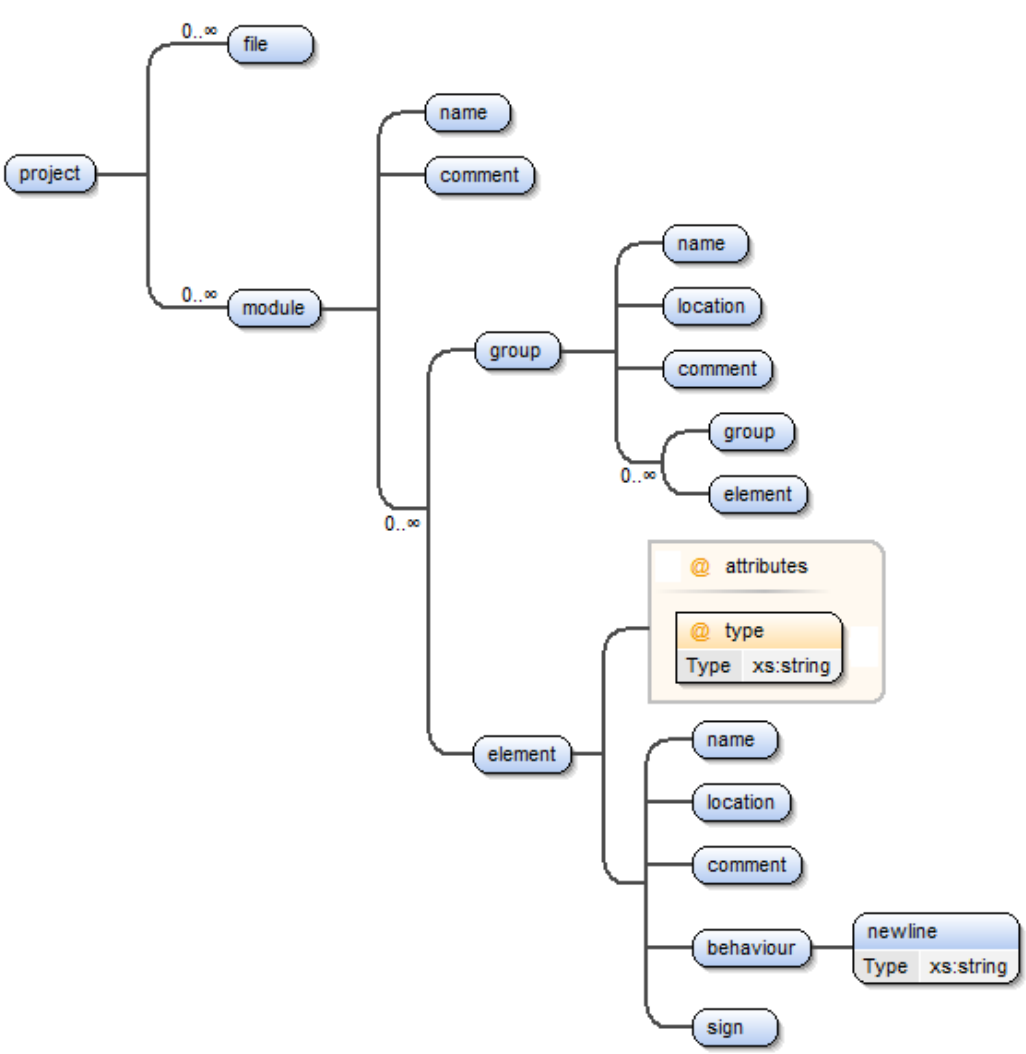

Figure 3.4: Graphical Representation of the T3DocML Schema

```
0  /*
1   * @author John Doe
2   * @desc This is an example of a module in ttcn-3
3   */
4  module Examplemodule {
5         group ExampleGroup {
6                 /*
7                 * @author John Doe
8                 * @version v0.1.4
9                 * @remark an embedded tag is used here, @see ExampleFunction
10                */
11                testcase ExampleTestcase() runs on Examplecomponent{
12                        // TTCN-3 testcase code
13                }
14        }
15        function ExampleFunction (in charstring par1, out charstring par2)
16        return boolean {
17                // TTCN-3 function code
18        }
19 }
```

Listing 3.1: TTCN-3: ExampleModule

```
0  <project>
1   <!-- [...] -->
2   <module>
3    <name>Examplemodule</name>
4    <comment> <!-- contents --> </comment>
5    <group>
6     <name>Examplegroup</name>
7     <element type="testcase"> <!-- contents --> </element>
8    </group>
9    <element type="function"> <!-- contents --> </element>
10  <module>
11 </project>
```

Listing 3.2: T3DocML: ExampleModule

The information from the T3Doc comments of TTCN-3 elements, groups, and modules is stored inside the `<comment>` element, where every T3Doc tag has its own element. For example, the comment `@author John Doe` would be stored as `<author>John Doe</author>`. Listing 3.3 shows the T3DocML representation of the documentation block preceding `ExampleTestcase()` in Listing 3.1. To mark references to other TTCN-3 elements (identified by embedded `@see` tags), `<esee>` elements are used as shown in Line 4.

```
<comment>
 <author>John Doe</author>
 <version>v.0.1.4</version>
 <remark>
  an embedded tag is used here, @see <esee>ExampleFunction</esee>
 </remark>
<comment>
```

Listing 3.3: T3DocML: Comments

The source code is contained inside `<behaviour>` elements, where `<newline/>` elements mark the beginning of a new line and references will be marked (similar to embedded `@see` tags) with `<link>` elements, which contain the location of the referenced elements as attribute (`<link loc="...">...</link>`). The root of the T3DocML document is the `<project>` element. It contains every `<module>` and a list of all the HTML files that are created, to verify references to other elements inside the TTCN-3 source code.

3.4 Transforming T3DocML to HTML

Using XSLT, a transformation from T3DocML to HTML can be described. This section describes the generation of the documentation in HTML format.

3.4.1 Navigation

To navigate between the different element- and module documentation files, a navigation menu is created. The part of the navigation, that contains a list of all modules, is generated by calling the template shown in Listing 3.4.

```
<xsl:template name="index_modulelist">
 <p class="index_headline"><a href="...">Modules</a></p>
 <ul>
  <xsl:for-each select="//module">
   <li>
    <a href="{name/text()}.html"><xsl:copy-of select="name/text()"/></a>
   </li>
  </xsl:for-each>
 </ul>
</xsl:template>
```

Listing 3.4: XSLT: Module Navigation

The lists of the elements are generated by selecting all elements of a certain type and calling the `index_elementlist` template with those elements as parameters, as shown in Listing 3.5.

```
0  <xsl:call-template name="index_elementlist">
1    <xsl:with-param name="this" select="$this//element[@type eq 'function']"/>
2  </xsl:call-template>
```

Listing 3.5: XSLT: Calling index_elementlist

The `index_group` template in Listing 3.6 generates a list of groups. This list can contain other lists as shown in Figure 3.5. This is done by recursion. In Lines 6-8, the template calls itself with all `<group>` elements of its current element as parameters and creates an unordered list with them.

```
0  <xsl:template name="index_group">
1   <xsl:param name="this"/>
2   <xsl:for-each select="$this">
3   <li><a href="{location/text()}"><xsl:copy-of select="name/text()"/></a>
4   <xsl:if test="./group">
5    <ul>
6     <xsl:call-template name="index_group">
7      <xsl:with-param name="this" select="./group"/>
8     </xsl:call-template>
9    </ul>
10   </xsl:if>
11  </li>
12  </xsl:for-each>
13 </xsl:template>
```

Listing 3.6: XSLT: Group Navigation

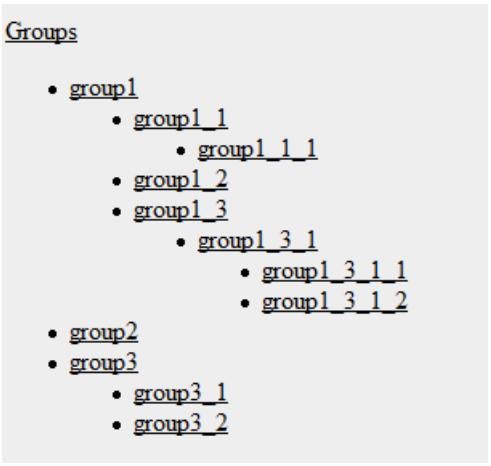

Figure 3.5: Group Structure of the HTML Navigation

3.4.2 Source Code and Comments

The source code of the TTCN-3 elements is transformed into a `<p>` element and `<newline/>` tags are replaced by `<br/>` tags to mark the end of a line. Similarly, the tagged T3Doc paragraphs are transformed to list-items (`<li>`) where every tag gets its own list.

