Preface 1

1.1 Cosubmitting Companies and Supporters

The following companies are pleased to co-submit the XML Metadata Interchange specification (hereafter referred to as XMI) in response to the Object Analysis & Design Task Force RFP3 - Stream based Model Interchange Format (SMIF):

- Unisys Corporation
- International Business Machines Corporation
- Cooperative Research Centre for Distributed Systems Technology (DSTC)
- Oracle Corporation
- Platinum Technologies, Inc.
- Fujitsu
- Softeam

Dr. Kerry Raymond

John Clark
Platinum Technology Inc.
E-m m4(il: cla)20(r)-18(k@)-15(platinum)-122.com

Jun Ginbayashi
Fujitsu
E-m m4(il: gin@)-15(toky)-22(o.s)14(e(m4(.f)7(u)-22(jits)14(u.co.jp)]TJ 0 -2.5122 TD [(P)
E-m5(5(m4(r)nke5(m4(ilgendo)3(r)v)76o)3(f)E)-8 1doay

Chapter 7 XML DTD Production

Specifies the production rules for DTDs, as part of the encoding of MOF based metamodels into the proposed format.

Proof of Concept 2

2.1 Copyright Waiver

In the event that this specification is adopted by OMG, the XMI cosubmitters grant to the OMG, a non-exclusive, royalty-free, paid-up, worldwide license to copy and distribute this specification document and to modify the document and distribute copies of the modified version. For more detailed information, see the disclaimer on the inside of the cover page of this submission.

2.2 Proof of Concept

XMI cosubmitters and supporters have extensive experience in the areas of metadata repositories, modeling tools, CORBA and the related problems of interchange of

Response to RFP Requirements 3

3.1 Mandatory Requirements

3.1.1 Required Meta-metamodel

Proposals shall use the MOF as its meta-metamodel.

0 Tw (3)Tj ET 1 g 184.98 30.66 208.56 -17.28 re f BT /F5 1 Tf

0.049 Tw [Sd 4(Ii)-8Fg specification

The focus of the XMI proposal is on current and emerging OMG metadata standards. The submitters believe that integration of XMI and STEP EXPRESS to address EDI

Design Rationale 4

4.1 Design Overview

The MOF Model

The “MOF Model” is the MOF’s built-in meta-metamodel. One can think of it as the “abstract language” for defining MOF metamodels. This is analogous to the way that the UML metamodel is an abstract language for defining UML models. While the MOF and UML are designed for two different kinds of modelling (i.e. metadata versus object modelling), the MOF Model and the core of the UML metamodel are closely

between elements in a MOF metamodel (M2-level entities) and the CORBA objects that represent metadata (M1-level entities).

- A Class in the metamodel maps onto an IDL interface for metadata objects and a metadata class proxy. These interfaces support the Operations, Attributes and References defined in the me9(o)6(d)-19(e)23(l, an)6(d)-19(in)6(th)6(e cas)17(e o)-19(f)10(cl
  acts as a holder for the proxies for the Classes and Associations contained by the Package, and therefore serves to define a logical extent for metadata associations, classifier level attributes and the like.

  The IDL that is produced by the mapping is defined in precise detail so that different vendor implementations of the MOFas generate compatible repository interfaces

DTDs also define the attribut6s that can be included in an element using an ATTLIST.

There are three major APIs to XML. DOM, the Document Object Model, is a language-neutral interface to XML documents for creation and reading data and

4.4.4 Knowledge of Metamodels

Design Decision: An XMI document consumer or producer needs

Usage Scenarios 5

5.1 Purpose

This section describes some of the problems that IT users and vendors face today and illustrates how XMI helps to address these problems.

5.2 Comb ning tools in a heterogeneous env ronment

Implementing an effective and efficient IT solution for an enterprise requires a dgaile dununt-22(rta)

information content. A second advantage of including the definitions in the stream is that the scope of information that can be transferred is not fixed; it can be extended with new definitions as more tools are integrated to exchange information.

