Using XML to Build Consistency Rules for Distributed Specifications

Andrea Zisman
City University
Department of Computing
Northampton Square
London EC1V 0HB
+44 (0)20 74778346
a.zisman@soi.city.ac.uk

Wolfgang Emmerich Anthony Finkelstein
University College London
Department of Computer Science
Gower Street
London WC1E 6BT
+44 (0)20 76794413
{w.emmerich | a.finkelstein}@cs.ucl.ac.uk

ABSTRACT

The work presented in this paper is part of a large programme of research aimed at supporting consistency management of distributed documents on the World Wide Web. We describe an approach for specifying consistency rules for distributed partial specifications with overlapping contents. The approach is based on expressing consistency rules using XML and XPointer. We present a classification for different types of consistency rules, related to various types of inconsistencies and show how to express these consistency rules using our approach. We also briefly describe a rule editor to support the specification of the consistency rules.

Keywords: Inconsistency, consistency rules, XML, XPointer.

1. INTRODUCTION

The size and complexity of industrial software systems require collaboration and co-ordination of physically distributed teams of people during systems development. Each person or group of people has their own perspective and understanding of the system. These different perspectives are based on the skills, responsibilities, knowledge and expertise of the people concerned. The result is multiple distributed *partial* specifications in a variety of forms produced using heterogeneous applications, word processors, specialised applications, software engineering tools, and similar.

Inevitably, the heterogeneity of the specifications and the diversity of stakeholders and development participants results in inconsistencies among the distributed partial specifications [18]. However as development proceeds there is a requirement for consistency.

Consistency management is a multifaceted and complex activity that is fundamental to the success of software development. Different approaches have been proposed to manage consistency [11][17][21][25][28]. This research has identified a strong need for mechanisms, techniques and tools that aid in *detecting, identifying* and *handling* inconsistencies among distributed partial specifications.

We are interested in identifying inconsistencies among partial specifications where the documents in which these specification fragments are represented exhibit Internet-scale distribution. We assume that the different specifications are constructed in languages that are themselves specified using the eXtensible Markup Language (XML) [5]. This assumption is, we believe, entirely reasonable since XML is evolving as the standard format for exchanging data among heterogeneous, distributed computer systems. Many tools are already using XML to represent information internally or as a standard export format. Examples are found in the next generation of IBM's VisualAge tool set, Microsoft Office 2000, ADLS, and Rational Rose 2000[1].

The work presented in this paper is part of a large programme of research to support consistency management of distributed documents on the World Wide Web [10]. An important issue when managing inconsistencies of distributed documents is to describe the relationships that are required to hold among the documents. The relationships are expressed through *consistency rules*. In [10] we proposed an approach to identify and detect inconsistencies among distributed documents, based on pre-defined consistency rules. The elements related by

[1] A detailed description of XML and its applications is beyond the scope of this paper.

these rules are associated through hyperlinks, named *consistency links.*

The work presented in this paper complements the work proposed in [10]. It describes a way of expressing the consistency rules by using a *consistency rule syntax*, based on XML [5] and XPointer [16]. We present a classification for the different types of consistency rules that we can represent with our syntax. In addition, we analyse the expressiveness of the consistency rule syntax and describe how to apply this syntax to complex software engineering notations. We also present a consistency rule editor to support the specification of consistency rules based on the syntax. Note that the notion of *consistency* used here does not correspond to the logical concept of consistency (for a discussion of this see [8] and [18]).

The remainder of the paper is structured as follows. Section 2 briefly describes our approach to managing consistency between distributed documents. Section 3 presents the syntax that is used in our approach to describe the consistency rules. Section 4 describes a classification for different types of consistency rules that can be expressed by using the syntax defined in section 3. Section 5 addresses the consistency rule editor. Section 6 presents some related work. Finally, section 7 summarizes the results, discusses some evaluation aspects, and suggests directions for future work.

2. MANAGING CONSISTENCY AMONG DISTRIBUTED DOCUMENTS

In [10] we proposed an approach to identify inconsistencies among documents with Internet-scale distribution. The approach is simple and lightweight, relying largely on the judicious use of standards. It consists of using *consistency rules* to identify related data, represented as elements, in distributed documents. The related elements are associated through hyperlinks, named *consistency links*. The links are traversed to identify either consistent or inconsistent elements. The approach is built on existing Internet technologies, in particular the eXtensible Markup Language (XML) [5] and related technologies (e.g. XPointer [16], XLink [15], and DOM [1]).