3.4.3 References

The references in the source code, which are marked with `<link>` elements inside a `<behaviour>` element, are turned into cross-references by calling the `link` template with them. This template searches all `<file>` elements of the root element for one that matches the `loc` attribute of the `<link>` elements. If one is found, a hyperlink referring to the element is generated.

```
0  <xsl:template match="see">
1   <xsl:variable name="text" select="normalize-space(text())" />
2   <xsl:choose>
3    <xsl:when test="contains($text, '.')">
4     <xsl:variable name="file" select="/project/file[contains(text(),
5      concat('_e_', substring-after($text, '.'), '.html')) and
6      contains(text(), concat('_m_', substring-before($text, '.'), '_o_'))]"/>
7     <xsl:if test="$file">
8      <li class="comment_listitem">
9       <a href="{$file/text()}"><xsl:copy-of select="$text"/></a>
10     </li>
11    </xsl:if>
12    <xsl:if test="not($file)">
13     <li class="comment_listitem"><xsl:copy-of select="text()"/></li>
14    </xsl:if>
15   </xsl:when>
16   <xsl:otherwise>
17    <xsl:variable name="file" select="/project/file[contains(text(),
18     concat('_e_', $text, '.html'))]"/>
19    <xsl:if test="$file">
20     <li class="comment_listitem">
21      <a href="{$file/text()}"><xsl:copy-of select="$text"/></a>
22     </li>
23    </xsl:if>
24    <xsl:if test="not($file)">
25     <li class="comment_listitem"><xsl:copy-of select="text()"/></li>
26    </xsl:if>
27   </xsl:otherwise>
28  </xsl:choose>
29 </xsl:template>
```

Listing 3.7: XSLT: The see Template

References using the `@see` tag inside the comments are handled by the `see` template shown in Listing 3.7. These references can either be just the name of an element or the name of a module and the name of an element separated by a dot (for example, `@see ExampleModule.ExampleTestcase`, which refers to the `ExampleTestcase` inside the `ExampleModule`). Similar to references in the source code, `<file>` elements are searched for one that contains these names, to ensure that only existing documentation pages are cross-referenced. When there is a dot between module an element name, Lines 4-6 are executed, where the template searches for an HTML file name that belongs to a module that matches the character string before the dot and represents an element that matches the character string after the dot. If there is one, it is stored inside a XSL variable called `file`. When there is no dot, the template tries to match only the element part (Lines 17-18). After that, if the `file` variable is not empty, a hyperlink referring to the first file inside this variable is generated (Lines 7-11 and 19-23). If it is empty, the text inside the `<see>` tags will be printed normally (Lines 12-14 and 24-26). The same is done for embedded `@see` tags.

3.5 Transforming T3DocML to XSL-FO

Similar to the previous section, this section describes a transformation from T3DocML to XSL-FO using XSLT. First, the template matching the `<module>` elements is called for every module in the project. This template creates an `<fo:block>` containing two other blocks: one for the T3Doc comments about the module and another for the elements and groups contained within the module. The same is done for groups, so every group is within the block belonging to its parent group. The contents of modules and groups are transformed using the `modulecontent` template shown in Listing 3.8. This listing shows only the part of the template that deals with functions.

```
0  <xsl:template name="modulecontent">
1   <xsl:param name="this"/>
2   <xsl:if test="count($this/element[@type eq 'function']) &gt; 0">
3    <fo:block border="3pt solid black" border-bottom-style="hidden"
4     border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
5     <fo:block text-align="left" font-weight="bold" font-size="10pt"
6      margin-top="10pt">
7      <fo:inline>Functions</fo:inline>
8     </fo:block>
9     <xsl:apply-templates select="$this/element[@type eq 'function']"/>
10   </fo:block>
11  </xsl:if>
12 <!-- ... -->
13  </xsl:template>
```

Listing 3.8: XSLT: The modulecontent Template

For every type of element, the template tests whether or not elements of this type exist within the module/group (Line 2). If there are elements of a certain type, this template will create a headline with the name of the type (Line 7) and call the corresponding templates with those elements (Line 9). The template matching elements will simply create an `<fo:block>` for an element, containing its signature as headline and underneath that, information from the documentation blocks.

4 Implementation

This chapter deals with the implementation of the approach described in Chapter 3. The first step is to get the necessary data from the TTCN-3 files. This is done via Java with the help of TRex [18] — an Eclipse plug-in and analysis infrastructure for TTCN-3. In the second step this data will be evaluated and stored in T3DocML as described in Section 3.3. The XSL Transformations are done by the open source Saxon XSLT processor [19]. The resulting XSL-FO document is then processed by Apache FOP [20] — an XSL-FO processor by Apache — into a PDF file. This procedure is depicted in Figure 4.1.

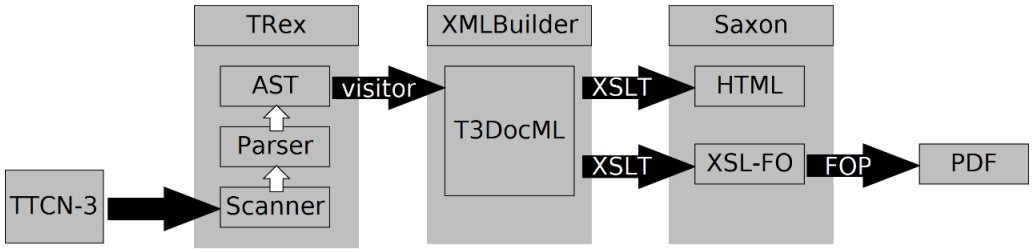

Figure 4.1: Implementation

4.1 Building T3DocML from TTCN-3

The TTCN-3 code is scanned and parsed by TRex. TRex creates an *Abstract Syntax Tree* (AST) of the code. This tree is then traversed by depth-first search using a *visitor* [21] that calls methods when visiting nodes of the tree. Whenever nodes representing new TTCN-3 elements are encountered, the visitor builds instances of the `TTCN3Element` class representing the elements with information on the elements extracted from the AST and the source code comments provided by TRex. The `TTCN3Element` instances are send to the *XMLBuilder* (see Section 4.1.3) component which handles new elements, groups, and modules. An example of an AST is given in Figure 4.2. It shows the module given in Listing 3.1 in the AST view of TRex.

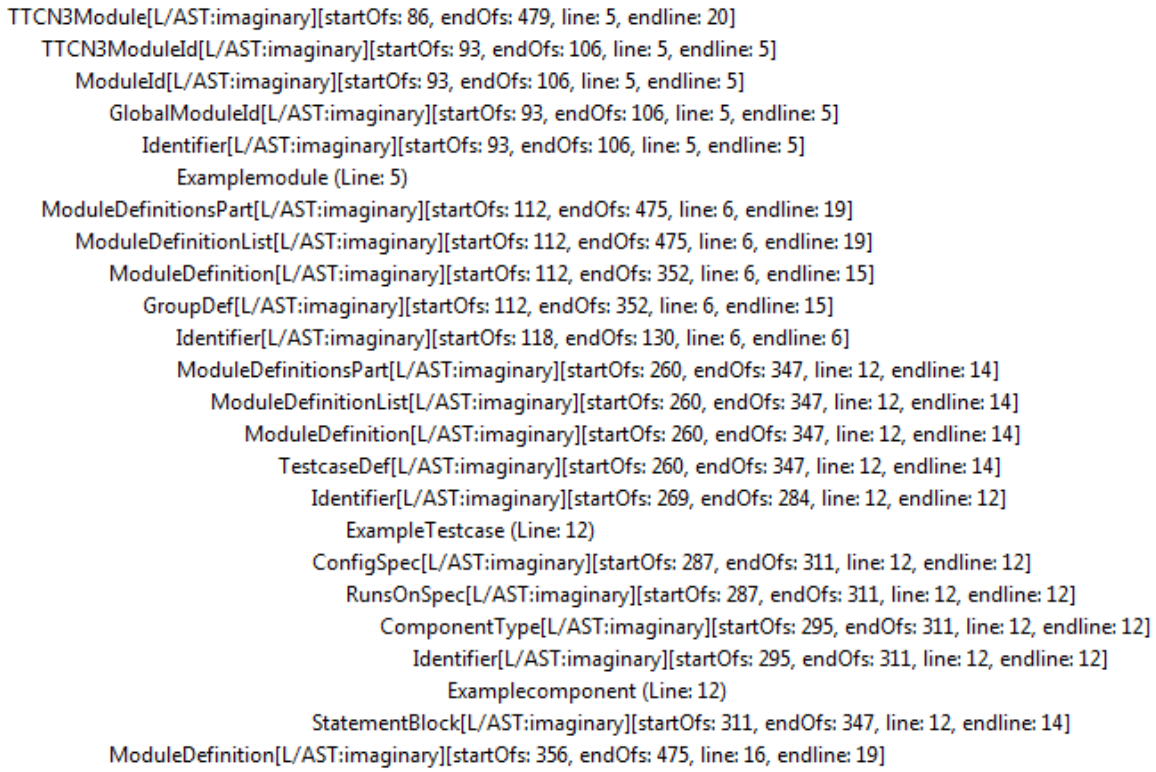

Figure 4.2: TRex AST View: ExampleModule

4.1.1 The Classes Representing TTCN-3 Code

The Java classes representing TTCN-3 code are

- `TTCN3Project`
- `TTCN3Module`
- `moduleElements`
- `TTCN3Element`
- `TTCN3El`
- `TTCN3Group`
- `TTCN3Comment`

A `TTCN3Project` consists of two arrays: one for `TTCN3Modules` and one for `Strings` representing the filenames of all HTML files that will be created by the Saxon XSLT processor.

