Communicating Software Specifications using XML: OpenSpec

Martin v. Mohrenschildt
Computing and Software
Faculty of Engineering
McMaster University
mohrens@mcmaster.ca

June 9, 1999

Abstract

In an ideal world, software specifications are machine readable (can be parsed) much of the tedious verification tasks could be automated by tools. But even if the specification can be processed by one specific tool it is often worthless to other tools not to mention that machine readable specifications are normally not easily read by humans. The first step to solve these obstacles is to develop and define a standard which allows communicate, meaning exchange semantical as well as typesetting information of software specifications between different tools and word processing systems including browsers.

The Extensible Markup Language XML allows to represent semantical as well as formatting information of a document. Purpose of this paper is to propose a standard called OpenSpec based on XML. The OpenSpec standard is designed to communicate formal and semi-formal software specifications. Further, OpenSpec allows to have different "views" of the same document. The presented approach is not restricted to one particular type of specification notation but is intended to be acceptable to "all" approaches to formal and informal specifications. Special consideration is given to support tabular specifications. We developed several software tools for parsing, writing, transforming, and type checking OpenSpec documents.

1 Introduction

The goal of this paper is to define the standard called *OpenSpec* (Open Specification) and to present some first tools for its support. The goal of OpenSpec is *to communicate formal and semi-formal software specifications.* Communicate in this context means to exchange information in between different software tools while preserving the semantics. A formal of semi-formal software specification

is in this context a document which describes the intended, required, or actual behavior of software using either mathematical language or structured natural language.
We like to point out that OpenSpec is designed such that it can represent the mixture of formal and informal information and even pictures, diagrams or other type for text processing information that could be part of a specification document. OpenSpec is not a new specification language by itself, it allows to represent and communicate specifications written in some (see later) specification language or format specifications
Based on past experiences with formal specifications and examining existing tools we concluded that the OpenSpec standard should meet the following criteria:

Criteria

- The standard should be able to incooperate and represent the different types of notations used in mathematics and computer science.
- The resulting documents should be easily transmitted electronically using the web and E-mail.
- The standard should allow a formal specification to be part of a longer documents that could contain elements such as graphics, tables, hyperlinks, indices, table of content, etc.
- Language independent: The standard should allow to represent text in many natural languages.
- The grammar used in the standard should be easy to be parsed (machine precessed) by a software tool.
- Semantical and typesetting information should be separated.
- It should be easy to convert an OpenSpec document into a typeset document in order to allow different "views" of the same document.
- The number of optional features has to be kept to a minimum.
- Platform independent

The intended audience for OpenSpec could be split into three groups:

- *Groups developing, extending, or defining formal and semi-formal specification techniques*
  Many different formal specification languages where developed. Most of them use ASCII based representations. We list some of the specification languages or methods which in our opinion can easily be represented using OpenSpec: SCR, Tabular, VDM, Z, B, etc..

- *Industries preparing software specifications*
  The group profiting the most form a specifications standard would be software developers using or required to use formal specifications such as nuclear industries, aircraft industries, etc.. The use of OpenSpec would allow to use different tools for the verification, and minimize the need to develop custom software drastically.

- *Software Systems*
  We classify the software systems or tools into three different types of systems:

  - **Specification Tools** Tools that where developed to develop specifications in specific specification languages such as TTS, SCR, tools for Z (Zeves, Zola, ZTC), B, OBJ, ML, Lotos, etc..
  - **General Symbolic Verification Systems**: Prove verification systems such as PVS, IMPS, Coq, Larch etc, computer algebra systems such as Maple and Mathematica
  - **Text-processing Systems and Browsers**: Typesetting and formatting systems such as Netscape, various word processors, Tex (Latex) RTF (Ritch Text Format), and PDF.
  - **Software development systems**: Systems which incooperate specification, documentation and code into one "hypertext" document. (e.g. Power-Builder).

