Adventures in Formal Methods at W3C: Using Z Notation to Specify WSDL 2.0

Arthur Ryman, IBM

Motivation

- I develop Web tools at IBM (WebSphere, Rational, Eclipse)
- A key attribute of Web services is interoperability between heterogeneous systems (e.g. J2EE, .NET, PHP, …)
- Our initial tool development experience with SOAP 1.1 and WSDL 1.1 exposed many problems that could be traced to specification defects
- The Web Services Interoperability (WS-I) Basic Profile 1.0 listed around 100 corrections, clarifications, and restrictions for WSDL 1.1
- My hope in joining the W3C Web Services Description Working Group was to help produce a high quality new spec

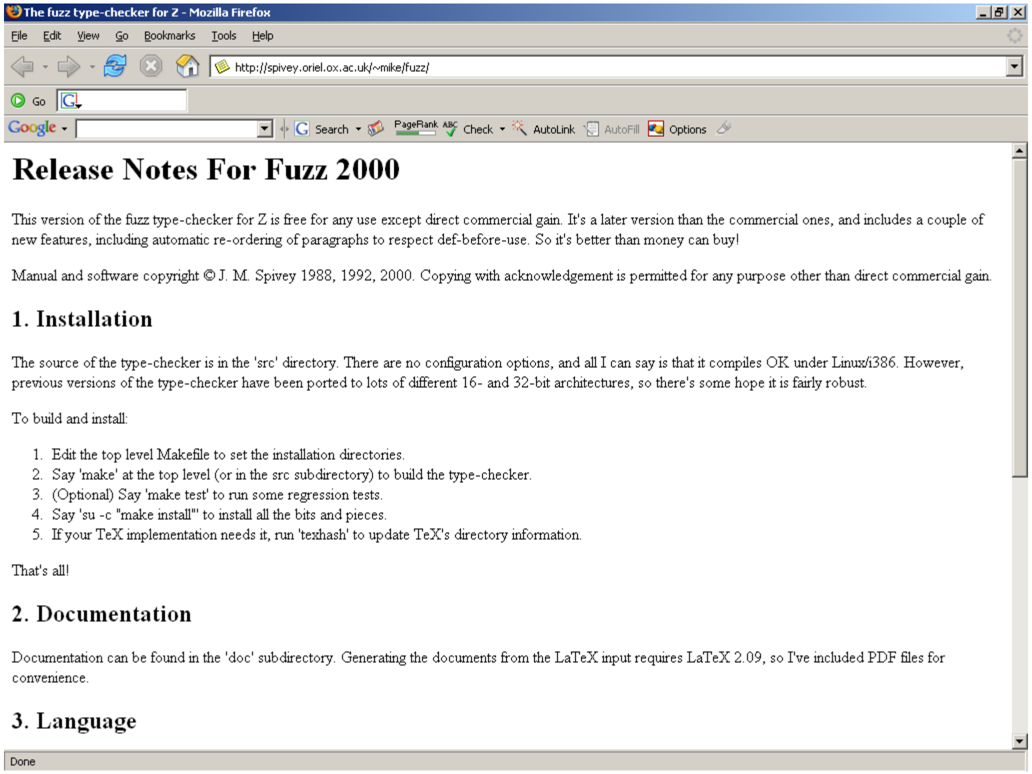
How Z Notation Got Introduced

- The working group decided to write the spec in terms of an abstract, informal, “Component Model” which was inspired by the XML Infoset and Schema specs
- The spec was getting long and I had little confidence in its overall consistency
- The Component Model looked like it could be easily expressed in Z Notation, a venerable formal specification technique that I learned, but abandoned, many years ago
- During a vacation break, I translated the spec into Z Notation, found a dozen problems, and then shared the results with the Working Group
- The Working Group “chartered” me include Z Notation in the spec

What is Z Notation?