A `TTCN3Module` consists of one `moduleElements`-, one `TTCN3Comment`-, and one `String` variable representing the name of the module. A `modulelements` instance contains a `Vector` of `TTCN3Elements`.

`TTCN3Element` is an abstract class representing the elements a module can contain, including groups. It has five fields:

- name — the name of the element
- location — the name of the HTML file documenting the element
- behavior — the source code of the element, including references
- sign — the signature of the element, without references
- comment — information from the T3Doc documentation blocks

The `TTCN3El` class extends the `TTCN3Element` class. It represents all TTCN-3 elements, except groups. It has a `type` attribute and a `toString()` method (shown in Listing 4.1) that returns the T3DocML representation of the element.

```
0  public String toString(){
1        return
2                "<element type=\"" + type + "\">\n" +
3                "<name>" + name + "</name>\n" +
4                "<location>" + location + "</location>\n" +
5                comment.toString() +
6                "<behaviour>" + behaviour + "</behaviour>" +
7                "<sign>" + sign + "</sign>" +
8                "</element>\n";
9  }
```

Listing 4.1: The toString() Method

`TTCN3Group` also extends the `TTCN3Element` class. It represents only groups. It contains all elements inside a group (even other groups) in a `moduleElements` instance. A `TTCN3Group` also has a variable representing the location of the group in the TTCN-3 code, so the documentation tool knows which elements belong to it.

The `TTCN3Comment` class contains information about the T3Doc comments of an element. It is instantiated with a variable containing a T3Doc documentation block as an argument and returns the T3DocML representation of this block via `toString()` method.

4.1.2 Visitor

As stated earlier, the visitor will execute certain methods upon encountering certain nodes in the tree that represents the TTCN-3 source code during a depth-first traversal. For example, if it visits a `TestcaseDef` node, it executes the `VisitTestcaseDef()` method. When it encounters the definition of a new Module (`TTCN3Module` node), it passes on the information to the XMLBuilder. Upon visiting other element definitions, the visitor will read and process the necessary data of this element from the syntax tree and use it to create an instance of `TTCN3El` that represents the element. The `TTCN3El` is then transfered to the XMLBuilder. The same is done for `TTCN3Groups`.

Getting the Data From the AST

First, the name of every element is required. To get it, the `identifier` node of the element definition needs to be found. The visitor traverses the AST until it finds this node and returns it. After that, the behavior of the element needs to be read and processed. Listing 4.2 shows the `getBehaviour()` method.

```
0  private String getBehaviour(LocationAST node){
1          String elName = getName(node);
2          String elLocation = getLocation(node);
3          Vector<TTCN3Reference> refvector = getReferences(node);
4          String b = getWholeElement(node);
5          b = StringEscapeUtils.escapeXml(b).replaceAll("\n", " <newline/> ");
6          for(TTCN3Reference r : refvector){
7                  b = b.replaceAll("( |\\t)" + r.name + " ", " <link loc=\"" +
8                          r.location + "\">" + r.name + "</link> ");
9                  b = b.replaceAll("( |\\t)" + r.name + ";", " <link loc=\"" +
10                         r.location + "\">" + r.name + "</link>;");
11                 b = b.replaceAll("( |\\t)" + r.name + ",", " <link loc=\"" +
12                         r.location + "\">" + r.name + "</link>,");
13         }
14         b = b.replaceAll(" " + elName + " ", " <link loc=\"" +
15                 elLocation      + "\">" + elName + "</link> ");
16         return b;
17 }
```

Listing 4.2: The getBehaviour() Method

It is called with the element definition node as argument and returns the behavior of an element with references marked by `<link>` tags, so they can be identified later on. The last step is to extract the documentation blocks from the TTCN-3 source code. This is realized by using the `getCommentsBefore()` method provided by TRex.

4.1.3 XMLBuilder

The `XMLBuilder` class holds all `TTCN3Modules`, `TTCN3Groups`, and `TTCN3Els`. It assigns the elements to the respective modules and groups and returns the T3DocML representation of the whole TTCN-3 project.

Adding Groups

Groups are added via `newGroup()` method by the visitor. This method is shown in Listing 4.3. It holds an array of `TTCN3Groups` which is used to maintain the original group structure. When all elements of a group are already parsed, the group is added to its parent group or directly to the current module.

```
0  public void newGroup(TTCN3Group group){
1          project.addFile(group.getLocation());
2          while(!groups.isEmpty() &&
3                  group.getOffset() > groups.lastElement().getOffset()){
4                  TTCN3Group tempGroup;
5                  tempGroup = groups.lastElement();
6                  groups.remove(groups.lastElement());
7                  if(!groups.isEmpty())
8                          groups.lastElement().add(tempGroup);
9                  else
10                         currentModule.add(tempGroup);
11         }
12         groups.add(group);
13 }
```

Listing 4.3: The newGroup() Method of the XMLBuilder

Adding Elements

Adding elements is similar to adding groups. Every element will either be added to its associated group or the current module.

Creating the T3DocML File

After all modules are added to the project, the T3DocML textual representation is returned by using the `buildXML()` method of the project. It returns the root element (`<project>`) as a `String`.

4.1.4 Processing the Comments

When an instance of `TTCN3Comment` is created, it is given the comments from the TTCN-3 source code and a variable representing an element type. The tagged paragraphs from the comments are matched via regular expressions and stored in arrays. Each T3Doc tag has its own array. The `@remark` paragraph from Listing 3.1 Line 9 would be stored in an array called `remark` as `"an embedded tag is used here, @see ExampleFunction"`.

Marking the References

To place the `<esee>` tags around the references identified by embedded `@see` tags, the `replaceSee()` method is called with elements from the T3Doc tag arrays as argument. This method is shown in Listing 4.4.

```
0  private String replaceSee(String c){
1   int pos;
2   c = c.replace(" @see ", " @SEE ") + ' ';
3   while(c.contains(" @SEE ")){
4     pos = c.indexOf(" @SEE ");
5     if(pos != -1){
6       String identifier = "";
7       for(int i = pos+6; i < c.length() && c.charAt(i) != ' '; i++){
8         identifier += c.charAt(i);
9       }
10      if(identifier.contains("@see"))
11        break;
12      identifier = identifier.replace("@SEE", "").replace(" ", "");
13      c = c.replace("@SEE " + identifier + " ", "@see <esee>"
14      + identifier + "</esee>");
15    }else
16      break;
17  }
18  return c.replace("@SEE", "@see ");
19 }
```

Listing 4.4: The replaceSee() Method

First, it replaces every "@see" with "@SEE" to mark references that have not been replaced yet. It then searches for every "@SEE" in the input string. Upon finding one, the `identifier` behind the `@SEE` tag is determined (Lines 7-9) and then marked with `<esee>` tags (Lines 13-14). After that, "@SEE" is changed back to "@see". When every "@SEE" is replaced, all references are marked and the paragraph is returned.

4.2 Transforming T3DocML to HTML

The actual transformation from T3DocML to HTML is done by a method called `xmltohtml()` of the `DocTransform` class. This method is shown in Listing 4.5. It implements the transformation described in Section 3.4. It uses an instance of the *Transformerfactory* class of Saxon to perform the transformation. The complete XSL stylesheet used for this transformation can be found in Listing 7.2.

```
0  public static void xmltohtml(String foldername, String xmlFileName){
1          System.setProperty("javax.xml.transform.TransformerFactory",
2          "net.sf.saxon.TransformerFactoryImpl");
3          TransformerFactory tfactory = TransformerFactory.newInstance();
4          Transformer transformer;
5          try {
6                  transformer = tfactory.newTransformer(new StreamSource
7                          (new File( xslStylesheet )));
8                  transformer.setParameter("folder", foldername);
9                  transformer.transform(new StreamSource
10                         (new File( xmlFile)),
11                                 new StreamResult(new FileOutputStream
12                                         (foldername + logfile)));
13         } catch (TransformerConfigurationException e) {
14                 e.printStackTrace();
15         } catch (TransformerException e) {
16                 e.printStackTrace();
17         } catch (FileNotFoundException e) {
18                 e.printStackTrace();
19         }
20 }
```

Listing 4.5: The xmltohtml()-method

4.3 Transforming T3DocML to PDF

This transformation is done similar to the one in the previous section by the `xmltopdf()`-method, with the additional use of Apache FOP to create the PDF file from the XSL-FO code. The complete XSL stylesheet used for this transformation can be found in Listing 7.3.