5.3 Co-operating with common metamodel definitions

The extent of the information that can be exchanged between two tools is limited by how much of the information can be understood by both tools. If they both share the same metamodel (the definition of the structure and meaning of the information being used), all of the information transferred can be understood and used. However, gaining consensus on a totally shared meta model is a difficult task even within a single company. It is more likely that a subset of the meta model can be shared with each tool

XMI DTD Design Principles 6

6.1 Purpose

any particular numbering scheme will be used. The

```
<!ELEMENT XMI.extension ANY >
```

6.5.12 XMI.difference

This XML element holds XML elements representing differences to base data. Users may use it within the content part of an XMI file or in a separate XMI.difference

```
<!ELEMENT XMI.replace ANY  >
<!ATTLIST XMI.replace
        %XMI.element.att;
        %XMI.link.att;
```

```
<!ATTLIST XMI.CorbaTcArray
        xmi.tcLength CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcObjRef EMPTY >
<!ATTLIST XMI.CorbaTcObjRef
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcEnum (XMI.CorbaTcEnumLabel)
<!ATTLIST XMI.CorbaTcEnum
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcEnumLabel EMPTY >
<!ATTLIST XMI.baTcEnumLabel
        xmi.tcName CDATA #REQUIRED
>

<!ELEMENT XMI.baTcUnibr (XMIorbaTypeCode,XMIany) >
<!ATTLIST XMI.CorbaTcUnibr
        xmi.tcName CDATA #REQUIRED
>

<!ELEMENT XMI.baTcUniXMICorbaTypeCode,XMICorbaTcUnibr*)
>
<!ATTLIST XMI.CorbaTcUni
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.baTcExcept (XMICorbaTcFiel*1>
<!ATTLIST XMI.baTcExcept
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcStiPTY >
<!ATTLIST XMI.CorbaTcStig
        xmi.tcLength CDATA #REQUIRED
>

<!ELEMENT XMI.CorbaTcWstri>
```

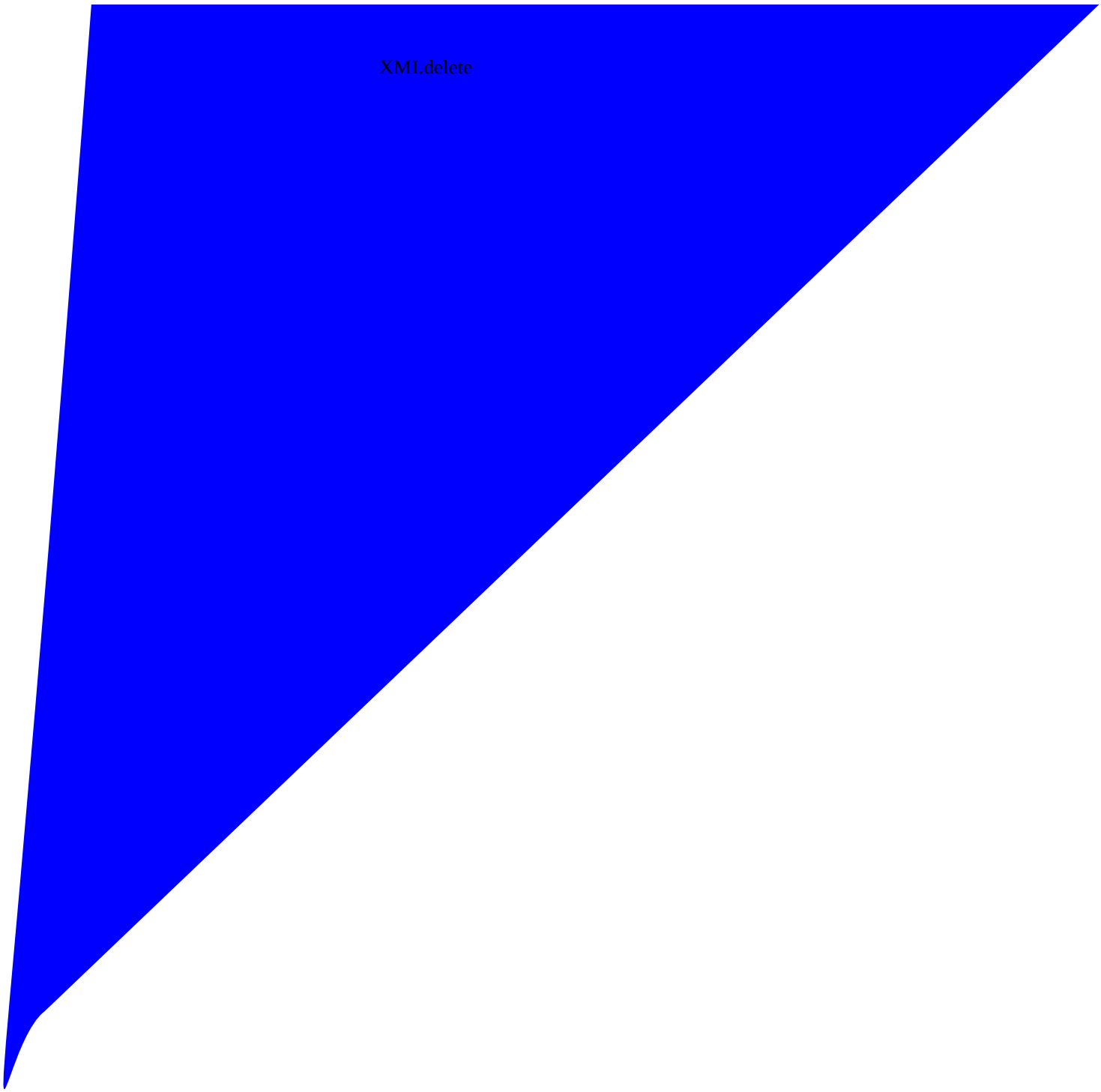
in an XML DTD, and how inheritance between metamodel classes is handled. It uses