- Z Notation is a formal specification technique based on fairly standard mathematical notation, and taught in the UK (many text books are available)
- It is based on Typed Set Theory which avoids certain technical difficulties, e.g. the Russell Paradox, and has the added benefit that it can be efficiently typed-checked
- It is designed to be added as a notation in specification documents as a way to complement the prose
- The most popular implementation is based on LaTeX
- There is a freely available type checker called Fuzz 2000 by Mike Spivey

Fuzz 2000 Web site


The fuzz type-checker for Z - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://spivey.oriel.ox.ac.uk/~mike/fuzz/
Go
Google
Search
PageRank
Check
AutoLink
AutoFill
Options
Done

Release Notes For Fuzz 2000

This version of the fuzz type-checker for Z is free for any use except direct commercial gain. It's a later version than the commercial ones, and includes a couple of new features, including automatic re-ordering of paragraphs to respect def-before-use. So it's better than money can buy!

Manual and software copyright © J. M. Spivey 1988, 1992, 2000. Copying with acknowledgement is permitted for any purpose other than direct commercial gain.

1. Installation

The source of the type-checker is in the 'src' directory. There are no configuration options, and all I can say is that it compiles OK under Linux/i386. However, previous versions of the type-checker have been ported to lots of different 16- and 32-bit architectures, so there's some hope it is fairly robust.

To build and install:

1. Edit the top level Makefile to set the installation directories.
2. Say 'make' at the top level (or in the src subdirectory) to build the type-checker.
3. (Optional) Say 'make test' to run some regression tests.
4. Say 'su -c "make install"' to install all the bits and pieces.
5. If your TeX implementation needs it, run 'texhash' to update TeX's directory information.

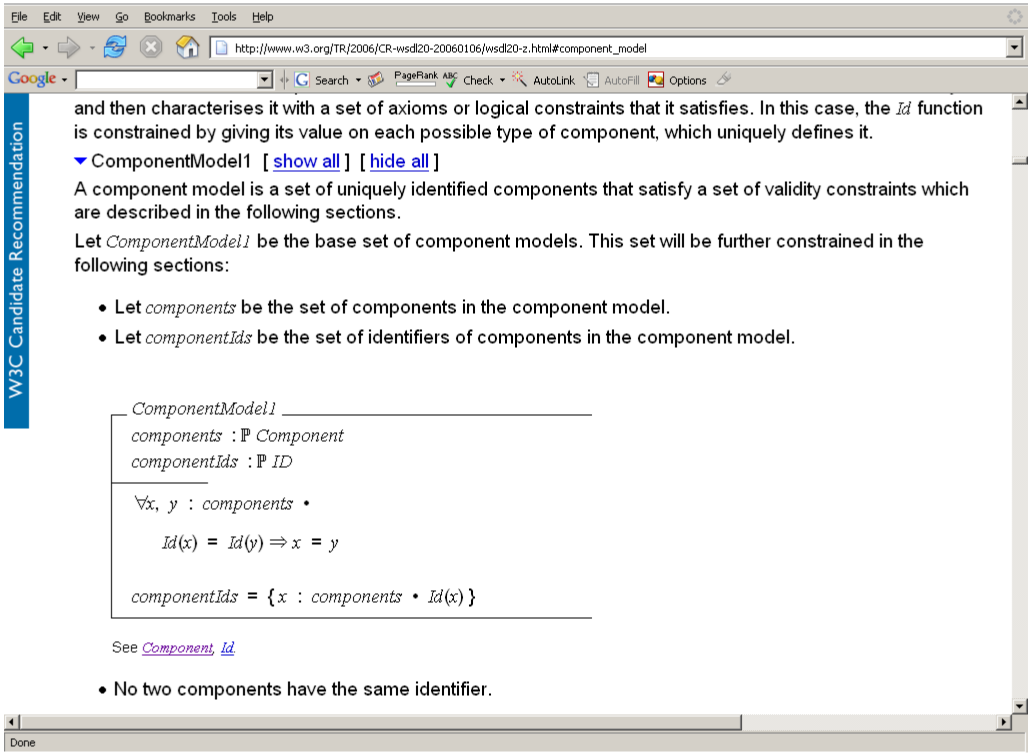
That's all!