Approach

There exist many formats to exchange typesetting information e.g. Adobe PDF, Rich Text Format RTF, HTML and others. These languages where developed with typesetting in mind and are not well suited to exchange semantical information. Language such as Postscript or HTML and systems such as E-MAIL or newsgroups are very wide spread. We think that one of the contributing factors to there success is that they are ASCII based. Nearly each computer system is capable to store, read and process ASCII information.
Investigating different possibilities we found that the *Extensible Markup Language XML* (see chapter 2) provides an ideal base to satisfy our goal. XML enables us to represent formal mathematical expressions as well as text and graphics, it is ASCII based, easy to parse (machine processed) and supported by an increasing [xml95] number of text processing systems and web browser.

2 XML

The goal of this section is not to give a complete introduction of XML but to highlight the feathers of XML which convinced us that XML is an ideal base to develop the OpenSpec standard.

Most people using the web have encountered the language HTML, used to compose and format web pages. The Extensible Markup Language (XML) is a subset of SGML (Standard Generalizes Markup Language, ISO 8879:1996) [xml95] that has been defined by the World Wide Web Consortium (W3C) [xml95]. Its goal is to enable generic SGML to be served, received, and processed on the web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interportability with both SGML and HTML. Even thought XML is relatively new, it is already widely accepted and used by many of the large software cooperations.
XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Entities are identified by so called elements. Parsed data is made up of characters, following a specific grammar. The concept might be known to the reader form HTML documents where elements called tags provide and structure information which is used for example by a browser e.g. `<H> String </H>` could mean that the sting is a title. In contrast to HTML, XML allows to create user defined elements. This is the basic mechanism which allows to give semantical meaning or to documents. Any algebraic expression can easily be translated into XML. Additionally, each of the tags can contain meta-information in form of attributes. Attributes are used to provide additional semantical or formatting information that can be used by other tools.
This paper is not the first to use XML to communicate mathematical objects. A good example of such an approach is OpenMath [OpenMath]. OpenMath is used to provide semantical information of mathematical expression in order to exchange mathematical objects inbetween computer algebra systems and libraries.
The formal character of XML allows to easily represent mathematical objects. For example the following XML (OpenSpec) expression

```
<OS_EXP>
     <OS_APP>
          <OS_NAME> f</OS_NAME>
          <OS_SYMB> x </OS_SYMB>
     <OS_APP>
</OS_EXP>
```

describes a function f which is applied to the symbol x. A formatting tool would generate some thing like

$$f(x),$$

while a type checking tool could verify type information provided by some attributes or type declarations in other parts of the OpenSpec document.
Let us summarize the main advantages of XML which convinced us that it is the right basis for OpenSpec:

- XML is a well defined international standard.
- The documents are very portable and easy to be machine processed.
- It is easy to represent mathematical expressions and structures.

- OpenSpec to mix formal, algebraic, logical information, and informal information such as text and pictures.
- Large software companies such as Microsoft, Sun, IBM, Lotus and others expressed high interest into XML and announced products using XML. XML is supported or is going to be supported in the near future by systems such as Mozilla (Netscape), Microsoft Word, Internet-Explorer 5.0, FrameMaker, Interleave, ...
- XML allows to describe natural languages which are not easily represented using the standard ASCII characters such as Chinese, Russian, Arabic, ... using MIME encodings.

2.1 Writing XML Documents

Clearly no-one would like to enter XML expressions, which can be long, by hand. Basically there are four ways to generate the XML expressions:

- **Composers:** There exists already several XML browsers, like Microsoft Ethernet-Explorer, Amaya [Amaya], etc. which provide an interface to generate XML expressions.
- **Converters:** We can easily[1] write converts which converts specifications written in different languages, e.g. PVS, TTS (McMaster table tools system), LaTeX, and other tools to and from there XML representation. Also the style language XSL containing the transformation language XSLT [xsl95] provides macro mechanism which allow to transform XML documents. We found that in some cases straight forward to use XSL to build converters. (See Appendix A and 4.) We build an ASCII to OpenSpec converter (See 6 and Appendix C).
- **XML Editors** Recently we found XML editors which allow to compose XML documents directly [IBM], citelotus.
- **Word Processors** Several word processors including Microsoft Word and soon Frame-Maker and Interleave will support XML.