information models are developed, they will frequently need to be interchanged before they are complete.

6.8.2 Linking

XLinks

When specifying a XLink, the **xml:link**
"simple" for unidirectional links. The “href” attribute may be used to specify an optional URI and XPointer that identify an XML element in another XML document. The **href**


XMI.delete

```
<XMI.content>
    <Package xmi.id="ppp" xmi.label="p1">
    </Package>
</XMI.content>
```

Next, the XMI.add:

```
<XMI.content>
    <Package xmi.id="ppp" xmi.label="p1">
        <Class xmi.label="c2">
        </Class>
    </Package>
</XMI.content>
```

Finally, the XMI.replace:

```
<XMI.content>
    <Package xmi.id="ppp" xmi.label="p2">
        <Class xmi.label="c2">
        </Class>
    </Package>
</XMI.content>
```

3.10 Document exchange with multiple tools

0 Tw (6)Tj ET 1 g 184.98 30.66 208.56 -17.28 re f BT /F3 1 Tf

```
</Class>
<Class xmi.label="c2" xmi.uuid="X012345678">
</Class>
```

4. The model is imported into Tool1. Tool1 assigns xmi.extenderID "ijklmnop" to "c2" and a new class "c3" is created with xmi.extenderID "qrstuvwxyz".

5. The model is merged back to XMI:

```
<Class xmi.label="c1" xmi.uuid="abcdefgh">
    <XMI.extension xmi.extender="Tool2" xmi.extenderID="JKLMNOP>
</Class>
<Class xmi.label="c2" xmi.uuid="X012345678">
    <XMI.extension xmi.extender="Tool1" xmi.extenderID="ijklmnop"/>
</Class>
<Class xmi.label="c3" xmi.uuid="qrstuvwxyz">
</Class>
```