The documents we are concerned with have overlapping content. In order to facilitate comparison and the identification of the relationships between the contents of distributed documents, we assume that the participating documents are specified in XML.

We developed a *consistency link generator* to produce consistency links automatically, based on the consistency rules. The consistency link generator evaluates consistency rules relevant to pairs of distributed documents. It identifies sets of possible related elements and checks these elements through the conditions of the consistency rules. Depending on the type of the consistency rule and on the result of the evaluation of the conditions, consistency links are created.

After creating the consistency links and associating related elements, the distributed documents are displayed in a browser allowing users to navigate through the documents by clicking on elements and being taken to the elements in other documents to which they relate.

3. CONSISTENCY RULE SYNTAX

In this section we describe the syntax used to express the consistency rules. The consistency rule syntax is based on XML [5] and XPointer [16]. The reasons for using XML and XPointer are that they provide an open and standardised basis for specifying the consistency rules and dramatically simplifies the task of constructing a rule interpreter (consistency link generator).

Figure 1 shows the Document Type Definition (DTD) [5] for our consistency rule syntax. The `ConsistencyRule` element is composed of six element contents and attributes *id* and *type*. The *id* attribute is a unique identification for a consistency rule. The *type* attribute specifies the type of the rule and contains three possible values: `CT`, `CF`, and `IF`. The rule type specification is used to express the kind of relationships among elements being compared. Thus, the first argument (`C` or `I`) determines whether two elements are related because they are consistent or inconsistent, with respect to a certain rule. The second argument (`T` or `F`) specifies whether the consistency rule is or is not mandatory (true or false, respectively). Note that it does not make sense to have the case `IT` (inconsistent and mandatory), which would mean that there have to be related inconsistent elements in the participating distributed partial specifications.

The description of the six element contents is as follows:

- `Description` - it contains a natural language explanation of the rule.
- `Source & Destination` - they contain XPointer expressions for identifying possible related elements depending on the type of the rule. It is likely that there may be more than one `Destination` element related to the same `Source` element. This occurs when a rule relates more than two elements in the same partial specification or distinct partial specifications. Each `Destination` element in a rule has a unique identification represented by attribute *dest_id*, which is referenced in element `Condition`.
- `Condition` - is composed of four attributes: (a) *expsource*, an expression related to the `Source`

element; (b) *op*, an operator associating *expsource* with *expdest*, which can have the following values: *equal, not_equal, greater_than, less_than, less_equal, gerater_equal, sum, difference, multiplicity, division, average*; (c) *dest_ref,* a reference to the unique identification of a `Destination` element; and (d) *expdest,* an expression related to the `Destination` element.

- `Operator` - this element is related to the situation in which the consistency rule is composed of more than one condition. In this case, `Operator` can have the boolean values "AND" and "OR".

```
<!ELEMENT ConsistencyRule(Description,
Source,Destination+,Condition,(Operator,
                                Condition)*)>
<!ATTLIST ConsistencyRule
          id        ID          #REQUIRED
          type      (CT|CF|IF)  #REQUIRED>
<!ELEMENT Description  (#PCDATA)>
<!ELEMENT Source       (XPointer)>
<!ELEMENT Destination  (XPointer)>
<!ATTLIST Destination dest_id ID  #REQUIRED>
<!ELEMENT XPointer     (#PCDATA)
<!ELEMENT Condition      EMPTY>
<!ATTLIST Condition
          expsource CDATA  #REQUIRED
          op        CDATA  #REQUIRED
          dest_ref  IDREFS #REQUIRED
          expdest   CDATA  #REQUIRED>
<!ELEMENT Operator   EMPTY>
<!ATTLIST Operator   value  (AND | OR)  "AND">
```

Figure 1: Consistency rule syntax

4. CLASSIFICATION FOR THE CONSISTENCY RULES

In this section we present a classification for different types of consistency rules that we can represent by using the syntax described in section 3. The different types of consistency rules are related to distinct types of inconsistencies that can exist among partial specifications generated during distributed software system development. In order to illustrate the classification, we present examples of the different types of consistency rules based on the DTD shown in Figure 1.