2. Documentation

Documentation can be found in the 'doc' subdirectory. Generating the documents from the LaTeX input requires LaTeX 2.09, so I've included PDF files for convenience.

3. Language

Example of Z Notation in WSDL 2.0 Spec


File Edit View Go Bookmarks Tools Help
http://www.w3.org/TR/2006/CR-wsdl20-20060106/wsdl20-z.html#component_model
Google
Search
PageRank
Check
AutoLink
AutoFill
Options
W3C Candidate Recommendation
Done

and then characterises it with a set of axioms or logical constraints that it satisfies. In this case, the *Id* function is constrained by giving its value on each possible type of component, which uniquely defines it.

▼ ComponentModel1 [show all] [hide all]

A component model is a set of uniquely identified components that satisfy a set of validity constraints which are described in the following sections.

Let *ComponentModel1* be the base set of component models. This set will be further constrained in the following sections:

- Let *components* be the set of components in the component model.
- Let *componentIds* be the set of identifiers of components in the component model.

ComponentModel1

$components : \mathbb{P}\ Component$

$componentIds : \mathbb{P}\ ID$

$$\forall x, y : components \bullet$$

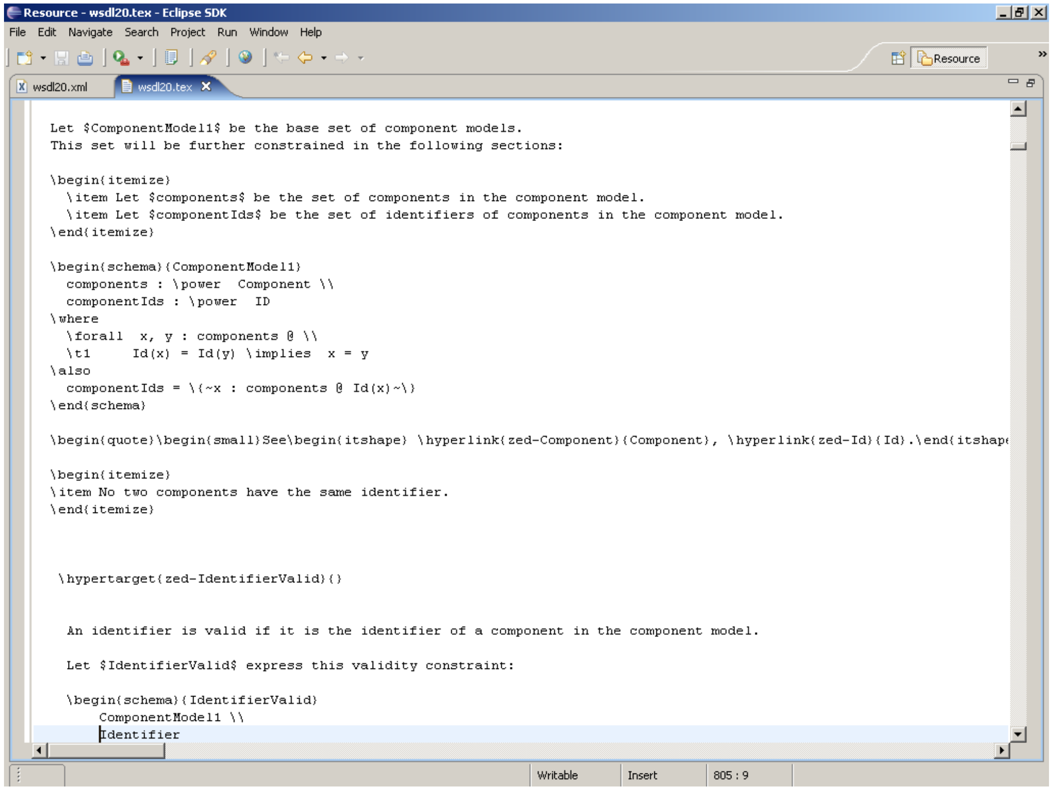
$$Id(x) = Id(y) \Rightarrow x = y$$

$$componentIds = \{ x : components \bullet Id(x) \}$$

See *Component*, *Id*.