5 Conclusion

This thesis presented an approach for automatic documentation generation from TTCN-3 source code using a standardized documentation comment specification. A prototypical implementation of this approach was successfully tested on several TTCN-3 test suites [22] of industrial size by ETSI and generated technical documentation that uses information from T3Doc documentation blocks. The tool utilizes methods of the eclipse plug-in TRex, which provides an analysis infrastructure for TTCN-3, the open source Saxon XSLT processor, and Apache FOP. For example, tested on ETSI's SIP test suite (v4.2.5, 2,24 MB, 8 files, about 62.000 lines of code), 2.229 documentation files were generated in 359.194ms (6 minutes) on a system with 2,4 GHz Dual Core and 2 GB RAM.

The source code and source code comments are correctly processed from the TTCN-3 files and the cross-references in the HTML documentation are properly set. Cross-references in the PDF documentation are currently not implemented, but are possible by modifying the XSL stylesheet that is used to generate the XSL-FO file.

Bibliography

[1] ETSI. ETSI standard ES 201 873-1: The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, 2009.

[2] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/2008/REC-xml-20081126`, 2008.

[3] H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures Second Edition. `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028`, 2004.

[4] J. Clark. XSL Transformations (XSLT). `http://www.w3.org/TR/1999/REC-xslt-19991116`, 1999.

[5] A. Berglund. Extensible Stylesheet Language (XSL) Version 1.1. `http://www.w3.org/TR/2006/REC-xsl11-20061205`, 2006.

[6] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz. *An Introduction to TTCN-3*. Wiley and Sons, 2005.

[7] International Organization for Standardization. International standard; ISO/IEC 9646-3: Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). Technical report, 1998.

[8] ETSI. ETSI standard ES 201 873-10: The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification, 2008.

[9] International Organization for Standardization. International standard; ISO 8879: Information processing – Text and office systems – Standard Generalized Markup Language (SGML). Technical report, 1986.

[10] S. Pemberton, D. Austin, J. Axelsson, T. Çelik, D. Dominiak, H. Elenbaas, B. Epperson, M. Ishikawa, S. Matsui, S. McCarron, A. Navaron, S. Peruvemba, R. Relyea, S. Schnitzenbaumer, and P. Stark. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). `http://www.w3.org/TR/2002/REC-xhtml1-20020801`, 2002.

[11] D. Winer. Really Simple Syndication 2.0. http://cyber.law.harvard.edu/rss/rss.html, 2002.

[12] D.Beckett. RDF/XML Syntax Specification. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210, 2004.

[13] D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language. http://www.w3.org/TR/2004/REC-owl-features-20040210, 2004.

[14] D. Ragget, A. Le Hors, and I. Jacobs. HTML 4.01 Specification. http://www.w3.org/TR/1999/REC-html401-19991224, 1999.

[15] J. Clark and S. DeRose. XML Path Language (XPath). http://www.w3.org/TR/1999/REC-xpath-19991116, 1999.

[16] International Organization for Standardization. International standard; ISO 32000-1: Document management – Portable document format – Part 1: PDF 1.7. Technical report, 2008.

[17] ECMA. ECMAScript Language Specification. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf, 1999.

[18] B. Zeiß, H. Neukirchen, J. Grabowski, D. Evans, and P. Baker. Refactoring and Metrics for TTCN-3 Test Suites. In *System Analysis and Modeling: Language Profiles*, pages 148–165. Springer, 2006.

[19] SAXON XSLT processor. URL http://saxon.sourceforge.net.

[20] Apache FOP. URL http://xmlgraphics.apache.org/fop.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. 1993.

[22] Public TTCN-3 Test Suites. URL http://www.ttcn-3.org/PublicTTCN3TestSuites.htm.

6 Acronyms

AST *Abstract Syntax Tree*

DTD *Document Type Definition*

ETSI *European Telecommunications Standards Institute*

HTML *Hypertext Markup Language*

PDF *Portable Document Format*

SGML *Standard Generalized Markup Language*

SUT *system under test*

TTCN-2 *Tree and Tabular Combined Notation Version 2*

TTCN-3 *Testing and Test Control Notation Version 3*

W3C *World Wide Web Consortium*

XML *Extensible Markup Language*

XSLT *Extensible Stylesheet Language Transformations*

XSD *XML Schema Definition*

XSL-FO *Extensible Stylesheet Language Formatting Objects*

7 Annex

7.1 XML Schema

```
0  <?xml version="1.0" encoding="UTF-8" ?>
1  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
2    <xs:element name="author">
3      <xs:complexType mixed="true">
4        <xs:choice>
5          <xs:element ref="esee" />
6          <xs:element ref="eurl" />
7        </xs:choice>
8      </xs:complexType>
9    </xs:element>
10   <xs:element name="behaviour">
11     <xs:complexType mixed="true">
12       <xs:choice>
13         <xs:element ref="newline" />
14       </xs:choice>
15     </xs:complexType>
16   </xs:element>
17   <xs:element name="sign">
18     <xs:complexType mixed="true" />
19   </xs:element>
20   <xs:element name="comment">
21     <xs:complexType>
22       <xs:choice>
23         <xs:element ref="author" />
24         <xs:element ref="config" />
25         <xs:element ref="desc" />
26         <xs:element ref="exception" />
27         <xs:element ref="member" />
28         <xs:element ref="param" />
29         <xs:element ref="purpose" />
30         <xs:element ref="remark" />
31         <xs:element ref="return" />
32         <xs:element ref="see" />
33         <xs:element ref="since" />
34         <xs:element ref="status" />
35         <xs:element ref="url" />
36         <xs:element ref="verdict" />
37         <xs:element ref="version" />
```

```
38        </xs:choice>
39      </xs:complexType>
40    </xs:element>
41    <xs:element name="config">
42      <xs:complexType mixed="true">
43        <xs:choice>
44          <xs:element ref="esee" />
45          <xs:element ref="eurl" />
46        </xs:choice>
47      </xs:complexType>
48    </xs:element>
49    <xs:element name="desc">
50      <xs:complexType mixed="true">
51        <xs:choice>
52          <xs:element ref="esee" />
53          <xs:element ref="eurl" />
54        </xs:choice>
55      </xs:complexType>
56    </xs:element>
57    <xs:element name="element">
58      <xs:complexType>
59        <xs:sequence>
60          <xs:element ref="name" />
61          <xs:element ref="location" />
62          <xs:element ref="comment" minOccurs="0" />
63          <xs:element ref="behaviour" />
64          <xs:element ref="sign" />
65        </xs:sequence>
66        <xs:attribute name="type" type="xs:string" use="required" />
67      </xs:complexType>
68    </xs:element>
69    <xs:element name="esee">
70      <xs:complexType mixed="true" />
71    </xs:element>
72    <xs:element name="eurl">
73      <xs:complexType mixed="true" />
74    </xs:element>
75    <xs:element name="exception">
76      <xs:complexType mixed="true">
77        <xs:choice>
78          <xs:element ref="esee" />
79          <xs:element ref="eurl" />
80        </xs:choice>
81      </xs:complexType>
82    </xs:element>
83    <xs:element name="file">
84      <xs:complexType mixed="true" />
85    </xs:element>
86    <xs:element name="group">
```

```
87      <xs:complexType>
88        <xs:sequence>
89          <xs:element ref="name" />
90          <xs:element ref="location" />
91          <xs:element ref="comment" minOccurs="0" />
92          <xs:choice>
93            <xs:element ref="group" />
94            <xs:element ref="element" />
95          </xs:choice>
96        </xs:sequence>
97      </xs:complexType>
98    </xs:element>
99    <xs:element name="location">
100     <xs:complexType mixed="true" />
101   </xs:element>
102   <xs:element name="member">
103     <xs:complexType mixed="true">
104       <xs:choice>
105         <xs:element ref="esee" />
106         <xs:element ref="eurl" />
107       </xs:choice>
108     </xs:complexType>
109   </xs:element>
110   <xs:element name="module">
111     <xs:complexType>
112       <xs:sequence>
113         <xs:element ref="name" />
114         <xs:element ref="comment" minOccurs="0" />
115         <xs:choice>
116           <xs:element ref="group" />
117           <xs:element ref="element" />
118         </xs:choice>
119       </xs:sequence>
120     </xs:complexType>
121   </xs:element>
122   <xs:element name="name">
123     <xs:complexType mixed="true" />
124   </xs:element>
125   <xs:element name="newline" type="xs:string" />
126   <xs:element name="param">
127     <xs:complexType mixed="true">
128       <xs:choice>
129         <xs:element ref="esee" />
130         <xs:element ref="eurl" />
131       </xs:choice>
132     </xs:complexType>
133   </xs:element>
134   <xs:element name="project">
135     <xs:complexType>
```

```
136       <xs:sequence>
137         <xs:element ref="file" minOccurs="0" maxOccurs="unbounded" />
138         <xs:element ref="module" minOccurs="0" maxOccurs="unbounded" />
139       </xs:sequence>
140     </xs:complexType>
141   </xs:element>
142   <xs:element name="purpose">
143     <xs:complexType mixed="true">
144       <xs:choice>
145         <xs:element ref="esee" />
146         <xs:element ref="eurl" />
147       </xs:choice>
148     </xs:complexType>
149   </xs:element>
150   <xs:element name="remark">
151     <xs:complexType mixed="true">
152       <xs:choice>
153         <xs:element ref="esee" />
154         <xs:element ref="eurl" />
155       </xs:choice>
156     </xs:complexType>
157   </xs:element>
158   <xs:element name="return">
159     <xs:complexType mixed="true">
160       <xs:choice>
161         <xs:element ref="esee" />
162         <xs:element ref="eurl" />
163       </xs:choice>
164     </xs:complexType>
165   </xs:element>
166   <xs:element name="see">
167     <xs:complexType mixed="true" />
168   </xs:element>
169   <xs:element name="since">
170     <xs:complexType mixed="true">
171       <xs:choice>
172         <xs:element ref="esee" />
173         <xs:element ref="eurl" />
174       </xs:choice>
175     </xs:complexType>
176   </xs:element>
177   <xs:element name="status">
178     <xs:complexType mixed="true">
179       <xs:choice>
180         <xs:element ref="esee" />
181         <xs:element ref="eurl" />
182       </xs:choice>
183     </xs:complexType>
184   </xs:element>
```

```
185   <xs:element name="url">
186     <xs:complexType mixed="true">
187       <xs:choice>
188         <xs:element ref="esee" />
189         <xs:element ref="eurl" />
190       </xs:choice>
191     </xs:complexType>
192   </xs:element>
193   <xs:element name="verdict">
194     <xs:complexType mixed="true">
195       <xs:choice>
196         <xs:element ref="esee" />
197         <xs:element ref="eurl" />
198       </xs:choice>
199     </xs:complexType>
200   </xs:element>
201   <xs:element name="version">
202     <xs:complexType mixed="true">
203       <xs:choice>
204         <xs:element ref="esee" />
205         <xs:element ref="eurl" />
206       </xs:choice>
207     </xs:complexType>
208   </xs:element>
209 </xs:schema>
```