6. A third closed tool, Tool3, adds its ids:

```
<Class xmi.label="c1" xmi.uuid="abcdefgh">
    <XMI.extension xmi.extender="Tool2" xmi.extenderID="JKLMNOP>
    <XMI.extension xmi.extender="Tool3" xmi.extenderID="s1234"/>
</Class>
<Class xmi.label="c2" xmi.uuid="X012345678">
    <XMI.extension xmi.extender="Tool1" xmi.extenderID="ijklmnop"/>
    <XMI.extension xmi.extender="Tool3" xmi.extenderID="s5678"/>
</Class>
<Class xmi.label="c3" xmi.uuid="qrstuvwxyz">
    <XMI.extension xmi.extender="Tool3" xmi.extenderID="s90ab"/>
</Class>
```

7. An open tool imports and modifies the file. There are no changes because the **xmi.uuids** are used by the tool.

6.11 UML DTD

Appendix A contains an automatically generated DTD generated that represents the UML metamodel. This DTD generally follows the specification of the above section on representing metamodel information. By examining this DTD, you can ga(DT)-8 a better understanding of the types of metamodel information that can be represented in an XML DTD, and the information that cannot be specified.

XML DTD Production 7

7.1 Purpose

This section describes the rules for creating a DTD from a MOF-based metamodel. Each of the three types of DTDs defined by the rules in this section describes the XML text created by following the rules of Chapter 9, *XML Document Production on page 167*. These rules are derived from informal programs written to implement each of the three methods described in this chapter. A formal reference implementation of the DTD production rules, when and if 7t appears, would, in all likelihood, include a revision of these rules.

Conformance

Any conformance to the XMI specification is based on generated XML and not on any DTD format. A conforming implementation of the rules 7n this section may implement any or all of these rule sets or may use its own when generating a DTD for a metamodel.

Notation

Since DTD generation programs are not a conformance point of this specification, the rules are expressed as pseudocode rather than any specific programming language. The stylistic guidelines for the DTD generation can be found in Chapter 6, *XMI DTD Design Principles on page 49*.

To Generate a PackageDTD:

For Each **Class** of the **Package** Do
    For each **Attribute** of the **Class** Do
        If **isDerived** is **false** Then
            If the **scope** of the **Attribute** is **classifierLevel** Then
                Generate an AttributeElementDef (#4) for the **Attribute**
            End
        End
    End
End
For Each **Association** of the **Package** Do
    If **isDerived** is **false** Then
        If the **Association**

To Genera6e a ClassDTD:

```
For Each A66ribu6e of the Class Do
    If isDerived is false Then
        If scope is instanceLevel then
            Genera6e the A6tribu6eElemen6Def (#4) for the A66ribu6e
        End
    End
End
For Each Reference of the Class Do
    If the isDerived at6ribu6e of the associated
```

```
    <Generalization xmi.idref="X2"/>
</generalization>
```

To generate a ReferenceEMlementDef:

Set RefName := The qualified name of the **Reference**
Set cls := **Reference.referencedEnd.type** (which is constrained to be a **Class**)
Set m := GetReferenceMultiplicity(the **Reference**)
Set RefContents := '(' + GetClasses(cls, '') + ') + m

the Reference to the AssociationEnd as an exposedEnd. It also appears in the content models of the subclasses of this Class.

To generate a CompositionDTD:

Generate the CompositionElementDef (#8)

8. CompositionElementDef

The CompositionElementDef is the XML element generated for an Association which has a Reference whose aggregation is composite. It names the Reference and the Class

11. AssociationEndDef

7.2.2 Auxiliary functions

All of the auxiliary functions defined in this section are used in the Simple DTD rule

GetAttributes

GetContainedPackages

The GetContainedPackages sg10/20/1998 -92 u1(-1 m 53.94 718 30.4

GetUnreferencedAssociations

This auxiliary function gets all of the Associations of the Package (and its parent package. tkt tkr>reeferfenr(e)23e.

