Formal Specification of the WSDL 2.0 Component Model

Arthur Ryman

July 15, 2004

Abstract

This document is a formal specification of the WSDL 2.0 Component Model. Its purpose is to help improve the quality of the informal specification and to provide a set of precise test assertions that can be checked by WSDL validators.

1 Introduction

This document is a formal specification of the WSDL 2.0 Component Model. The main goal of this document is to ensure that the text of the W3C specification for WSDL 2.0 is complete, precise, and unambiguous. By translating the English statements of the W3C specification into formal mathematical statements, problems in the English text will be exposed and corrected.

The formal specification language, Z Notation, is used for the mathematical statements. One advantage of Z Notation is that it can be automatically type checked, a process which eliminates many common specification errors, thereby improving the quality of the specification.

Another goal of this document is to serve as a Test Assertion Document (TAD) which will be used by software that validates WSDL documents. A test assertion is a constraint or rule that a WSDL document must satisfy in order to be compliant with the specification. Test assertions can be manually translated into an executable programming language such as Java or C#.

2 Preliminary Definitions

This section contains some definitions of data types that are used in the component model.

2.1 AbsoluteURI

Certain URIs, such as network addresses, must be absolute.

Let $AbsoluteURI$ be the set of all absolute URIs:

$$AbsoluteURI : \mathbb{P}\ URI$$

2.2 NamespaceURI

XML namespaces are used extensively.

Let $NamespaceURI$ be the set of all URIs that are valid XML namespace names:

$$NamespaceURI : \mathbb{P}\ URI$$

2.3 QName

QNames are used as component identifiers and elsewhere.

Let $QName$ be the set of all XML QNames:

- Let $localName$ be the local name.
- Let $namespaceName$ be the namespace name.

QName

$localName : NCName$
$namespaceName : NamespaceURI$

2.4 OptionalQName

Some QName-valued component properties, such as the interface associated with a binding, are optional.

Let *OptionalQName* be the set of all optional QNames:

$$OptionalQName ::= noValueQName \mid valueQName\langle\langle QName\rangle\rangle$$

2.5 QNamed

In the component model, many components have a name and target namespace that define a QName.

Let *QNamed* be the set of all name and target namespace properties and their derived QName:

- Let *name* be an NCName.
- Let *targetNamespace* be a namespace name.
- Let *qName* be the derived QName.

QNamed

$name : NCName$
$targetNamespace : NamespaceURI$
$qName : QName$

$qName.localName = name$
$qName.namespaceName = targetNamespace$

- The local name is the name.
- The namespace name is the target namespace.

3 Component Model

The WSDL 2.0 specification describes the abstract component model for a Web service description as a set of components with properties.

- Definitions
  - Element Declaration
  - Interface
    - Interface Fault
    - Interface Operation
      - Message Reference
      - Fault Reference
      - Feature
      - Property
    - Feature
    - Property
  - Binding
    - Binding Fault
    - Binding Operation
      - Binding Message Reference
      - Feature
      - Property
    - Feature
    - Property
  - Service
    - Endpoint

Figure 1: The WSDL 2.0 Component Containment Tree

3.1 The Component Containment Tree

Each component has a specific type and is either the root Definitions component, or is contained within another component. Figure 1 illustrates the containment tree for all the component types.

Note that Feature and Property components appear at several places in the tree.

Each component can be uniquely identified by a subset of its properties within the scope of its parent component. The type of the identifier depends on the type of the component, but it is usually either a URI, a QName or an NCName. Components reference other components using their identifiers.

3.2 The Element Declaration Component

See: The Definitions Component.

An element declaration component corresponds to a declaration, in some

type system, of a data type that is like an XML Schema global element declaration.

An element declaration defines the local name, namespace name, children, and attributes of an element information item.

3.2.1 ElementDeclaration

Since WSDL 2.0 is extensible with respect to type systems, it is not possible to parameterize Element Declaration components with a set of simple properties, so we'll introduce a basic set of element declarations and assume that there is a way to map any element declaration from any suitable type system to this set. We can think of this set as the set of canonical serializations of element declarations.