Listing 7.1: The T3DocML Schema

7.2 XSL Stylesheet - HTML

```
0  <?xml version="1.0"?>
1  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2   xmlns="http://www.w3.org/1999/xhtml" version="2.0">
3  <xsl:output method="xml" indent="yes" name="html"
4   doctype-public="-//W3C//DTD XHTML 1.1//EN"
5   doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
6
7  <xsl:param name="folder"/>
8
9  <xsl:template match="/">
10
11   <xsl:for-each select="//element">
12    <xsl:variable name="filename"
13     select="concat('file:/', $folder, '/html/', location/text())"/>
14    <xsl:value-of select="$filename"/>
15     <xsl:result-document href="{$filename}" format="html">
16
```

```
17 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
18  <head>
19   <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
20   <link rel="stylesheet" type="text/css" href="css/doc.css"/>
21   <script type="text/javascript" src="js/doc.js"></script>
22   <title><xsl:copy-of select="name/text()"/></title>
23  </head>
24
25  <body onload="init('{@type}')">
26   <table id="table_all"><tr>
27    <td id="table_all_left">
28     <xsl:call-template name="completeindex">
29      <xsl:with-param name="this" select="ancestor::module"/>
30     </xsl:call-template>
31    </td>
32    <td id="table_all_right">
33    <xsl:call-template name="header">
34     <xsl:with-param name="this" select="."/>
35    </xsl:call-template>
36     <xsl:call-template name="element_withcomment">
37      <xsl:with-param name="this" select="."/>
38     </xsl:call-template>
39    </td>
40   </tr>
41  </table>
42  </body>
43 </html>
44   </xsl:result-document>
45  </xsl:for-each>
46
47  <xsl:for-each select="//module">
48   <xsl:variable name="filename"
49    select="concat('file:/', $folder, '/html/', name/text(), '.html')"/>
50   <xsl:value-of select="$filename"/>
51    <xsl:result-document href="{$filename}" format="html">
52
53 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
54  <head>
55   <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
56   <link rel="stylesheet" type="text/css" href="css/doc.css"/>
57   <script type="text/javascript" src="js/doc.js"></script>
58   <title><xsl:copy-of select="name/text()"/></title>
59  </head>
60  <body onload="init('module')">
61   <table id="table_all"><tr>
62    <td id="table_all_left">
63     <xsl:call-template name="completeindex">
64      <xsl:with-param name="this" select="."/>
65     </xsl:call-template>
```

```
66      </td>
67      <td id="table_all_right">
68      <xsl:call-template name="header">
69       <xsl:with-param name="this" select="."/>
70      </xsl:call-template>
71      <div id="div_content">
72      <xsl:apply-templates select="comment"/>
73      <div id="div_element">
74       <xsl:call-template name="module_behaviour">
75        <xsl:with-param name="this" select="."/>
76       </xsl:call-template>
77      </div>
78    </div>
79      </td>
80    </tr>
81   </table>
82
83   </body>
84  </html>
85    </xsl:result-document>
86   </xsl:for-each>
87
88   <xsl:for-each select="//group">
89    <xsl:variable name="filename"
90      select="concat('file:/', $folder, '/html/', location/text())"/>
91    <xsl:value-of select="$filename"/>
92      <xsl:result-document href="{$filename}" format="html">
93
94  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
95   <head>
96    <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
97    <link rel="stylesheet" type="text/css" href="css/doc.css"/>
98    <script type="text/javascript" src="js/doc.js"></script>
99    <title><xsl:copy-of select="name/text()"/></title>
100  </head>
101
102  <body onload="init('group')">
103   <table id="table_all"><tr>
104    <td id="table_all_left">
105     <xsl:call-template name="completeindex">
106      <xsl:with-param name="this" select="ancestor::module"/>
107     </xsl:call-template>
108    </td>
109   <td id="table_all_right">
110    <xsl:call-template name="header">
111     <xsl:with-param name="this" select="."/>
112    </xsl:call-template>
113    <div id="div_content">
114    <xsl:apply-templates select="comment"/>
```

```
115     <div id="div_element">
116      <xsl:call-template name="group_behaviour">
117       <xsl:with-param name="this" select="."/>
118      </xsl:call-template>
119     </div>
120    </div>
121     </td>
122    </tr>
123   </table>
124
125   </body>
126  </html>
127    </xsl:result-document>
128   </xsl:for-each>
129
130   <xsl:for-each select="//project">
131    <xsl:variable name="filename"
132     select="concat('file:/', $folder, '/html/index.html')"/>
133    <xsl:value-of select="$filename"/>
134     <xsl:result-document href="{$filename}" format="html">
135
136  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
137   <head>
138    <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
139    <link rel="stylesheet" type="text/css" href="css/doc.css"/>
140    <script type="text/javascript" src="js/doc.js"></script>
141    <title>Index</title>
142   </head>
143
144   <body onload="init()">
145       <table id="table_all"><tr>
146     <td id="table_all_left">
147      <xsl:call-template name="completeindex">
148       <xsl:with-param name="this" select="//module"/>
149      </xsl:call-template>
150     </td>
151     <td id="table_all_right">
152     </td>
153    </tr>
154   </table>
155   </body>
156  </html>
157    </xsl:result-document>
158   </xsl:for-each>
159  </xsl:template>
160
161  <xsl:template match="comment">
162   <div id="div_comment">
163    <xsl:apply-templates select="desc"/>
```

```
164 <xsl:if test="author">
165  <p class="comment_headline">Author(s):</p>
166  <ul class="comment_list">
167   <xsl:apply-templates select="author"/>
168  </ul>
169 </xsl:if>
170 <xsl:if test="config">
171  <p class="comment_headline">Config:</p>
172  <ul class="comment_list">
173   <xsl:apply-templates select="config"/>
174  </ul>
175 </xsl:if>
176 <xsl:if test="exception">
177  <p class="comment_headline">Exception(s):</p>
178  <ul class="comment_list">
179   <xsl:apply-templates select="exception"/>
180  </ul>
181 </xsl:if>
182 <xsl:if test="member">
183  <p class="comment_headline">Member(s):</p>
184  <ul class="comment_list">
185   <xsl:apply-templates select="member"/>
186  </ul>
187 </xsl:if>
188 <xsl:if test="param">
189  <p class="comment_headline">Parameter(s):</p>
190  <ul class="comment_list">
191   <xsl:apply-templates select="param"/>
192  </ul>
193 </xsl:if>
194 <xsl:if test="purpose">
195  <p class="comment_headline">Purpose:</p>
196  <ul class="comment_list">
197   <xsl:apply-templates select="purpose"/>
198  </ul>
199 </xsl:if>
200 <xsl:if test="remark">
201  <p class="comment_headline">Remark(s):</p>
202  <ul class="comment_list">
203   <xsl:apply-templates select="remark"/>
204  </ul>
205 </xsl:if>
206 <xsl:if test="return">
207  <p class="comment_headline">Return:</p>
208  <ul class="comment_list">
209   <xsl:apply-templates select="return"/>
210  </ul>
211 </xsl:if>
212 <xsl:if test="see">
```

```
213   <p class="comment_headline">See:</p>
214   <ul class="comment_list">
215    <xsl:apply-templates select="see"/>
216   </ul>
217  </xsl:if>
218  <xsl:if test="since">
219   <p class="comment_headline">Since:</p>
220   <ul class="comment_list">
221    <xsl:apply-templates select="since"/>
222   </ul>
223  </xsl:if>
224  <xsl:if test="status">
225   <p class="comment_headline">Status:</p>
226   <ul class="comment_list">
227    <xsl:apply-templates select="status"/>
228   </ul>
229  </xsl:if>
230  <xsl:if test="url">
231   <p class="comment_headline">Url(s):</p>
232   <ul class="comment_list">
233    <xsl:apply-templates select="url"/>
234   </ul>
235  </xsl:if>
236  <xsl:if test="verdict">
237   <p class="comment_headline">Verdict(s):</p>
238   <ul class="comment_list">
239    <xsl:apply-templates select="verdict"/>
240   </ul>
241  </xsl:if>
242  <xsl:if test="version">
243   <p class="comment_headline">Version:</p>
244   <ul class="comment_list">
245    <xsl:apply-templates select="version"/>
246   </ul>
247  </xsl:if>
248  </div>
249 </xsl:template>
250
251 <xsl:template match="desc">
252 <p class="comment_desc_line"><xsl:apply-templates/></p>
253 </xsl:template>
254
255 <xsl:template match="author | return | remark | param |
256   since | status | url | verdict | version | member |
257   config | exception | purpose | since">
258 <li class="comment_listitem"><xsl:apply-templates/></li>
259 </xsl:template>
260
261
```

```
<xsl:template match="see">
 <xsl:variable name="text" select="normalize-space(text())" />
 <xsl:choose>
  <xsl:when test="contains($text, '.')">
   <xsl:variable name="file" select="/project/file[contains(text(),
    concat('_e_', substring-after($text, '.'), '.html')) and contains(text(),
    concat('_m_', substring-before($text, '.'), '_o_'))]"/>
   <xsl:if test="$file">
    <li class="comment_listitem">