- No two components have the same identifier.

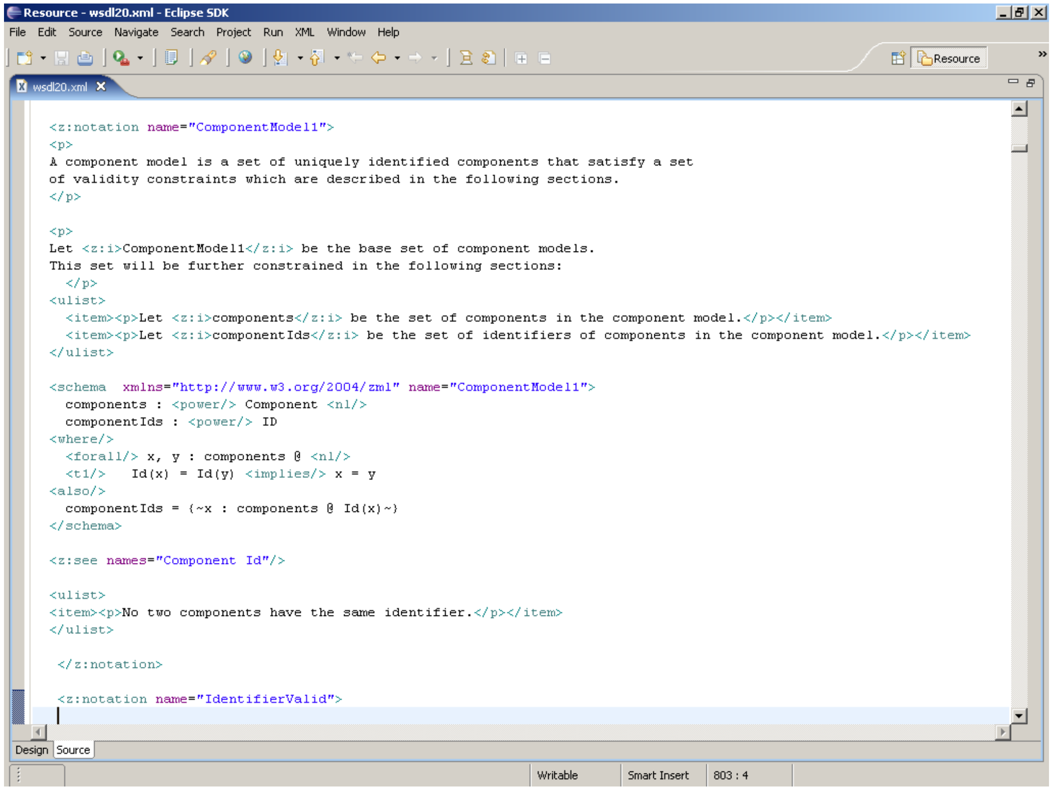
Example of Z Notation LaTeX Source



Resource - wsdl20.tex - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Resource
wsdl20.xml
wsdl20.tex

```
Let $ComponentModel1$ be the base set of component models.
This set will be further constrained in the following sections:

\begin{itemize}
  \item Let $components$ be the set of components in the component model.
  \item Let $componentIds$ be the set of identifiers of components in the component model.
\end{itemize}

\begin{schema}{ComponentModel1}
  components : \power  Component \\
  componentIds : \power  ID
\where
  \forall  x, y : components @ \\
  \t1     Id(x) = Id(y) \implies  x = y
\also
  componentIds = \{~x : components @ Id(x)~\}
\end{schema}

\begin{quote}\begin{small}See\begin{itshape} \hyperlink{zed-Component}{Component}, \hyperlink{zed-Id}{Id}.\end{itshape

\begin{itemize}
\item No two components have the same identifier.
\end{itemize}



 \hypertarget{zed-IdentifierValid}{}


  An identifier is valid if it is the identifier of a component in the component model.

  Let $IdentifierValid$ express this validity constraint:

  \begin{schema}{IdentifierValid}
      ComponentModel1 \\
      Identifier
```