Let *ElementDeclaration* be the set of all element declarations that define elements that have a QName:

$$[ElementDeclaration]$$

3.2.2 elementQName

Let *elementQName* map element declarations to element QNames:

$$elementQName : ElementDeclaration \longrightarrow QName$$

3.2.3 ElementDeclarationComponent

Let *ElementDeclarationComponent* be the set of Element Declaration components:

- Let *qName* be the QName of the element defined by the declaration.
- Let *elementDeclaration* be the element declaration.

$$\begin{array}{|l} \textit{ElementDeclarationComponent} \\ \hline qName : QName \\ elementDeclaration : ElementDeclaration \\ \hline qName = elementQName\ elementDeclaration \\ \hline \end{array}$$

- The declaration defines an element with the given QName.

The QName uniquely identifies the Element Declaration component within its parent component.

3.3 The Feature Component

See: The Feature Component.

A Feature is a quality, such as reliability or security, that can be associated with a message exchange. A Feature component associates a Feature with part of a Web service description.

3.3.1 FeatureComponent

Let *FeatureComponent* be the set of Feature components:

- Let *name* be the absolute URI that identifies the feature.
- Let *required* be a Boolean that indicates if the requester must use the Feature.

FeatureComponent

$$name : AbsoluteURI$$
$$required : Boolean$$

The name uniquely identifies the Feature component within its parent component.

3.4 The Property Component

See: The Property Component.

A Property is a name-value pair that typically influences the behavior of a Feature. A Property Component associates a set of values with a Property.

3.4.1 XsAnyType

A Property component may optionally define a value for the property. The value may be of any XML Schema type.

Let *XsAnyType* be the set of all values that belong to the XML Schema type xs:anyType:

$$XsAnyType : \mathbb{P}\ String$$

3.4.2 OptionalXsAnyType

Let *OptionalXsAnyType* be the set of optional *XsAnyType* values:

$$OptionalXsAnyType ::= noValueXsAnyType \mid valueXsAnyType\langle\langle XsAnyType\rangle\rangle$$

3.4.3 PropertyComponent

Let *PropertyComponent* be the set of Property components:

- Let *name* be the absolute URI that identifies the Property.
- Let *required* be a Boolean that indicate if the requester must use this Property.
- Let *valueConstraint* be an optional QName that refers to an XML Schema type definition which defines the permissible set of values of the Property.

- Let *value* by an optional value of XML Schema type xs:anyType which defines the only permissible value of the Property.

$$
\begin{array}{|l}
\textit{PropertyComponent} \\
\hline
\textit{name} : \textit{AbsoluteURI} \\
\textit{required} : \textit{Boolean} \\
\textit{valueConstraint} : \textit{OptionalQName} \\
\textit{value} : \textit{OptionalXsAnyType} \\
\hline
\textit{valueConstraint} = \textit{noValueQName} \Leftrightarrow \\
\quad \textit{value} \neq \textit{noValueXsAnyType} \\
\hline
\end{array}
$$

- Either the value constraint or the value is specified, but not both.

The name uniquely identifies the Property Component within its parent component.

3.4.4 FeaturesAndProperties

As noted above, Feature and Property components appear in several places in the component model. It is therefore convenient to group them as follows.

Let *FeaturesAndProperties* be the set all collections of Feature and Property components:

- Let *features* be a set of Feature components.
- Let *properties* be a set of Property components.

$$
\begin{array}{|l}
\textit{FeaturesAndProperties} \\
\hline
\textit{features} : \mathbb{F}\ \textit{FeatureComponent} \\
\textit{properties} : \mathbb{F}\ \textit{PropertyComponent} \\
\hline
\forall\, x, y : \textit{features} \bullet x.\textit{name} = y.\textit{name} \Rightarrow x = y \\
\forall\, x, y : \textit{properties} \bullet x.\textit{name} = y.\textit{name} \Rightarrow x = y \\
\hline
\end{array}
$$

- Each Feature component is unqiuely identified by its name.
- Each Property component is unqiuely identified by its name.