3 OpenSpec

Designing the grammar for OpenSpec generality was our highest priority. OpenSpec extends XML with a small number of additional tags. These tags where chosen to provide a minimal required structure to OpenSpec documents.
Examining specifications we came to the conclusion that beside text and graphics specifications typically contain the following types of elements. (It is not our intention to state what a good or even what a specification is.)

[1] Using yacc-lex or similar tools or XSL (XSLT to be precise, see 6.3).

- **Sort or Type Declarations**: Used to declare sorts or types used within the specification.
- **Symbol and Function Declarations**: Used for variable and function declarations.
- **Definitions**: Definitions are primarily used for assigning symbols with expressions. We do not allow assignments within a mathematical expressions.
- **Mathematical Statements**: Mathematical statements are statements containing no free variables such as Theorems, identities.
- **Tabular expressions**: Information presented in tabular form.
- **Relations and Functions** Relations and functions used in the specifications such as abstractions functions. Also axioms, rewrite rules, and evaluation rules would all into this category.
- **Code Segments** Specifications sometimes contain code segments. We could exchange (documented) code in form of a XML document.

We propose the following grammar for OpenSpec. The grammar is given in several parts which allows to give more detailed information. The grammar is given in EBNF. The grammar is presented in form of productions, nonterminals are bold, we use the regular expression constructors | to represent alternatives, $*$ for any number of repetitions, + for one or more repetitions, ? for none or one repetition and [ ] for grouping. A DTD [2] (Document Type Definition) for OpenSpec can bout found in appendix A. I give a very compact representation of the OpenSpec grammar and is used by XML tools to validate an OpenSpec document.

[2] DTD is a XML document which gives a grammar for the OpenSpec elements

OpenSpec 3.1

doc	::=	[**spec**| **xml**]*
spec	::=	**sort** | **decl**| **def** | **theorem** |
		|**tabular** | **relation** |**code**
sort	::=	<OS_SORT **attri**> S? **name** S? **exp** S? </OS_SORT>
decl	::=	<OS_DECL **attri**> S? **name** S? (**exp**| **sig**) S? </OS_DECL>
def	::=	<OS_DEF **attri**> S? **name** S? **exp** S? </OS_DEF>
theorem	::=	<OS_THEO **attri**> S? **name** S? **exp** S? </OS_THEO>
tabular	::=	<OS_TAB **attri**> S? **name** S? **tab** S? </OS_TAB>
relation	::=	<OS_REL **attri**> S? **name** S? **exp** S? </OS_REL>
code	::=	<OS_CODE **attri**> S? **exp** S? </OS_CODE>
attri	::=	[S? Name="CString" S?]*
name	::=	<OS_NAME **attri**> S? String S? </OS_NAME>
sig	::=	<OS_DOM> **exp*** </OS_DOM> <OS_RAN> **exp*** </OS_RAN>
exp	::=	**alg** | **xml**
xml	::=	any xml expression
S	::=	weight space

Comments:

- An OpenSpec document has to be a well formed [xml] XML document. (Well formed is defined as part of the XML specification.)
- Attributes can freely be used to provide informations on specific formates as well as for typesetting information.
- It is allowed to put hypertext labels and links at any place of the document. This means that <a ..> tags are ignored if they occur inbetween OpenSpec tags.
- We use three types of strings.
  - *String* is an ASCII string not containing $< [\mathtt{a}-\mathtt{zA}-\mathtt{Z}]$, (< is not followed by a letter)
  - *CString* is an ASCII String not containing ".
  - *Name* is an letter sequence: $[\mathtt{a}-\mathtt{zA}-\mathtt{Z}][\mathtt{a}-\mathtt{zA}-\mathtt{Z0}-\mathtt{9}_]$
- We allow **exp** to be any xml expression. This is to acknowledge that not all specifications are entirely formal (based on mathematical notation) but could contain elements written in a natural language or even diagrams.
- For sort declaration we provide OS_DOM and OS_RAN to give the domain and the range.