     <a href="{$file/text()}"><xsl:copy-of select="$text"/></a>
    </li>
   </xsl:if>
   <xsl:if test="not($file)">
    <li class="comment_listitem"><xsl:copy-of select="text()"/></li>
   </xsl:if>
  </xsl:when>
  <xsl:otherwise>
   <xsl:variable name="file"
    select="/project/file[contains(text(), concat('_e_', $text, '.html'))]"/>
   <xsl:if test="$file">
    <li class="comment_listitem">
     <a href="{$file/text()}"><xsl:copy-of select="$text"/></a>
    </li>
   </xsl:if>
   <xsl:if test="not($file)">
    <li class="comment_listitem"><xsl:copy-of select="text()"/></li>
   </xsl:if>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match="esee">
 <xsl:variable name="text" select="normalize-space(text())" />
 <xsl:choose>
  <xsl:when test="contains($text, '.')">
   <xsl:variable name="file" select="/project/file[contains(text(),
    concat('_e_', substring-after($text, '.'), '.html')) and contains(text(),
    concat('_m_', substring-before($text, '.'), '_o_'))]"/>
   <xsl:if test="$file">
    <a href="{$file/text()}"><xsl:copy-of select="$text"/></a>
   </xsl:if>
   <xsl:if test="not($file)">
    <xsl:copy-of select="text()"/>
   </xsl:if>
  </xsl:when>
  <xsl:otherwise>
   <xsl:variable name="file" select="/project/file[contains(text(),
    concat('_e_', $text, '.html'))]"/>
   <xsl:if test="$file">
```

```
311       <a href="{$file/text()}"><xsl:copy-of select="$text"/></a>
312      </xsl:if>
313      <xsl:if test="not($file)">
314       <xsl:copy-of select="text()"/>
315      </xsl:if>
316     </xsl:otherwise>
317    </xsl:choose>
318    <xsl:copy-of select="' '"/>
319   </xsl:template>
320
321   <xsl:template match="behaviour">
322    <div class="div_behaviour">
323     <p class="statement">
324      <xsl:apply-templates/>
325     </p>
326    </div>
327   </xsl:template>
328
329   <xsl:template match="newline">
330    <br/>
331   </xsl:template>
332
333   <xsl:template match="link">
334    <xsl:variable name="loc" select="@loc" />
335
336    <xsl:choose>
337     <xsl:when test="not(/project/file[text() eq $loc])">
338      <xsl:copy-of
339       select="replace(replace(text(), ' ', ' '), '\t', '&#x9;')"/>
340     </xsl:when>
341     <xsl:otherwise>
342      <a href="{@loc}">
343       <xsl:copy-of
344        select="replace(replace(text(), ' ', ' '), '\t', '&#x9;')"></a>
345     </xsl:otherwise>
346    </xsl:choose>
347   </xsl:template>
348
349
350   <xsl:template name="index_group">
351    <xsl:param name="this"/>
352    <xsl:for-each select="$this">
353    <li><a href="{location/text()}"><xsl:copy-of select="name/text()"/></a>
354
355    <xsl:if test="./group">
356     <ul>
357      <xsl:call-template name="index_group">
358       <xsl:with-param name="this" select="./group"/>
359      </xsl:call-template>
```

```
360   </ul>
361   </xsl:if>
362
363
364  </li>
365  </xsl:for-each>
366
367 </xsl:template>
368
369
370 <xsl:template name="index_modulelist">
371    <p class="index_headline"><a href="index.html">Modules</a></p>
372     <ul>
373      <xsl:for-each select="//module">
374       <li>
375        <a href="{name/text()}.html"><xsl:copy-of select="name/text()"/></a>
376      </li>
377      </xsl:for-each>
378     </ul>
379 </xsl:template>
380
381 <xsl:template name="index_grouplist">
382  <xsl:param name="this"/>
383   <p class="index_headline">
384    <a href="#" onclick="toggle('ul_groups')">Groups</a>
385   </p>
386    <ul id="ul_groups">
387     <xsl:call-template name="index_group">
388      <xsl:with-param name="this" select="$this"/>
389     </xsl:call-template>
390     <xsl:if test="not($this)">
391      <li>-</li>
392     </xsl:if>
393    </ul>
394 </xsl:template>
395
396 <xsl:template name="index_elementlist">
397  <xsl:param name="this"/>
398  <xsl:param name="type"/>
399  <p class="index_headline">
400   <a href="#" onclick="toggle('ul_{$type}s')">
401    <xsl:copy-of select="$type"/>s</a>
402  </p>
403  <ul id="ul_{$type}s">
404   <xsl:if test="not($this)">
405    <li>-</li>
406   </xsl:if>
407   <xsl:for-each select="$this">
408    <li>
```

```
409     <a href="{location/text()}"><xsl:copy-of select="name/text()"/></a>
410    </li>
411    </xsl:for-each>
412   </ul>
413  </xsl:template>
414
415
416  <xsl:template name="completeindex">
417   <xsl:param name="this"/>
418    <div id="div_modules">
419     <xsl:call-template name="index_modulelist"/>
420    </div>
421    <div id="div_index">
422
423     <xsl:call-template name="index_grouplist">
424      <xsl:with-param name="this" select="$this/group"/>
425     </xsl:call-template>
426
427     <xsl:call-template name="index_elementlist">
428      <xsl:with-param name="this" select="$this//element[@type eq 'function']"/>
429      <xsl:with-param name="type" select="'Function'"/>
430     </xsl:call-template>
431
432     <xsl:call-template name="index_elementlist">
433      <xsl:with-param name="this" select="$this//element[@type eq 'type']"/>
434      <xsl:with-param name="type" select="'Type'"/>
435     </xsl:call-template>
436
437     <xsl:call-template name="index_elementlist">
438      <xsl:with-param name="this" select="$this//element[@type eq 'signature']"/>
439      <xsl:with-param name="type" select="'Signature'"/>
440     </xsl:call-template>
441
442     <xsl:call-template name="index_elementlist">
443      <xsl:with-param name="this" select="$this//element[@type eq 'testcase']"/>
444      <xsl:with-param name="type" select="'Testcase'"/>
445     </xsl:call-template>
446
447     <xsl:call-template name="index_elementlist">
448      <xsl:with-param name="this" select="$this//element[@type eq 'altstep']"/>
449      <xsl:with-param name="type" select="'Altstep'"/>
450     </xsl:call-template>
451
452     <xsl:call-template name="index_elementlist">
453      <xsl:with-param name="this" select="$this//element[@type eq 'template']"/>
454      <xsl:with-param name="type" select="'Template'"/>
455     </xsl:call-template>
456
457     <xsl:call-template name="index_elementlist">
```

```
458       <xsl:with-param name="this" select="$this//element[@type eq 'constant']"/>
459       <xsl:with-param name="type" select="'Constant'"/>
460     </xsl:call-template>
461
462     <xsl:call-template name="index_elementlist">
463       <xsl:with-param name="this" select="$this//element[@type eq 'parameter']"/>
464       <xsl:with-param name="type" select="'Parameter'"/>
465     </xsl:call-template>
466
467   </div>
468 </xsl:template>
469
470
471 <xsl:template name="modulecontent_behaviour">
472  <xsl:param name="this"/>
473  <xsl:for-each select="$this/element">
474   <xsl:apply-templates select="behaviour"/>
475  </xsl:for-each>
476 </xsl:template>
477
478 <xsl:template name="module_behaviour">
479  <xsl:param name="this"/>
480  <p><xsl:copy-of select="'module '"/>
481   <a href="{$this/location/text()}">
482    <xsl:copy-of select="$this/name/text()"/>
483   </a>
484   <xsl:copy-of select="' {'"/>
485  </p>
486  <div class="div_group_behaviour">
487   <xsl:for-each select="$this/group">
488    <xsl:call-template name="group_behaviour">
489     <xsl:with-param name="this" select="."/>
490    </xsl:call-template>
491   </xsl:for-each>
492   <xsl:call-template name="modulecontent_behaviour">
493    <xsl:with-param name="this" select="$this"/>
494   </xsl:call-template>
495  </div>
496  <p><xsl:copy-of select="'}'"/></p>
497 </xsl:template>
498
499 <xsl:template name="group_behaviour">
500  <xsl:param name="this"/>
501  <p><xsl:copy-of select="'group '"/>
502   <a href="{$this/location/text()}"><xsl:copy-of select="$this/name/text()"/>
503   </a>
504   <xsl:copy-of select="' {'"/>
505  </p>
506  <div class="div_group_behaviour">
```

```
507   <xsl:for-each select="$this/group">
508    <xsl:call-template name="group_behaviour">
509     <xsl:with-param name="this" select="."/>
510    </xsl:call-template>
511   </xsl:for-each>
512   <xsl:call-template name="modulecontent_behaviour">
513    <xsl:with-param name="this" select="$this"/>
514   </xsl:call-template>
515  </div>
516  <p><xsl:copy-of select="'}'"/></p>
517 </xsl:template>
518 
519 <xsl:template name="element_withcomment">
520  <xsl:param name="this"/>
521   <div id="div_content">
522    <xsl:apply-templates select="comment"/>
523 
524    <div id="div_element">
525     <xsl:apply-templates select="$this/behaviour"/>
526    </div>
527   </div>
528 </xsl:template>
529 
530 <xsl:template name="header">
531  <xsl:param name="this"/>
532  <p id="p_header">
533  <xsl:if test="$this/ancestor::module">
534   <a href="{$this/ancestor::module/name/text()}.html">
535    <xsl:copy-of select="ancestor::module/name/text()"/>
536   </a>
537   <br/>
538  </xsl:if>
539 
540  <xsl:if test="$this/ancestor::module"><xsl:copy-of select="'-> '"/></xsl:if>
541   <xsl:copy-of select="$this/name/text()"/></p>
542 
543 </xsl:template>
544 
545 </xsl:stylesheet>
```