3.5 The Interface Fault Component

See: The Interface Fault Component.

An Interface Fault is an exception message that can be exchanged through an Interface.

3.5.1 InterfaceFaultComponent

Let *InterfaceFaultComponent* be the set of Interface Fault Components:

- Let *name* be the Interface Fault name.
- Let *element* be the QName of an Element Declaration component.

InterfaceFaultComponent
name : *NCName*
element : *QName*

The name uniquely identifies an Interface Fault component within its parent component.

3.6 The Message Reference Component

See: The Message Reference Component.

A Message Reference associates a content model with a message exchanged by an operation.

3.6.1 Direction

Let *Direction* be the set of all message directions, i.e. #in or #out:

Direction ::=
inDirection |
outDirection

3.6.2 MessageContentModel

Let *MessageContentModel* be the set of message content models, i.e. #none, #any, or #element:

MessageContentModel ::=
noneMessageContentModel |
anyMessageContentModel |
elementMessageContentModel

3.6.3 MessageReferenceComponent

Let *MessageReferenceComponent* be the set of Message Reference components:

- Let *messageLabel* be the NCName message label as defined in the message exchange pattern for the operation.
- Let *direction* be the message direction.

- Let *messageContentModel* be the message content model. If the message content model is #element then the element declaration must also be specififed.
- Let *element* be the optional QName of the Element Declaration component in the case that the message content model is #element.

MessageReferenceComponent

$$
\begin{array}{l}
messageLabel : NCName \\
direction : Direction \\
messageContentModel : MessageContentModel \\
element : OptionalQName \\
\hline
element \neq noValueQName \Leftrightarrow \\
\quad messageContentModel = elementMessageContentModel
\end{array}
$$

- The Element Declaration component QName is present exactly when the message content model is #element.

The message label uniquely identifies the Message Reference component within its parent component.

3.7 The Fault Reference Component

See: The Fault Reference Component.

A Fault Reference component associates an Interface Fault component with an operation.

3.7.1 FaultReferenceComponent

Let *FaultReferenceComponent* be the set of Fault Reference components:

- Let *messageLabel* be the NCName message label of the fault as defined by the message exchange pattern of the operation.
- Let *direction* be the direction of the fault.
- Let *faultReference* be the QName of the Interface Fault component that describes the content of the fault.

FaultReferenceComponent

$$
\begin{array}{l}
messageLabel : NCName \\
direction : Direction \\
faultReference : QName
\end{array}
$$

The message label uniquely identifies the Fault Reference component within its parent component.

3.8 The Interface Operation Component

See: The Interface Operation Component.

An Interface Operation component defines an operation of an interface.

3.8.1 InterfaceOperationComponent

Let *InterfaceOperationComponent* be the set of Interface Operation components:

- Let *messageExchangePattern* be the message exchange pattern.
- Let *messageReferences* be a set of Message Reference components.
- Let *faultReferences* be a set of Fault Reference components.
- Let *style* be the style of the operation.
- Let *safety* be an optional Boolean that indicates the safety of the operation.

InterfaceOperationComponent

$$
\begin{array}{l}
QNamed \\
messageExchangePattern : AbsoluteURI \\
messageReferences : \mathbb{F}\ MessageReferenceComponent \\
faultReferences : \mathbb{F}\ FaultReferenceComponent \\
style : AbsoluteURI \\
safety : OptionalBoolean \\
FeaturesAndProperties \\
\hline
\forall x, y : messageReferences \bullet \\
\quad x.messageLabel = y.messageLabel \Rightarrow x = y \\
\forall x, y : faultReferences \bullet \\
\quad x.messageLabel = y.messageLabel \Rightarrow x = y
\end{array}
$$

- Each Message Reference component is uniquely identified by its message label.
- Each Fault Reference component is uniquely identified by its message label.