3.1 Mathematical Expressions

Any formal software specification contains mathematical expressions. Mathematical expression are commonly seen in a tree form. For example $f(a, g(3, 4))$ represented as a tree

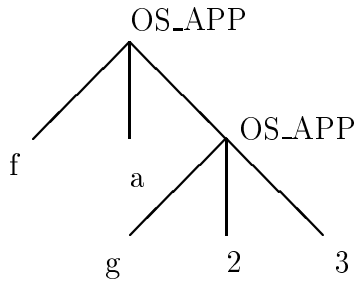

Figure 1

In a tree we distinguhich constructors, the tree branches and leaves. We propose the following very general way to represent mathematical expressions:

OpenSpec 3.2

alg	::=	<OS_EXP **attri**>	S? **algexp** S? </OS_EXP>
algexp	::=	**atom** | **constr**	
constr	::=	<OS_CONS **attri**>	S? **algexp**+ S? </OS_CONS>
		|<OS_BIND **attri**>	S? **name** S?
		(**symb** S? **dom**?)+	S? **algexp** S? </OS_BIND>
		|<OS_APP **attri**>	**name** S? **algexp**+ S? </OS_APP>
dom	::=	<OS_DOM **attri**>	S? String S? </OS_DOM>
atom	::=	**symb** | <OS_CONST> S? Number S? </OS_CONST> | **tabular**	
symb	::=	<OS_SYM **attri**>	S? String S? </OS_SYM>

Comments:

- Inspired by comments and by OpenMath [OpenMath] we provide three types of constructors: construction, application and binding.
  - Construction is used to create collections or data types such as lists, sets, records.
  - Application is used to apply a function to expressions
  - Binding is used to bind variables of an expression to an operator. Binding can be used for quantification, lambda expressions, or operators (with operator we mean objects that return functions). Attributes are used to provide domain information.
  - Again, attributes are used to provide any additional information, e.g. <OS_CONSTtype="real"> 3.14192 </OS_CONST>.
  - Numbers can be integer or real numbers [0-9]'.'[0-9]*

- A tables can occur inside an expressions to allow to represent expressions containing tables and tables containing tables.

It is straight forward, thinking about trees, to represent mathematical expressions using our notation.

Example 3.3 *We give some examples. (Note that the formatting chosen is not required.)*

- $f(a, g(2, 3))$ *would be represented as*

```
<OS_EXP>
   <OS_APP>
        <OS_NAME> f </OS_NAME>
        <OS_SYM> a </OS_SYM>
        <OS_APP>
             <OS_NAME> g <OS_CONST>
             <OS_CONST> 2</OS_CONST>
             <OS_CONST> 3 </OS_CONST>
        <\OS_APP>
   </OS_APP>
</OS_EXP>
```

- *A statement like* $(\forall(x)\ even(2 * x))$ *would we:*

```
<OS_EXP>
   <OS_BIND>
        <OS_NAME> forall </OS_NAME>
        <OS_SYM> x </OS_SYM>
        <OS_APP>
             <OS_NAME> even </OS_NAME>
              <OS_APP pos="infix">
                 <OS_NAME> * <OS_NAME>
                 <OS_CONST> 2</OS_CONST>
                 <OS_SYM> x   </OS_SYM>
              </OS_APP>
        </OS_APP>
   </OS_CONS>
</OS_EXP>
```

- *Or the* $[1, 2]$*;*

```
<OS_EXP>
   <OS_CONS>
        <OS_CONST> 1 </OS_CONST>
        <OS_CONST> 2 </OS_CONST>
        <OS_CONST> 3 </OS_CONST>
   </OS_CONS>
</OS_EXP>
```

3.2 Tabular Expressions

We have a special interest into tabular representations [Par94.1], [Jan95.2], [Heit96], [Par94.3]. Tabular representations are used very commonly to represent functions, relation, condition-action relations, finite state machines, and other types of informations. The semantics of a tabular expression is not determined by just the content of the table, additional information on "how to read a table" is needed. The Semantical information such as the formal rules presented in [Jan95.2] are stored using an additional element provided, or can also be given using attributes.
A table is a tuple $T = (G, H_1, H_2, \cdots, H_n)$ where G is called a grid and the H_i are called headers. Grids and headers are n-dimensional indexed sets. We give a grammar in order to represent tabular expressions.