The classification and respective examples described in this section are related to UML models, Z specifications, Data Flow Diagrams, and other software engineering documents. We use the Unified eXtensible Format (UXF) [24] and the XML Metadata Interchange (XMI) [19] DTDs, for the UML models. We also use a DTD for the Z specification based on the Z Interchange Format (ZIF), presented in Z standards [30].

In order to produce the classification described below, some important criteria have been taken into account. Partial specifications are represented as XML documents. First, in our approach to consistency management [10] the participating documents and the content of the `Source` and `Destination` sets in a consistency rule are always compared pair wise. Second, in XML documents data is represented as *elements* and *attributes*. XML has no rules regarding when data should be represented as an element or as an attribute. For instance, in XMI [19] all the components of the UML metamodel are represented as XML elements. As a consequence of this we identify *general types* of consistency rules, which compare documents, elements, and mixture of documents and elements. In the text, we use the term elements to mean both XML elements and XML attributes.

Table 1 summarises the classification that we use for the consistency rules. In the table, the names in the rows and columns are related to the different types of components being compared: the `Source` set and the `Destination` set, respectively. The content of each position in the table refers to different types of consistency rules described below. The most common rules compare two elements and we refine Type 1 in Table 2 with a set of specialized types of rules appropriate for partial software engineering specifications.

	Element	Document	Unrelated
Element	Type 1	Type 2	Type 3
Document	Type 4	Type 5	Type 6

Table 1: Classification of consistency rules

Transitivity	Type 7
Cross-reference	Type 8
Dependency	Type 9
Associativity	Type 10
Fixed values	Type 11

Table 2: Specialized types of consistency rules

4.1 Consistency rules

In this subsection we describe and illustrate different general types of consistency rules. In the examples we present instances of mandatory, non-mandatory, consistent and inconsistent rules (CT, CF, and IF).

Type 1: *Existence of related elements*
This type is related to the situation in which the existence of an element *e1* in a partial specification *d1* requires the

existence of an element *e2* in another partial specification *d2*.

Example: For every instance e1 in a UML collaboration diagram d1, there must exist a class e2 in a UML class diagram d2, where the name of e2 equals the type of e1.

```
<ConsistencyRule     id = "r1"
                     type = "CT">
  <Description> For every instance in a
   collaboration diagram there must
   exist a class in a class diagram
   where the name of the class equals
   the type of the instance.
  </Description>
  <Source><XPointer>
   root().child(all,Package).
   (all,CollaborationDiagram).
   (all,Collaboration).(all,Instance)
  </XPointer></Source>
  <Destination dest_id="CL1"><XPointer>
   root().child(all,Package).
   (all,ClassDiagram).(all,Class)
  </XPointer></Destination>
  <Condition
      expsource="origin().attr(CLASS)"
      op="equal"
      dest_ref="CL1"
     expdest="origin().attr(NAME)">
  </Condition></ConsistencyRule>
```

Type 2: *Existence of documents, due to the existence of an element*

This type of rule is related to the situation in which the existence of an element *e1* in a partial specification *d1* requires the existence of another partial specification *d2*.

Example: For every non-primitive process e1 in a data flow diagram (DFD) d1, there must exist a DFD d2 with the same name as e1.

```
<ConsistencyRule     id = "r2"
                     type = "CT">
 <Description>For every non-primitive
   process e1 in a DFD, there must exist a
   DFD with the same name as  e1.
 </Description>
 <Source><XPointer>root().child(all,
   Diagrams).(all,DFD).all(Process).all
   (Non-primitive)</XPointer></Source>
 <Destination dest_id="DFD1"><XPointer>
   root().child(all,Diagrams).(all.DFD).
   (all,TaggedValue).(all,Tag)
 </XPointer></Destination>
 <Condition  expsource="origin().attr(NAME)"
      op="equal"
      dest_ref="DFD1"
     expdest="origin().attr(VALUE)">
 </Condition></ConsistencyRule>
```

Type 3: *Existence of an unrelated element*

This type of rule is related to the existence of an unrelated element *e1* in a partial specification *d1*.