Example of Z Notation XMLSPEC Source



Resource - wsdl20.xml - Eclipse SDK
File Edit Source Navigate Search Project Run XML Window Help
Resource
wsdl20.xml
Design Source
Writable
Smart Insert
803 : 4

```
<z:notation name="ComponentModel1">
<p>
A component model is a set of uniquely identified components that satisfy a set
of validity constraints which are described in the following sections.
</p>

<p>
Let <z:i>ComponentModel1</z:i> be the base set of component models.
This set will be further constrained in the following sections:
  </p>
<ulist>
  <item><p>Let <z:i>components</z:i> be the set of components in the component model.</p></item>
  <item><p>Let <z:i>componentIds</z:i> be the set of identifiers of components in the component model.</p></item>
</ulist>

<schema  xmlns="http://www.w3.org/2004/zml" name="ComponentModel1">
  components : <power/> Component <nl/>
  componentIds : <power/> ID
<where/>
  <forall/> x, y : components @ <nl/>
  <t1/>    Id(x) = Id(y) <implies/> x = y
<also/>
  componentIds = {~x : components @ Id(x)~}
</schema>

<z:see names="Component Id"/>

<ulist>
<item><p>No two components have the same identifier.</p></item>
</ulist>

 </z:notation>

 <z:notation name="IdentifierValid">
```

Benefits – The Translation Effect

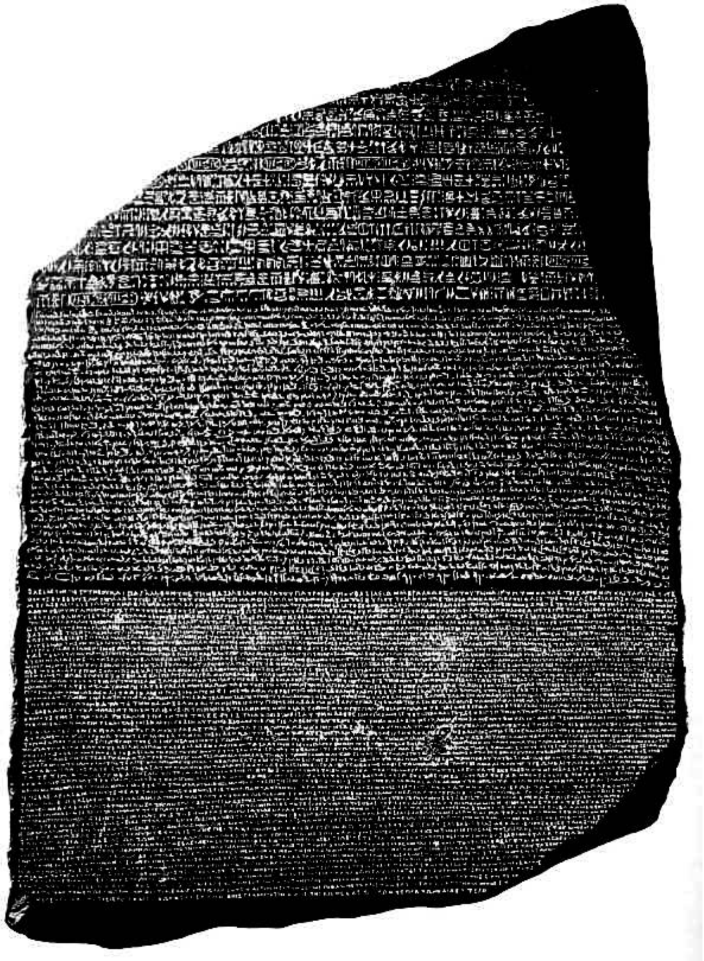
- Writing Z Notation forces you to read the prose carefully, which is a great way to review it and find errors
- You would actually get this benefit by translating the prose into any other language, e.g. French or Larch
- Having two or more alternate representations of the same information can help people understand it better, c.f. the Rosetta Stone