OpenSpec 3.4

tab	::=	**grid** S? **header**+ S? [**info**S?]*
grid	::=	<OS_GRID **attri**> S? **ele*** S? </OS_GRID>
header	::=	<OS_HEAD **attri**> S? **ele*** S? </OS_HEAD>
ele	::=	**exp** |<OS_CONS **attri**> S? **ele+** S? </OS_CONS>
info	::=	<OS_INFO **attri**> S? **exp** S? </OS_INFO>

Comments:

- We do not require that the dimensions of the grid and headers correspond. This should be verified by some tool.
- Note, even we use the same tag `OS_CONS` as for expressions we will not have any parsing problems; expressions are inside a `OS_EXP` tag.
- We allow non formal (non-mathematical) table elements via **exp**.

For example the table

	$h11$	$h120$
$h21$	$f1$	$f2$
$h22$	$f3$	$f4$

is translated to the following OpenSpec expression:

```
<OS_TAB>
  <OS_GRID>
       <OS_CONS>
         <OS_EXP> <OS_SYMB> f1 </OS_SYMB> </OS_EXP>
         <OS_EXP> <OS_SYMB> f2 </OS_SYMB> </OS_EXP>
       </OS_CONS>
       <OS_CONS>
         <OS_EXP> <OS_SYMB> f3 </OS_SYMB> </OS_EXP>
         <OS_EXP> <OS_SYMB> f4 </OS_SYMB> </OS_EXP>
       </OS_CONS>
  </OS_GRID>
```

```
  <OS_HEAD>
        <OS_EXP> <OS_SYMB> h11 </OS_SYMB> </OS_EXP>
        <OS_EXP> <OS_SYMB> h12 </OS_SYMB> </OS_EXP>
  </OS_HEAD>
  <OS_HEAD>
        <OS_EXP> <OS_SYMB> h21 </OS_SYMB> </OS_EXP>
        <OS_EXP> <OS_SYMB> h22 </OS_SYMB> </OS_EXP>
  </OS_HEAD>
</OS_TAB>
```

4 Typesetting of OpenSpec Documents

As the reader might have noticed, an OpenSpec document does not contain typesetting information. We consider it as essential to our approach that we separate semantical and typesetting information. Typesetting is in some sense a matter iof taste. But clearly an OpenSpec document provides sufficient information which can be used to typeset an OpenSpec document. Further the use of attributes allows to provide additional typesetting information.
There exist manly two typesetting standards used for XML documents: XSL [xsl] and CSS [CSS]. There is an ongoing discussion if both are needed. We decided to use XSL since mainly since bisde beeing a style language it also provides a powerfull macro processer (XML document transformer) which is not restricted to XSL syles 6.3 and a XSL document is a XML document gain. XSL provides typesetting and a sophisticated macro processor (pattern matcher) which translated XML documents into a formatted languages understood by browsers. The advantage of this approach is that someone loading an OpenSpec specification can use his XSL definitions resulting in a typesetting style of his taste or needs. It is straight forward to implement an OpenSpec database, OpenSpec documents are stored on server, browsers can make requests over the web, the OpenSpec documents are transformed to the requested format and transmitted. We like to point out that an OpenSpec documetn can also easily be converted into a Latex document by modifying our XSLT 6.3 transformations, see also appendix B.
We developed set of XSL macros (XSLT to be precise) which will allow to transform an OpenSpec document into HTML. This intention is to demonstrate the incorporation of XSL into OpenSpec and to provide a default set of transformations for the tags of OpenSpec. We transform to HTML since all browsers supprt HTML but not many XSL at the moment.

4.0.1 Expressions

For the transformation of expressions we provide some attributes to typeset the constructors `OS_CONS, OS_APP`.

- For `OS_APP` we provide

```
mode="[ prefix| infix| wrapped ]"
```

The default is prefix. Wrapped is used to enclose the function arguments e.g. $|S|$.

- For `OS_CONS` and `OS_APP` th the `wrapped` attribute we can use the attributes

  ```
  open= "String" close= "String"
  ```

  which will give the symbols used for wrapping, the default of `OS_CONS` is `open= "[" close= "]"`.