Listing 7.2: The XSL Stylesheet (HTML)

7.3 XSL Stylesheet - XSL-FO

```
0  <?xml version="1.0" encoding="UTF-8"?>
1  <xsl:stylesheet version="1.0"
2     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3     xmlns:fo="http://www.w3.org/1999/XSL/Format">
4   <xsl:output method="xml" indent="yes"/>
5
6  <xsl:template match="/">
7
8  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
9
10 <fo:layout-master-set>
11   <fo:simple-page-master master-name="A4">
12     <fo:region-body padding-left="0pt" padding-right="0pt"
13       padding-top="0pt" padding-bottom="0pt"/>
14   </fo:simple-page-master>
15 </fo:layout-master-set>
16
17 <fo:page-sequence master-reference="A4">
18  <fo:flow flow-name="xsl-region-body">
19   <xsl:apply-templates select="//module"/>
20  </fo:flow>
21 </fo:page-sequence></fo:root>
22
23 </xsl:template>
24
25 <xsl:template name="modulecontent">
26  <xsl:param name="this"/>
27
28 <xsl:if test="count($this/element[@type eq 'function']) &gt; 0">
29    <fo:block border="3pt solid black" border-bottom-style="hidden"
30     border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
31     <fo:block text-align="left" font-weight="bold" font-size="10pt"
32      margin-top="10pt">
33      <fo:inline>Functions</fo:inline>
34     </fo:block>
35     <xsl:apply-templates select="$this/element[@type eq 'function']"/>
36    </fo:block>
37   </xsl:if>
38
39   <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
40    <fo:block border="3pt solid black" border-bottom-style="hidden"
41     border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
42     <fo:block text-align="left" font-weight="bold" font-size="10pt"
43      margin-top="10pt">
44      <fo:inline>Testcases</fo:inline>
45     </fo:block>
```

```
46      <xsl:apply-templates select="$this/element[@type eq 'testcase']"/>
47     </fo:block>
48    </xsl:if>
49
50    <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
51     <fo:block border="3pt solid black" border-bottom-style="hidden"
52      border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
53      <fo:block text-align="left" font-weight="bold" font-size="10pt"
54       margin-top="10pt">
55       <fo:inline>Altsteps</fo:inline>
56      </fo:block>
57      <xsl:apply-templates select="$this/element[@type eq 'altstep']"/>
58     </fo:block>
59    </xsl:if>
60
61    <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
62     <fo:block border="3pt solid black" border-bottom-style="hidden"
63      border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
64      <fo:block text-align="left" font-weight="bold" font-size="10pt"
65       margin-top="10pt">
66       <fo:inline>Constants</fo:inline>
67      </fo:block>
68      <xsl:apply-templates select="$this/element[@type eq 'constant']"/>
69     </fo:block>
70    </xsl:if>
71
72    <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
73     <fo:block border="3pt solid black" border-bottom-style="hidden"
74      border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
75      <fo:block text-align="left" font-weight="bold" font-size="10pt"
76       margin-top="10pt">
77       <fo:inline>Parameters</fo:inline>
78      </fo:block>
79      <xsl:apply-templates select="$this/element[@type eq 'parameter']"/>
80     </fo:block>
81    </xsl:if>
82
83    <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
84     <fo:block border="3pt solid black" border-bottom-style="hidden"
85      border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
86      <fo:block text-align="left" font-weight="bold" font-size="10pt"
87       margin-top="10pt">
88       <fo:inline>Templates</fo:inline>
89      </fo:block>
90      <xsl:apply-templates select="$this/element[@type eq 'template']"/>
91     </fo:block>
92    </xsl:if>
93
94    <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
```

```
95      <fo:block border="3pt solid black" border-bottom-style="hidden"
96       border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
97       <fo:block text-align="left" font-weight="bold" font-size="10pt"
98        margin-top="10pt">
99        <fo:inline>Types</fo:inline>
100      </fo:block>
101      <xsl:apply-templates select="$this/element[@type eq 'type']"/>
102     </fo:block>
103    </xsl:if>
104
105    <xsl:if test="count($this/element[@type eq 'testcase']) &gt; 0">
106     <fo:block border="3pt solid black" border-bottom-style="hidden"
107      border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
108      <fo:block text-align="left" font-weight="bold" font-size="10pt"
109       margin-top="10pt">
110       <fo:inline>Signatures</fo:inline>
111      </fo:block>
112      <xsl:apply-templates select="$this/element[@type eq 'signature']"/>
113     </fo:block>
114    </xsl:if>
115
116    <xsl:if test="count(group) &gt; 0">
117     <fo:block border="3pt solid black" border-bottom-style="hidden"
118      border-left-style="hidden" border-right-style="hidden" margin-top="7pt">
119      <fo:block text-align="left" font-weight="bold" font-size="10pt"
120       margin-top="10pt">
121       <fo:inline font-weight="bold">Groups</fo:inline>
122      </fo:block>
123      <xsl:apply-templates select="group"/>
124     </fo:block>
125    </xsl:if>
126
127  </xsl:template>
128
129  <xsl:template match="module">
130   <fo:block margin-top="30pt" margin-left="20pt" margin-right="20pt"
131    padding="10pt" border="1pt solid black" background-color="#DDDDDD"
132     break-before="page">
133    <fo:block font-size="12pt" text-decoration="underline" text-align="center"
134     font-weight="bold" margin-top="10pt" margin-bottom="20pt">
135     <fo:inline>Module  <xsl:copy-of select="name/text()"/></fo:inline>
136    </fo:block>
137    <xsl:apply-templates select="comment"/>
138    <xsl:call-template name="modulecontent">
139     <xsl:with-param name="this" select="."/>
140    </xsl:call-template>
141   </fo:block>
142  </xsl:template>
143
```

```
144
145 <xsl:template match="group">