To Generate an AttributeElementDef:

Set AttribName := the qualified name of the **Attribute**.
If the **type** reference refers to a **DataType** Then
    If **DataType.typeCode** is **tk_Boolean** or **tk_enum** Then
        Set AttribContents := 'EMPTY'
        Set TypeName := the qualified name of the enumerated type or Boolean

To Generate a PackageElementDef

Set PkgName := the fully qualified name of the **Package**

7.3.2 Auxiliary functions

The following auxiliary functions are used in this rule set. They have a suffix of “2”,

OutputPropertiesEntityDef2

The OutputPropertiesEntityDef2 function is a recursive function that creates an Entity definition for the instance-level Attributes of a Class and then calls itself to generate those for all of the subclasses of the Class. This Entity definition consists of a listing of all of the instance-level Attributes for the Class. It is possible for the entity content to be empty; if so, the entity is not generated. This fact is remembered 10(r)o that the

OutputRefsEntityDef2

GetContainedClasses2

The GetContainedClasses2 function returns a string describing the Classes contained in

GetCompsEntities2

The GetCompsEntities2 function collects together a sequence of invocations of the CompsEntityDefs for the given Class and the Classes from whici (n1is)-11(der)7(i)0(ved)-22(.)21(
End

To Generate an AttributeElementDef:

Set AttribName := the qualified name of the **Attribute**.
If the **type** reference refers to a

entities summarize this information instead. The entity invocations do not appear if they would be empty.

To Generate a ClassElementDef:

```
Set ClassName := the qualified name of the Class
Set props := ''
If a properties entity was generated for this Class Then
    Set props := '%' + ClassName + 'Properties' + ";"
End
Set refs :=
If a References entity was generated for this Class Then
    Set refs :=  '|' + '%' + ClassName + 'Associations' + ';'
End
Set refs :=  '(' + ' XMI.extension' + '*' + refs + ')'

If a comps entity was generated for this Class Then
    Set com.007 ps := '%' + ClassNam.007 e + 'Compositions' + ';
End
Set coms2 := GetContainedClasses(t(t(he )]TJı˝/F6 1 Tfı˝8.64871 0 TDı˝0.01 Tcı˝[(Cl)-9(as)25(s)]TJı˝/

    lfg7(t)208((6˝[(Cl)18((a)26Con(at)20n(at)2s)2r)21=he)-20)2r tn(:)]TJ˝-2.027 -1.2432 TDı˝0.008 Twı˝+ ClaCon(a
End

        S
```

To Generate a PackageElementDef

Set PkgName := the fully qualified name of the **Package**

Compositions. The AssociationDTD defines elements for the two AssociationEnds of the Association..

To Generate an AssociationDTD:

Generate an AssociationEna6(aDn)-23(e6(aft)-7 (r
Assoiai-18(e)0ae

7.4.2 Auxiliary functions

OutputRefsEntityDef3

The OutputRefsEntityDef3 function is similar to OutputPropertiesEntityDef3, except that it produces a set of RefsEntitiesDefs instead of PropertiesEntityDefs.

```
Subroutine OutputRefsEntityDef3(in cls: Class, inout prevCls: String,
                                inout baseCls: String)
    If cls appears in prevCls, Then
        Return the empty string (’’)
    End
                                                    Class
```

OutputCompsEntityDef3

The OutputCompsEntityDef3 function is similar to OutputPropertiesEntityDef3, except that it produces a set of CompsEntitiesDefs instead of PropertiesEntityDefs.

```
Subroutine OutputCompsEntityDef3(in cltp2lutCl
inout baseCls: String)
```

Cs appears in prevClen

GetAllInstanceAttributes3

The GetAllInstanceAttributes3 function returns a string containing the name of the Properties entity of the parent Class of the 6(na)iven class plus all of th(na)e non-derived instance-level attributes of the Class itself.