The name and target namespace form the QName of the Interface Operation component which uniquely identifies it within its parent component.

3.9 The Interface Component

See: The Interface Component.

An Interface component collects a set of operations and faults that define the abstract interface of a Web service.

3.9.1 InterfaceComponent

Let *InterfaceComponent* be the set of all Interface components:

- Let *extendedInterfaces* be a set of references to Interface components that this interface extends.
- Let *faults* be a set of Interface Fault components.
- Let *operations* be a set if Interface Operation components.

InterfaceComponent

$$
\begin{array}{l}
QNamed \\
extendedInterfaces : \mathbb{F}\ QName \\
faults : \mathbb{F}\ InterfaceFaultComponent \\
operations : \mathbb{F}\ InterfaceOperationComponent \\
FeaturesAndProperties \\
\hline
\forall x, y : faults \bullet x.name = y.name \Rightarrow x = y \\
\forall x, y : operations \bullet x.name = y.name \Rightarrow x = y \\
\forall x : operations \bullet x.targetNamespace = targetNamespace
\end{array}
$$

- Each Interface Fault component is uniquely identified by its name.
- Each Interface Operation component is uniquely identified by its name.
- Each Interface Operation component has the same target namespace as this component.

The name and target namespace for the QName of the Interface component which uniquely identifies it within its parent component.

There are other rules that an interface must satisfy, such as an interface must not extend itself either directly or indirectly, and an interface must not redefine an operation, but these will be described later.

3.10 The Binding Fault Component

See: The Binding Fault Component.

A Binding Fault component binds an Interface Fault to a concrete protocol.

3.10.1 BindingFaultComponent

Let *BindingFaultComponent* be the set of Binding Fault components:

- Let *faultReference* be a QName reference to an Interface Fault that belongs to the interface bound by this component.

BindingFaultComponent

faultReference : *QName*

The fault reference uniquely identifies the Binding Fault component within its parent component.

3.11 The Binding Message Reference Component

See: The Binding Message Reference Component.

A Binding Message Reference component binds a Message Reference component to a concrete protocol.

3.11.1 BindingMessageReferenceComponent

Let *BindingMessageReferenceComponent* be the set of all Binding Message Reference Components:

- Let *messageLabel* be the message label that references a Message Reference component that belongs to the interface bound by this component.
- Let *direction* be the message direction.

BindingMessageReferenceComponent

messageLabel : *NCName*

direction : *Direction*

The message label uniquely identifies the Binding Message Reference component within its parent.

3.12 The Binding Operation Component

See: The Binding Operation Component.

A Binding Operation component binds an Interface Operation component to a concrete protocol.

3.12.1 BindingOperationComponent

Let *BindingOperationComponent* be the set of all Binding Operation components:

- Let *operationReference* be the QName reference to the Interface Operation component that is bound by this component.
- Let *messageReferences* be a set of Binding Message Reference components.

BindingOperationComponent

$operationReference : QName$
$messageReferences : \mathbb{F}\ BindingMessageReferenceComponent$
$FeaturesAndProperties$

$\forall\, x, y : messageReferences \bullet$
$\quad x.messageLabel = y.messageLabel \Rightarrow x = y$

- Each Binding Message Reference component is uniquely identified by its message label.

A Binding Operation component is uniquely identified by its operation reference within its parent component.

3.13 The Binding Component

See: The Binding Component.

A Binding component binds an Interface component to a concrete protocol.

3.13.1 BindingComponent

Let *BindingComponent* be the set of all binding components:

- Let *interface* be the optional QName that references the Interface bound by this binding. If the QName is specified then the binding applies only to the referenced interface. Otherwise it applies to any interface.
- Let *type* identify the concrete protocol of the binding.
- Let *faults* be a set of Binding Fault components.
- Let *operations* be a set of Binding Operation components.