Example: For every UML model d1, there must exist a package e1 in d1.

```
<ConsistencyRule     id = "r3"
                     type = "CT">
<Description> For every UML model there must
 exist a package e1. </Description>
<Source><XPointer>
 root().child(all,XMI.content).(all,
 Model_Management.Model)</XPointer></Source>
<Destination dest_id="P1"><XPointer>root().
 child(all,XMI.content).(all,Model_
 Management.Model)</XPointer></Destination>
<Condition expsource="origin().descendant
  (all,FoundationCore.Namespace.
   ownedElement).child(all,Model_Management.
   Package)"
  op="equal"
  dest_ref="P1"
   expdest="#unrelated">
</Condition></ConsistencyRule>
```

Type 4: *Existence of elements, due to the existence of a document*

This type of rule is related to the situation in which the existence of a partial specification *d1* requires the existence of an element *e1* in another partial specification *d2*.

Example: A partial specification d1 containing an informal structure text describing the flow of events of a use case and a use case e1, in a UML model d2, with the same name, are considered to be related.

```
<Consistency>
<ConsistencyRule     id = "r4"
                     type = "CF">
 <Description> An informal structured text
  describing the flow of events of a use
  case and a use case e1 in a UML model with
  the same name, are considered to be
  related. </Description>
 <Source><XPointer>
  root().child(all,XMI.content).(all,
  Model_Management.Model).descendant
  (all,Foundation.Core.Namespace.
  ownedElement). child(all,
  Behavioral_Elements.Use_Cases.
  UseCase).(all,Foundation.Core.Model
  Element.taggedvalue).(all,Foundation.
  Extension_Mechanisms.TaggedValue)
 </XPointer></Source>
 <Destination dest_id="UC1"><XPointer>
  root().child(all,XMI.content).
   (all,Model_Management.Model).
  descendant(all,Foundation.Core.
  Namespace.ownedElement).child(all,
  Behavioral_Elements.Use_Cases.UseCase)
 </XPointer></Destination>
 <Condition  expsource="origin().
   ancestor(1,Foundation.Core.
   ModelElement.taggedValue).ancestor(1,
   Behavioral_Elements.Use_Cases.
   UseCase).child(1, Foundation.Core.
   ModelElement.name)"
  op="equal"
  dest_ref="UC1"
   expdest=" origin().child(1,Foundation.
```

```
   Core.ModelElement.name)"></Condition>
<Operator value="AND"/>
<Condition expsource=origin().child(1,
  Foundation.Extension_Mechanisms.
  TaggedValue.tag)"
 op="equal"
 dest_ref="UC1"
 expdest="#documentation">
</Condition></ConsistencyRule>
```

Type 5: *Existence of related documents*
This type of rule is related to the situation in which the existence of a partial specification *d1* requires the existence of another partial specification *d2*.
Example: A document design partial specification d1 and a document code partial specification d2, with the same name, are considered to be related.

```
<ConsistencyRule     id = "r5"
                     type = "CF">
 <Description> A document design
  specification and a document code
  specification, with the same name, are
  considered to be related.</Description>
 <Source><XPointer> root().child(all,
   Doc_Des_Spe).(all,TagValue)
 </XPointer></Source>
 <Destination dest_id="DCS1"><XPointer>
   root().child(all,Doc_Cod_Spe).(all,
   TagValue)</XPointer></Destination>
 <Condition expsource="origin().attr(NAME)"
     op="equal"
     dest_ref="DCS1"
    expdest="origin().attr(NAME)">
 </Condition></ConsistencyRule>
```

Type 6: *Existence of an unrelated document*
This type of rule is related to the existence of an unrelated partial specification *d1*.
Example: For every software system being developed there must exist a partial specification d1 concerning its software requirements partial specification (SRS).

```
<ConsistencyRule     id = "r6"
                     type = "CT">
 <Description> For every software
  system being developed, there must
  exist a document with its SRS.
 </Description>
 <Source><XPointer>
   root().child(all,Documents)
 </XPointer></Source>
 <Destination dest_id="D1">
 <XPointer>root().child(all,Documents)
 </XPointer></Destination>
 <Condition expsource="origin().
   child(all,Sof_Req_Spe)"
  op="equal"
  dest_ref="D1"
  expdest="#unrelated">
 </Condition></ConsistencyRule>
```

4.2 Specialised types of consistency rules

In this subsection we describe and illustrate different specialised types of consistency rules.