For example:

```
<OS_EXP>
   <OS_CONS  open="{" close="}">
           <OS_SYM> a </OS_SYM>
           <OS_SYM> b </OS_SYM>
           <OS_SYM> c </OS_SYM>
   </OS_CONS>
</OS_EXP>
```

Will print as $\{a, b, c\}$.

4.0.2 Formatting of Tabular Expressions

The question of formatting is especially interesting for tabular expressions. If tabular expression is two dimensional we define attributes which allow an automated formatting. The advantage of this approach is that everyone who defined a new type of tables does not have to implement any typesetting transformations, but can just use our attributes.

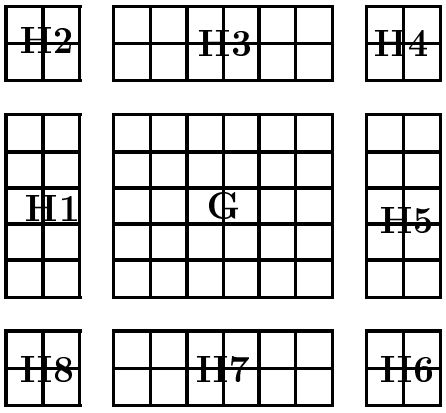

Figure 2

The attributes we support for tables are:

- The attributes for the headers are

  ```
  pos= "[ H1| H2| H3| H4 | H5 | H6| H7| H8 ]"
  ```

  to indicate which position a header goes. (see Figure 2)

- And for both gird and headers

  ```
  dir="[horizontal | vertical ]"
  ```

  To indicate if the gird (head) matrix is a row or column matrix. Default is column, if the header or grid is one dimensional this attribute is ignored.

Note, it is the responsibility of a tool, the formating tool to verify that the dimensions of the headers and girds match.
Studying literature (e.g. a survey of table types in [Abr97] we found that we can represent all the different types of tables found in literature and even more.

5 Ongoing and Future Work

6 Software Tools

In order to actually use OpenSpec we need tools that can parse an OpenSpec specification. We are developing an open source library resulting in a tool, that provides the basic functionality needed to read and and write OpenSpec documents. A prototype is completed. The intention is that this prototype can serve as a starting bases for OpenSpec tools.

6.1 A Prototype Tool

To test your ideas we are developing a prototype application PKOS which can read and write and type check OpenSpec documents. The base system of PKOS consists of three major components which are combined into a library:

- The **Kernel** allows to represent (store) an OpenSpec specification and provides access programs which allow to manipulate the data structures.
- Two **Parsers**:
  - A XML (OpenSpec) parser which can read XML documents and parse the OpenSpec elements.
  - An ASCII parser which reads an ASCII representation of sort, variable and function declarations and expressions. (It makes entering OpenSpec expressions easy and can be used to parse output form other tools such as Maple, PVS, ...) See Appendix B.
- An **Export module** which allows to export the data structures of the kernel as a XML document again.

We like to point out that this was accomplished with very little code (< 1500 lines of C, Lex and Yacc code). A non Lex,Yacc implementation would use little more code; OpenSpec is very easy to parse.
PKOS can be used as an library to develop other tools and converters. We highlight the fundamental functionality provided by PKOS:

- read_xml reads a XML document and parses the OpenSpec expressions. The OpenSpec expressions are stored, group under there type, e.g. sort, declaration, ... together with the attributes. (The parser and kernel can easily be extended to store the general, non OpenSpec elements of the XML document too if needed).
- Export a kernel data structure as a string in OpenSpec format
- Manipulate the kernel data structure.

We developed a sort, syntax checker which verifies sorts and signatures of expressions as an and on to PKOS. PKOS is written in C using FLEX and BISON (yacc). The source code for PKOS can be obtained from our web-page [OpenSpec].

6.2 Prototype Extractor-Converter

The prototype extractor-converter is a simple FLEX, BISON (yacc) based tool that allows to extract the open spec elements form an XML file and can be used as a base to develop converters and tools of OpenSpec Documents. With very little programming we can for example create an OpenSpec to PVS converter. The needed files can be obtained from my OpenSpec web-page [OpenSpec].