BindingComponent

$QNamed$
$interface : OptionalQName$
$type : AbsoluteURI$
$faults : \mathbb{F}\ BindingFaultComponent$
$operations : \mathbb{F}\ BindingOperationComponent$
$FeaturesAndProperties$

$interface = noValueQName \Rightarrow$
$\quad faults = \varnothing \land operations = \varnothing$

$\forall\, x, y : faults \bullet$
$\quad x.faultReference = y.faultReference \Rightarrow x = y$

$\forall\, x, y : operations \bullet$
$\quad x.operationReference = y.operationReference \Rightarrow x = y$

- If no interface is specified then no binding faults or operations may be defined.
- Each Binding Fault component is uniquely identified by its fault reference.
- Each Binding Operation component is uniquely identified by its operation reference.

The name and target namespace of a Binding component define its QName which uniquely identifies it within its parent component.

3.14 The Endpoint Component

See: The Endpoint Component.

An Endpoint component defines a concrete endpoint for accessing a Web service.

3.14.1 EndpointComponent

Let *EndpointComponent* be the set of all Endpoint components:

- Let *name* be the endpoint name.
- Let *binding* be a QName reference to a Binding component.
- Let *address* be an optional absolute URI that is the network address of the endpoint.

$$
\begin{array}{|l}
\textit{EndpointComponent} \\
\hline
name : NCName \\
binding : QName \\
address : OptionalURI \\
\hline
address = noValueURI \lor \\
\quad (\exists\, uri : AbsoluteURI \bullet address = valueURI\ uri) \\
\hline
\end{array}
$$

- If the address is present it must be an absolute URI.

The name uniquely identifies an Endpoint component within its parent component.

3.15 The Service Component

See: The Service Component.

A Service component defines a set of endpoints for accessing a Web service that implements an interface.

3.15.1 ServiceComponent

Let *ServiceComponent* be the set of all Service components:

- Let *interface* be a QName reference to an Interface component.
- Let *endpoints* be a set of Endpoint components.

$$
\begin{array}{|l}
\textit{ServiceComponent} \\
\hline
\textit{QNamed} \\
\textit{interface} : \textit{QName} \\
\textit{endpoints} : \mathbb{F}\ \textit{EndpointComponent} \\
\hline
\forall\, x, y : \textit{endpoints} \bullet \\
\quad x.\textit{name} = y.\textit{name} \Rightarrow x = y \\
\hline
\end{array}
$$

- Each endpoint has a unique name.

The name and target namespace form a QName that uniquely identifies a Service component within its parent component.

3.16 The Definitions Component

See: The Definitions Component.

A Definitions component collects together Interface, Binding, Service, and Element Declaration components that describe Web services.

3.16.1 DefinitionsComponent

Let *DefinitionsComponent* be the set of all Definitions components:

- Let *interfaces* be a set of Interface components.
- Let *bindings* be a set of Binding components.
- Let *services* be a set of Service components.
- Let *elementDeclarations* be a set of Element Declaration components.

DefinitionsComponent

$interfaces : \mathbb{F}\ InterfaceComponent$
$bindings : \mathbb{F}\ BindingComponent$
$services : \mathbb{F}\ ServiceComponent$
$elementDeclarations : \mathbb{F}\ ElementDeclarationComponent$

$\forall x, y : interfaces \bullet$
$\quad x.qName = y.qName \Rightarrow x = y$

$\forall x, y : bindings \bullet$
$\quad x.qName = y.qName \Rightarrow x = y$

$\forall x, y : services \bullet$
$\quad x.qName = y.qName \Rightarrow x = y$

$\forall x, y : elementDeclarations \bullet$
$\quad x.qName = y.qName \Rightarrow x = y$

- Each Interface component is uniquely identified by its QName.
- Each Binding component is uniquely identified by its QName.
- Each Service component is uniquely identified by its QName.
- Each Element Declaration component is uniquely identified by its QName.