Rosetta Stone

"The decree, voted by the priests of Egypt at Memphis **[WSDL WG]**, is repeated in two languages– Egyptian **[prose]** (in both hieroglyphic and demotic scripts) and Greek **[Z]** --and records the good deeds of Ptolemy and the honours proposed for the twelve year old King.

Through the Rosetta Stone and other similar bilingual inscriptions scholars **[developers]** were able to decipher the hieroglyphs **[specs]** of ancient Egypt **[W3C]**."

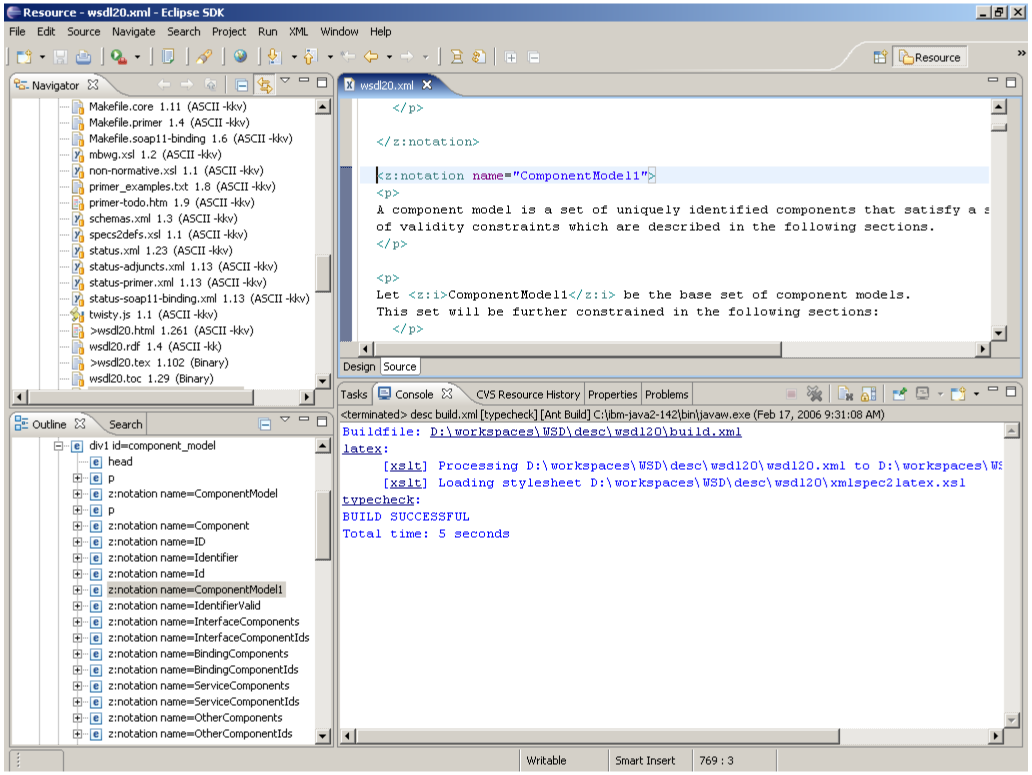
- British Museum



Benefits – Global Consistency Checking

- Specs tend to get long
- Unfortunately, humans are bad at global consistency checking, e.g.
  - Does the use of a term on p. 137 match its definition way back on p. 42?
- Fortunately, humans are good at local consistency checking, e.g.
  - Does the Z Notation on p. 42 match the prose on p. 42?
- Computer programs are good at global consistency checking, e.g.
  - Does the Z Notation term used on p. 137 match its Z Notation definition on p. 42?