6.3 Typesetting, XSL, XSLT

XSLT is a language for transforming XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. XSL specifies the styling of a XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary. As described in section 4, we developed a default XSL style file which allows to transform OpenSpec documents into HTML documents for viewing with browsers. In a strict sense, we do not use XSL, only XSLT and transform to HTML. Since XSL is a recent standard (first quarter 1999) not many web browsers do support XSL at the moment (even the number is increasing). At the moment (June 1999) Microsoft Internet Explorer 5.0 is the only (non alpha or beta) XML-XSL browser we could find. Unfortunately if does not implement the complete XSL and XSLT standard. There exist several Java based applications, which can transform XML documents according to a XSL file to Java applets which then can be used with most browsers. We used the Lotus XSL Processor [Lotus] which again uses the IBM XML Parser [IBM]. We present parts of your XSL file in appendix B. The complete XSL file will be available on [OpenSpec].
We like to point out that the specifications for XSL and XSLT are very resent at the moment, we found that the XSL processors to not behave exactly as specified. In any case of doubt we implemented according to the XSL specification and accepted that a particular processor did not perform the task as specified.

Conclusion

We propose the standard OpenSpec a general formal language based on XML which allows to represent formal and informal software specifications. We are convinced that as verification tools will support OpenSpec, it could strongly impact the way formal and semi formal software specifications are used, verified, presented and exchanged.

Appendix A

There exist several XML editors which accept a DTD and verify XML documents accordingly. We give a DTD for OpenSpec. Note that we use the element

```
<!-- DTD for OpenSpec   M. v. Mohrenschildt  31.5.1999  -->

<!ELEMENT OS_SORT  (OS_NAME,OS_EXP)>
<!ELEMENT OS_DECL  (OS_NAME,OS_EXP)>
<!ELEMENT OS_DEF   (OS_NAME,OS_EXP)>
<!ELEMENT OS_THEO  (OS_NAME,OS_EXP)>
<!ELEMENT OS_TAB   (OS_NAME,OS_GRID,OS_HEAD+,OS_INFO*)>
<!ELEMENT OS_REL   (OS_NAME,OS_EXP)>
<!ELEMENT OS_CODE  (#PCDATA)>

<!ELEMENT OS_NAME  (#PCDATA)>

<!-- Expressions -->

<!ELEMENT & alg    (OS_SYMB|OS_CONST|OS_CONS|OS_BIND
                   |OS_APP|OS_TAB)>

<!ELEMENT OS_EXP   (alg)>
<!ELEMENT OS_CONS  (alg*))>
<!ELEMENT OS_BIND  (OS_EXP,OS_NAME+,(alg)*)>
<!ELEMENT OS_APP   (OS_NAME,(alg)*)>
<!ELEMENT OS_SYMB  (#PCDATA)*)>
<!ELEMENT OS_CONST (#PCDATA)*)>

<!-- Tables  -->

<!ELEMENT  OS_GRID (OS_EXP|OS_CONSTRUCT)*>
<!ELEMENT  OS_HEAD (OS_EXP|OS_CONSTRUCT)*>
<!ELEMENT  OS_INFO (OS_EXP|#PCDATA)>
```

Appendix B

We present a part of th XSL type-sheet which transforms OpenSpec documents into HTML documents. The complete XSL file is not presented since of space

reasons put on be obtained form [OpenSpec]. We like to point out that this XSL document can easily be changes so that is output an ASCII representation or other output such as PVS.

```
<!--   APP stuff  -->
<xsl:template match="OS_APP/*">
<xsl:if test="..[not(@pos='infix')]">
   <xsl:if test=".[first-of-any()]">
        <xsl:apply-templates/>(
  </xsl:if>
  <xsl:if test=".[not(last-of-any()) and not(first-of-any())]">
     <xsl:apply-templates/>,
  </xsl:if>
  <xsl:if test=".[last-of-any()]">
        <xsl:apply-templates/>)
  </xsl:if>
</xsl:if>
<xsl:if test="..[@pos='infix']">
  <xsl:if test=".[not(last-of-any()) and not(first-of-any())]">
     <xsl:apply-templates/>
     <xsl:value-of select="../OS_NAME"/>
  </xsl:if>
  <xsl:if test=".[last-of-any()]">
     <xsl:apply-templates/>
  </xsl:if>
</xsl:if>
</xsl:template>



<xsl:template match="OS_CONS"> [<xsl:apply-templates/>] </xsl:template>
<xsl:template match="OS_CONS/*">
    <xsl:apply-templates/>
    <xsl:if test=".[not(last-of-any())]">,
    </xsl:if>
</xsl:template>
```