3.17 Intercomponent References

There are additional rules that the component model must satisfy. Many components refer to other components. The component being refered to must exist. The following list summarizes the reference rules:

3.17.1 InterfaceFaultElement

The Interface Fault component element property refers to an Element Declaration component.

InterfaceFaultElement

DefinitionsComponent

$\forall i : interfaces \bullet$
$\quad \forall f : i.faults \bullet$
$\quad\quad \exists e : elementDeclarations \bullet$
$\quad\quad\quad f.element = e.qName$

3.17.2 MessageReferenceElement

The Message Reference component element property refers to an Element Declaration component.

MessageReferenceElement

DefinitionsComponent

$$
\begin{array}{l}
\forall\, i : interfaces \bullet \\
\quad \forall\, o : i.operations \bullet \\
\quad\quad \forall\, mr : o.messageReferences \mid \\
\quad\quad\quad mr.element \neq noValueQName \bullet \\
\quad\quad\quad \exists\, e : elementDeclarations \bullet \\
\quad\quad\quad\quad mr.element = valueQName\ e.qName
\end{array}
$$

3.17.3 FaultReferenceFaultReference

The Fault Reference component faultReference property refers to an Interface Fault component.

FaultReferenceFaultReference

DefinitionsComponent

$$
\begin{array}{l}
\forall\, i : interfaces \bullet \\
\quad \forall\, o : i.operations \bullet \\
\quad\quad \forall\, fr : o.faultReferences \bullet \\
\quad\quad\quad \exists\, f : i.faults; \\
\quad\quad\quad\quad qn : QName \mid \\
\quad\quad\quad\quad qn.localName = f.name \land \\
\quad\quad\quad\quad qn.namespaceName = i.targetNamespace \bullet \\
\quad\quad\quad\quad qn = fr.faultReference
\end{array}
$$

Note that this rule isn't correct because it assumes that the Interface Fault is defined by the Interface. However, an Interface can extend other Interfaces and it seems reasonable to allow a reference to an Interface Fault defined by one the of the extended Interfaces.

3.17.4 InterfaceExtendedInterfaces

The Interface component extendedInterfaces property refers to Interface components.

InterfaceExtendedInterfaces

DefinitionsComponent

$$
\begin{array}{l}
\forall\, i : interfaces \bullet \\
\quad \forall\, qn : i.extendedInterfaces \bullet \\
\quad\quad \exists\, xi : interfaces \bullet \\
\quad\quad\quad qn = xi.qName
\end{array}
$$

3.17.5 BindingFaultFaultReference

The Binding Fault component faultReference property refers to an Interface Fault component.

BindingFaultFaultReference

DefinitionsComponent

$$
\begin{array}{l}
\forall\, b : bindings;\ i : interfaces \mid \\
\quad b.interface = valueQName\, i.qName \bullet \\
\quad \forall\, bf : b.faults \bullet \\
\quad\quad \exists\, if : i.faults;\ qn : QName \mid \\
\quad\quad\quad if.name = qn.localName \land \\
\quad\quad\quad i.targetNamespace = qn.namespaceName \bullet \\
\quad\quad\quad qn = bf.faultReference
\end{array}
$$

Again, this is not quite right because we are not checking for Interface Faults defined in the Interfaces that are extended by this Interface.

3.17.6 BindingOperationOperationReference

The Binding Operation component operationReference property refers to an Interface Operation component.

BindingOperationOperationReference

DefinitionsComponent

$$
\begin{array}{l}
\forall\, b : bindings;\ i : interfaces \mid \\
\quad b.interface = valueQName\, i.qName \bullet \\
\quad \forall\, bo : b.operations \bullet \\
\quad\quad \exists\, io : i.operations \bullet \\
\quad\quad\quad io.qName = bo.operationReference
\end{array}
$$

Again, this is not quite right because we are not checking for Interface Operations defined in the Interfaces that are extended by this Interface.

3.17.7 BindingInterface

The Binding component interface property refers to an Interface component.