Type checking Z Notation in WSDL 2.0 Spec



```
Makefile.core  1.11  (ASCII -kkv)
Makefile.primer  1.4  (ASCII -kkv)
Makefile.soap11-binding  1.6  (ASCII -kkv)
mbwg.xsl  1.2  (ASCII -kkv)
non-normative.xsl  1.1  (ASCII -kkv)
primer_examples.txt  1.8  (ASCII -kkv)
primer-todo.htm  1.9  (ASCII -kkv)
schemas.xml  1.3  (ASCII -kkv)
specs2defs.xsl  1.1  (ASCII -kkv)
status.xml  1.23  (ASCII -kkv)
status-adjuncts.xml  1.13  (ASCII -kkv)
status-primer.xml  1.13  (ASCII -kkv)
status-soap11-binding.xml  1.13  (ASCII -kkv)
twisty.js  1.1  (ASCII -kkv)
>wsdl20.html  1.261  (ASCII -kkv)
wsdl20.rdf  1.4  (ASCII -kk)
>wsdl20.tex  1.102  (Binary)
wsdl20.toc  1.29  (Binary)
```

```
  </p>

</z:notation>

<z:notation name="ComponentModel1">
<p>
A component model is a set of uniquely identified components that satisfy a s
of validity constraints which are described in the following sections.
</p>

<p>
Let <z:i>ComponentModel1</z:i> be the base set of component models.
This set will be further constrained in the following sections:
  </p>
```

```
div1 id=component_model
  head
  p
  z:notation name=ComponentModel
  p
  z:notation name=Component
  z:notation name=ID
  z:notation name=Identifier
  z:notation name=Id
  z:notation name=ComponentModel1
  z:notation name=IdentifierValid
  z:notation name=InterfaceComponents
  z:notation name=InterfaceComponentIds
  z:notation name=BindingComponents
  z:notation name=BindingComponentIds
  z:notation name=ServiceComponents
  z:notation name=ServiceComponentIds
  z:notation name=OtherComponents
  z:notation name=OtherComponentIds
```

```
<terminated> desc build.xml [typecheck] [Ant Build] C:\ibm-java2-142\bin\javaw.exe (Feb 17, 2006 9:31:08 AM)
Buildfile: D:\workspaces\WSD\desc\wsdl20\build.xml
latex:
     [xslt] Processing D:\workspaces\WSD\desc\wsdl20\wsdl20.xml to D:\workspaces\WS
     [xslt] Loading stylesheet D:\workspaces\WSD\desc\wsdl20\xmlspec2latex.xsl
typecheck:
BUILD SUCCESSFUL
Total time: 5 seconds
```

Obstacles

- Z Notation is not widely known either within the Working Group or the intended audience of the spec
  - Lack of Working Group expertise to review the Z Notation rendered it as Informative (Non-Normative)
- W3C uses XMLSPEC (not LaTeX) and defines a Character Model for math symbols (not supported by Internet Explorer)
  - XMLSPEC markup was defined
  - XSLT transforms markup to XHTML (Character Model and Internet Explorer), and LaTeX (for Fuzz 2000) were developed (see WG CVS)
- No existing library of formal specs for XML, XML Infoset, XML Schema, SOAP, HTTP and other standards used by WSDL 2.0
  - Only formalized Component Model and not Bindings

Z Nirvana

- Formal specification becomes a QA Best Practice
- Standard markup and toolset available for use by Working Groups
  - MathML support
  - Go beyond type checking (use theorem proving technology to check semantics)
  - Maybe even generate reference implementations
- Standard library of formal specifications available for existing W3C Recommendations
  - Normative status
  - And also for IETF, OASIS, etc. specs