Appendix C

We give the EBNF grammar of the ASCII-parser implemented by our tool: (terminals are quoted with (') nonterminals are capital).

```
START ::= SORT | DECL | FUNC | EXP | ATT

SORT  ::=  'elesort' NAME                          // symbolic sort
          |'sort' NAME '::' NAME                   // subsort
          |'sort' NAME ':: {' (NAME,)+ '}'         //  finite sort
```

```
          |'sort' NAME ':: INTEGER              //  integer sort
          |'sort' NAME '::  REAL                //   real sort


ATT   ::=  NAME 'attribute' EXP


FUNC  ::=  'func' NAME '::'  NAME+ '->' NAME  IN?     //signature
          |'func' NAME ':: [' NAME '] ->' NAME IN?   //sequence

IN    ::=  : 'infix'

DECL  ::=  NAME '::' NAME

EXP   ::=   NAME
          | INT
          | REL
          | NAME '(' (EXP (',' EXP)*)? ')'
          | EXP NAME EXP                        //infix function
          | '(' EXP ')'


NAME  ::= [+-*/!@$#%^&*/<>] |  [a-zA-Z][a-zA-Z0-9_]*
INT   ::= [0-9]+
REL   ::= [0-9]*'.'[0-9]+
```

References

[xml95] "The XML specification" *www.w3c.org*

[xsl95] "The XSL specification" *www.w3c.org*

[gti96] I. Volnocheck, M. v. Mohrenschildt (1996), "Gti: Grand Table Interface", *CRL Report 96.*

[Par95] D. L. Parnas, (1995), "Tabular Representation of Relations", *CRL Report No. 260 1995*

[Par94.1] D. L. Parnas, (1994), "Mathematical Description and Specification of Software", *Proceedings of IFIP World, Congress 1994, Volume1 pp.270-277.*

[Par94.2] D. L. Parnas, (1994), "Inspection safety critical software using function tables", *Proceedings of IFIP World Congress 1994, Volume I.*

[Par94.3] D. L. Parnas, J. Madey, M. Iglewski, (1994), "Precise documentation of well-structured programs", *IEEE trans. Software Engineering, vol. 20, no. 12 pp 948-976, (1994).*

[Hen80] K.L. Heniger, (1980), "Specifying software requirements for complex systems", *IEEE Transactions on Software Engineering SE-6, 2-13.*

[Jan95.2] R. Janicki, D. L. Parnas, J. Zucker, (1995), "Tabular Representations in Rational Documents", *CRL Report No. 313 1995* .

[Heit96] C. L. Heitmeyer, R. D. Jeffords, B. G. Labaw, (1996), "Automated consistency checking of requirements specifications", *ACM trans. Software Engineering and Methodology, col5. No. 3, pp 231-261, 1996.*

[OpenSpec] OpenSpec Home Page `http://ece.mcmaster.ca/ mohrens/openspec/openspec.html`.

[PVS] PVS

[OpenMath] OpenMath Consortium (24-969), The OpenMath Standard, `http://www.openmath.org/`

[Sax] SAXON, XML tools `http://home.iclweb.com/icl2/mhkay/saxon.html`

[IBM] IBM XML tools: `http://www.alphaworks.ibm.com/`

[Lotus] LotusXSL, `http://www.alphaworks.ibm.com/`

[DIN79] German Industrial Norm on Tables, (1979) "Decision Tables", DIN 66 241

[Abr97] R. Abraham, (1997), "Evaluating Generalized Tabular Expressions in Software Documentation", *CRL Report CRL 346.*

[CSS] Cascading Style Sheets, *http://www.w3.org/Style/CSS/.*

[Amaya] Amaya, Editor/Browser for OpenMath `http://www.w3.org/Amaya/`