BindingInterface

DefinitionsComponent

$$
\begin{array}{l}
\forall\, b : bindings \mid \\
\quad b.interface \neq noValueQName \bullet \\
\quad \exists\, i : interfaces \bullet \\
\quad\quad b.interface = valueQName\, i.qName
\end{array}
$$

3.17.8 EndpointBinding

The Endpoint component binding property refers to a Binding component.

EndpointBinding

DefinitionsComponent

$$\forall s : services \bullet \forall e : s.endpoints \bullet \exists b : bindings \bullet b.qName = e.binding$$

3.17.9 ServiceInterface

The Service component interface property refers to an Interface component.

ServiceInterface

DefinitionsComponent

$$\forall s : services \bullet \exists i : interfaces \bullet i.qName = s.interface$$

4 Comments on the WSDL Specification

4.1 Inconsistency Between Interface Fault and Interface Operation

Interface Fault has no targetNamespace property, but Interface Operation does. I think for consistency, Interface Fault should have a target Namespace.

Interface Fault components are referenced by QName from Fault Reference components so the Interface Fault component should have a QName with the constraint that the target namespace equals that of its parent Interface component.

4.2 Use namespaceName instead of targetNamespace

Several components, such as Service, Interface, have a property named targetNamespace. However, I think this is a misnomer because the property is really the namespace name of the component. The term target namespace makes sense in the context of an XML document that is defining components, but after the components have been defined, the target namespace of the document becomes the namespace name of the defined component. A better name would be namespace name.

4.3 Use localName instead of name

Similarly, several components have a name property, but this is really the local name of the QName used to identify the component.

4.4 Use QName instead of NCName + URI

In the case where the properties of a component define its QName, it would be simpler to use a single property, qName of type QName, instead of the NCName and URI pair.

4.5 Inconsistent and Umbiguous Language for Component Defintions versus References

The specification does not use consistent language to describe contained component definitions and sometimes uses the same language to describe references to components defined elsewhere. The specification should adopt clear and consistent language to described contained component definitions and component references.

When a property contains the definition of a component of type T, say "a T component". When a property contains a reference of type R to a component of type T, say "an R that references a T component".

For example, consider the {interfaces} property of the Definitions component. Instead of "A set of named interface definitions" say "A set of Interface components."

Compare this with the {interface} property of the Service component. Instead of "An Interface component" say "A QName that references an Interface component."

4.6 Use Description instead of Definitions

Since we are the Web Service Description WG and are defining Web Service Description Language, should the root component be Description instead of Definitions?

4.7 Resolution of the Value Constraint QName in the Property Component

The Property component may contain an option value constraint that references an XML Schema type definition. However, the component model only contains element declarations. Does this imply that there are no validity rules that require the type definition to exist somewhere? Is it valid for the constraint to have a QName that does not refer to a type definition? Must the type be defined in XML Schema?

4.8 No Mapping from Infoset to the {required} property of the Property Component

There is no entry in the mapping table for the {required} property. This looks like an omission.

4.9 Incorrect Language in Mapping from Infoset to {value constraint} property of Propery Component

The text has two otherwise clause, but there is only one alternative. The constraint element is either in the infoset or it isn't. If the constrain is absent then a value must be present so the first otherwise clause applies. The second otherwise clause should be dropped.

5 To Do

5.1 Extension Components

Add extension components, e.g. for bindings.

5.2 Simplify Handling of Optional Types

Define a generic function to create the set consisting of the empty set and the singleton set for a given input set.

This is like the empty set generic.

5.3 Remove Infoset Dependency

The component model is independent of the infoset. Move the basic XML definitions to a separate common document.

5.4 Interface Extension

Specify the rules.

5.5 Binding Agreement with Interface

Specify rules that guarantee the binding matches its interface.

5.6 Operation Style

Document the RPC style rules.

5.7 Message Exchange Patterns

Document them and the rules that guarantee consistency with the component model.

5.8 Consistency Between Endpoint and Service

An endpoint must use a binding that agrees with the interface of the service.