In the case of multiple inheritance, this function invokes a multiple-inheritance management function to gets the Attributes from the parent Classes in the second (and any additional) set of parent Classes. These are between the parent Properties entity and the Attributes of the Class itself.

```
Function GetAllInstanceAttributes3(in cls : Class,  in baseCls: String) Returns String
    Set parentEntity := ”
    Set parentContents := ”
    For each Class referenced by cls.supertype Do
        If cls.supertype
```

GetParentReferences3

7.5 Fixed DTD elements

There are some elements of the DTD which are fixed, constituting a form of

```
<!ELEMENT XMI.model ANY >
<!ATTLIST XMI.model
              %XMI.link.att;
              xmi.name      CDATA #REQUIRED
              xmi.version   CDATA #IMPLIED >

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.metamodel identifies the metamodel(s) for the transferred -->
<!-- data                                                         -->
<!-- ____________________________________________________________ -->
<!ELEMENT XMI.metamodel ANY >
<!ATTLIST XMI.metamodel
              %XMI.link.att;
              xmi.name      CDATA #REQUIRED
              xmi.version   CDATA #IMPLIED >

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.metametamodel identifies the metametamodel(s) for the    -->
<!-- transferred data                                             -->
<!-- ____________________________________________________________ -->
<!ELEMENT XMI.metametamodel ANY >
<!ATTLIST XMI.metametamodel
              %XMI.link.att;
              xmi.name      CDATA #REQUIRED
              xmi.version   CDATA #IMPLIED >

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.content is the actual data being transferred             -->
<!-- ____________________________________________________________ -->
<!ELEMENT XMI.content ANY >

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.extensions contains data to transfer that does not conform -->
<!-- to the metamodel(s) in the header                            -->
<!-- ____________________________________________________________ -->
<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
              xmi.extender CDATA #REQUIRED >

<!-- ____________________________________________________________ -->
```

<!-- -->

```
<!ELEMENT XMI.CorbaTcShort EMPTY >
<!ELEMENT XMI.CorbaTcLong EMPTY >
<!ELEMENT XMI.CorbaTcUshort EMPTY >
<!ELEMENT XMI.CorbaTcUlong EMPTY >
<!ELEMENT XMI.CorbaTcFloat EMPTY >
<!ELEMENT XMI.CorbaTcDouble EMPTY >
<!ELEMENT XMI.CorbaTcBoolean EMPTY >
<!ELEMENT XMI.CorbaTcChar EMPTY >
<!ELEMENT XMI.CorbaTcWchar EMPTY >
<!ELEMENT XMI.CorbaTcOctet EMPTY >
<!ELEMENT XMI.CorbaTcAny EMPTY >
<!ELEMENT XMI.CorbaTcTypeCode EMPTY >
<!ELEMENT XMI.CorbaTcPrincipal EMPTY >
<!ELEMENT XMI.CorbaTcNull EMPTY >
<!ELEMENT XMI.CorbaTcVoid EMPTY >
<!ELEMENT XMI.CorbaTcLongLong EMPTY >
<!ELEMENT XMI.CorbaTcLongDouble EMPTY >
```

instances which compose the SimpleGraph metamodel are shown in Figure 8-2 on page 155 (with much detail omitted).

Since this metamodel is expressed via the MOF, its model instances can be represent.-21 in

At this point, all the values that make up the model have been written out as XML. The Net object is completed with the end tag:

```
</SimpleGraph.Net>
```

All this XML will be embedded i9 the standard XML element, as described later. Also,

```
        <SimpleGraph.Node xmi.id='a6' />
      </SimpleGraph.Node.targetNodes>
    </SimpleGraph.Node>
```

will not be represented in the generated document. Instead a href will be used, which can be resolved to navigate to a representation of the NodeZ object.

9.3.1 EBNF Productions

```
ContentsFromRoot(root : RefObject) : Sequence(string)
ContentsFromRoot(root) =
  Sequence{ '<XMI.content>',
            ObjectAsElement(root),
            OtherLinks(root),
            RequiredTypeDefinitions(),
            '</XMI.content>'
          }
```

9.4.1.2 OtherLinks

Object references provide a representation of links, when a Reference is defined i8 the

9.4.2.2 ClassAttributes

9.4.3 Object Productions

The rest of the expressions in this document are not specific to either the object containment or package extent productions. The object productions define expressions producing XML from objects.

9.4.3.1 ObjectAsElement

An object is represented as an element by producing an element start tag, then the object (and any objects it contains) as the contents of the element, followed by the element end tag.