146
147  <fo:block margin-top="5pt" padding-left="5pt" padding-right="5pt"
148    border="1pt solid black" background-color="#DDDDDD" font-size="8pt">
149   <fo:block text-align="left" padding="5pt" border="1pt solid black"
150    background-color="#BBBBFF" font-size="8pt">
151    <fo:inline font-weight="bold">
152     Group <xsl:copy-of select="name/text()"/>
153    </fo:inline>
154   </fo:block>
155   <xsl:apply-templates select="comment"/>
156   <fo:block margin-left="5pt" margin-right="5pt">
157    <xsl:call-template name="modulecontent">
158     <xsl:with-param name="this" select="."/>
159    </xsl:call-template>
160   </fo:block>
161  </fo:block>
162 </xsl:template>
163
164 <xsl:template match="element">
165   <fo:block text-align="left" margin-top="10pt" padding="5pt"
166    border="1pt solid black" background-color="#BBBBFF" font-size="8pt">
167    <fo:inline font-weight="bold">
168     <xsl:copy-of select="sign/text()"/>
169    </fo:inline>
170   </fo:block>
171   <xsl:apply-templates select="comment"/>
172 </xsl:template>
173
174
175
176 <xsl:template match="comment">
177  <fo:block text-align="left" margin-bottom="0pt" padding="5pt"
178   background-color="#EEEEFF" border="1pt solid black" font-size="8pt">
179   <xsl:if test="count(desc) &gt; 0">
180     <xsl:apply-templates select="desc"/>
181   </xsl:if>
182   <fo:block margin-bottom="18pt"/>
183
184   <xsl:if test="count(param) &gt; 0">
185    <fo:inline font-weight="bold">Parameter(s):</fo:inline>
186    <fo:list-block margin-left="30pt">
187     <xsl:apply-templates select="param"/>
188    </fo:list-block>
189   </xsl:if>
190
191   <xsl:if test="count(return) &gt; 0">
192    <fo:inline font-weight="bold">Returns:</fo:inline>
```

```
193     <fo:list-block margin-left="30pt">
194       <xsl:apply-templates select="return"/>
195     </fo:list-block>
196   </xsl:if>
197
198   <xsl:if test="count(author) &gt; 0">
199     <fo:inline font-weight="bold">Author(s):</fo:inline>
200     <fo:list-block margin-left="30pt">
201       <xsl:apply-templates select="author"/>
202     </fo:list-block>
203   </xsl:if>
204
205   <xsl:if test="count(remark) &gt; 0">
206     <fo:inline font-weight="bold">Remark(s):</fo:inline>
207     <fo:list-block margin-left="30pt">
208       <xsl:apply-templates select="remark"/>
209     </fo:list-block>
210   </xsl:if>
211
212   <xsl:if test="count(status) &gt; 0">
213     <fo:inline font-weight="bold">Status:</fo:inline>
214     <fo:list-block margin-left="30pt">
215       <xsl:apply-templates select="status"/>
216     </fo:list-block>
217   </xsl:if>
218
219   <xsl:if test="count(url) &gt; 0">
220     <fo:inline font-weight="bold">Url(s):</fo:inline>
221     <fo:list-block margin-left="30pt">
222       <xsl:apply-templates select="url"/>
223     </fo:list-block>
224   </xsl:if>
225
226   <xsl:if test="count(verdict) &gt; 0">
227     <fo:inline font-weight="bold">Verdicts(s):</fo:inline>
228     <fo:list-block margin-left="30pt">
229       <xsl:apply-templates select="verdict"/>
230     </fo:list-block>
231   </xsl:if>
232
233   <xsl:if test="count(version) &gt; 0">
234     <fo:inline font-weight="bold">Version:</fo:inline>
235     <fo:list-block margin-left="30pt">
236       <xsl:apply-templates select="version"/>
237     </fo:list-block>
238   </xsl:if>
239
240   <xsl:if test="count(member) &gt; 0">
241     <fo:inline font-weight="bold">Member(s):</fo:inline>
```

```
242     <fo:list-block margin-left="30pt">
243       <xsl:apply-templates select="member"/>
244     </fo:list-block>
245   </xsl:if>
246
247     <xsl:if test="count(config) &gt; 0">
248     <fo:inline font-weight="bold">Config:</fo:inline>
249     <fo:list-block margin-left="30pt">
250       <xsl:apply-templates select="config"/>
251     </fo:list-block>
252   </xsl:if>
253
254     <xsl:if test="count(exception) &gt; 0">
255     <fo:inline font-weight="bold">Exception(s):</fo:inline>
256     <fo:list-block margin-left="30pt">
257       <xsl:apply-templates select="exception"/>
258     </fo:list-block>
259   </xsl:if>
260
261     <xsl:if test="count(purpose) &gt; 0">
262     <fo:inline font-weight="bold">Purpose:</fo:inline>
263     <fo:list-block margin-left="30pt">
264       <xsl:apply-templates select="purpose"/>
265     </fo:list-block>
266   </xsl:if>
267
268     <xsl:if test="count(since) &gt; 0">
269     <fo:inline font-weight="bold">Since:</fo:inline>
270     <fo:list-block margin-left="30pt">
271       <xsl:apply-templates select="since"/>
272     </fo:list-block>
273   </xsl:if>
274
275     <xsl:if test="count(see) &gt; 0">
276     <fo:inline font-weight="bold">See:</fo:inline>
277     <fo:list-block margin-left="30pt">
278       <xsl:apply-templates select="see"/>
279     </fo:list-block>
280   </xsl:if>
281
282  </fo:block>
283
284 </xsl:template>
285
286 <xsl:template match="desc">
287  <fo:block>
288   <xsl:apply-templates/>
289  </fo:block>
290 </xsl:template>
```

```
291
292 <xsl:template match="text">
293   <xsl:copy-of select="text()"/>
294 </xsl:template>
295
296
297 <xsl:template match="esee">
298   <xsl:copy-of select="text()"/>
299   <xsl:copy-of select="' '"/>
300 </xsl:template>
301
302 <xsl:template match="see">
303  <fo:list-item>
304   <fo:list-item-label>
305    <fo:block></fo:block>
306   </fo:list-item-label>
307   <fo:list-item-body>
308     <fo:block><xsl:copy-of select="text()"/></fo:block>
309   </fo:list-item-body>
310  </fo:list-item>
311 </xsl:template>
312
313 <xsl:template match="author | return | remark | param | since | status |
314  url | verdict | version | member | config | exception | purpose | since">
315  <fo:list-item>
316   <fo:list-item-label>
317    <fo:block></fo:block>
318   </fo:list-item-label>
319   <fo:list-item-body>
320     <fo:block><xsl:apply-templates/></fo:block>
321   </fo:list-item-body>
322  </fo:list-item>
323 </xsl:template>
324
325 </xsl:stylesheet>
```

Listing 7.3: The XSL Stylesheet (XSL-FO)