The operation produces the name of the element from the Class instance in the metamodel defining the class of this object. The fully-qualified name, using a dot notation, is used to avoid any ambiguity in naming the Class. The element start tag

The ObjectContents operation produces XML to represent the contents of an object – its state (attributes and references). Three steps are required to produce XML from the input object: produce the XML for the object's non-derived, instance-level attribute values, then produce the XML for the object's non-derived, non-composite, non-component reference values, and finally produce the XML for the objects' component objects. From the object's class, all the Attributes are obtained, including inherited attributes. Among those, the -derived,, instance-level attr0(de)ibutes are select6(d,)-. Over that sequence, string representations of the values are produced, using the Attr0(de)ibuteAsElement operation. The value oper0(de)ation, from the MOF's RefObj6(d,)ct interface, provides the attribute v0(de)alue or values.

Then, among the classes' References, those which match the following crit6(rive)a are selected: not based upon a derived Association, not with a referencedEnd which is a

Because the interactions with the MOF are defined using the MOF's reflective interfaces, the values of object attributes, references and link ends are represented using the CORBA Any type, matching the return type of those interfaces' operations. The extract_Object operation is an operation defined for the CORBA Any type. It is used to convert the Any value to an object. The ObjectContents operation is used to define the contents of this element.

EmbeddedObject(

```
Sequence {
   '<XMI.unionDiscrim>',
   ObjRefOrDataValue(ExtractUnionDiscrim(value),
```

```
            else
              if value.type().kind() =  tk_ulonglong then
                value.extract_ulonglong()
              else
                if value.type().kind() =  tk_octet then
                  value.extract_octet()
                else
                  -- undefined
                endif
              endif
            endif
          endif
        endif
      endif
    endif)
```

9.4.7.9 *RealValue*

Each of the real types a20(e)-0(a)1 handl0(a)1d by this operation.

The type of the value determines which extraction operation to use.

```
RealValue(value : Any) : s. (br)29(ing)]TJ  ET  145.14 390.18 165.12 0.72 re  f  BT  /F
```

```
    Sequence { ' xmi.id="', id, '"' }
  else
    Sequence { }
  endif),
 '>',
(if tc.kind() = tk_alias then
    TcAlias(tc)
  else
    if tc.kind() = tk_struct then
      TcStruct(tc)
    else
      if tc.kind() = tk_sequence then
        TcSequence(tc)
      else
        if tcf  9.1143 0 T
```

9.4.8.4 TcAlias

This production generates a representation of an alias type.

9

9.4.9.9 *ExtractUnionField*

```
                    self.objectInventory ← self.objectInventory->append(obj)
                    self.objectIds ← self.objectIds.append(result)
```

```
IdOfTypeDefinition(tc : TypeCode) : Sequence(string)
IdOfTypeDefinition(tc) =
if Sequence{ 1..(self.tcInventory->size) }->select(i |
       self.tcInventory->at(i) = tc)->isEmpty then
    NewTcId(tc)
  else
    self.tcIds.at(Sequence{ 1..(self.tcInventory->size) }->select(i |
       self.tcInventory->at(i) = tc)->first)
  endif
```

11.3.2 XMI DTD Compliance

References

Glossary

This glossary defines the terms that are used to describe the XMI specification. The

any A CORBA primitive data type. A strongly typed “universal union” type that can

builtin type

instance level In MOF metamodels and UML models, this label indicates that the labelled feature is 0 TD 0 -1.26

OCL

state The state of an object is the group of values that constitute its properties at a given point in time.

static In C++ or Java, a static attribute or a static member function is shared by all instances of a class. Synonym: *classifier level*.

static type checking Contrast: *dynamic type checking*.

static typing Contrast: *dynamic typing*.

strong typing A characteristic of a computational system that type failures are guaranteed not to occur.

stereotype A new type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel.

$$TC(v_0, m) \cong \{ v \in V : (v = v_0$$

Index

A

L