Type 7: *Transitivity*
This type of rule is related to the situation in which the existence of two elements *e1* and *e2*, in two different partial specifications *d1* and *d2*, with element *e1* related to element *e2* in partial specification *d1*, demands the absence of a relationship between elements *e1 and e2,* in partial specification *d2*. The relationship between *e1* and *e2* should not appear in *d2* either directly or indirectly, by transitivity through other elements.
Example: For every class e1 and subclass e2 in a UML class diagram d1, e2 should not be a superclass of e1 in any other UML class diagram d2, of the same UML model, for any level of nesting.

```
<ConsistencyRule      id = "r7"
                      type = "IF">
 <Description>For every class e1 and
  subclass e2 in a class diagram, e2
  should not be a superclass of e1 in
  any other class diagram, for any
  level of nesting </Description>
 <Source><XPointer>
  root().child(all,XMI.content).(all,
  Model_Management.Model).(all,
  Foundation.Core.Namespace.
  ownedElement).(all,Foundation.Core.
  Generalization) </XPointer></Source>
 <Destination dest_id="CL1"><XPointer>
  root().child(all,XMI.content).(all,
  Model_Management.Model).(all,Foundation.
  Core.Namespace.ownedElement).(all,
  Foundation.Core.Generalization)</XPointer>
 </Destination>
 <Condition expsource="#transitivity"
            op="equal"
            dest_ref="CL1"
            expdest="#transitivity">
 </Condition>
 <Operator  value="AND"/>
 <Condition expsource="id(origin().
    descendant(all,Foundation.Core.
    Generalization.subtype).child(1,
    #element,xmi.idref,*)).child(1,
    Foundation.Core.ModelElement.name)"
   op="equal"
   dest_ref="CL1"
   expdest="id(origin().descendant(all,
    Foundation.Core.Generalization.
    supertype).child(1,#element,
    xmi.idref,*)).child(1,Foundation.
    Core.ModelElement.name)"></Condition>
 <Operator  value="AND"/>
 <Condition expsource=="id(origin().
    descendant(all,Foundation.Core.
    Generalization.supertype).
    child(1,#element,xmi.idref,*)).
    child(1,Foundation.Core.
```

```
    ModelElement.name)"
   op="equal"
   dest_ref="CL1"
   expdest="id(origin().descendant(all,
    Foundation.Core.Generalization.
    subtype).child(1,#element,
    xmi.idref,*)).child(1,Foundation.
    Core.ModelElement.name)">
 </Condition></ConsistencyRule>
```

Type 8: *Cross-reference*

This type of rule is concerned with the situation in which the existence of two related elements *e1* and *e2* in partial specification *d1*, and the existence of an element *e'2,* related to element *e2,* in another partial specification *d2*, demand the existence of an element *e'1* in partial specification *d2*, where *e'1* is related to *e1*, and *e'1* is related to *e'2.*

Example: *For every class e1 with a subclass e2 in a UML class diagram d1, if there is a schema e'2, in a Z specification document d2, with the same name as the subclass e2, then there must exist an inclusion e'1 in schema e'2, with the same name as class e1.*

```
<ConsistencyRule     id = "r8"
                     type = "CT">
  <Description>For every class e1 and
   subclass e2 in a UML class diagram,
   if there is a schema e'2 in a Z
   document with the same name as the
   subclass e2, then there must exist
   an inclusion e'1 in e'2 with the
   same name as class e1 </Description>
 <Source><XPointer>root().child(all,
   Package).(all,ClassDiagram).(all,
   Class).(all,Generalization)
 </XPointer></Source>
 <Destination dest_id="SC1">
   root().child(all,Zparas).(all,Schemadef).
   (all,formals)<XPointer></Destination>
 <Condition expsource="origin().
  ancestor(1,Class).attr(NAME)"
   op="equal"
   dest_ref="SC1"
   expdest="origin().ancestor(1,Schemadef).
  attr(NAME)"></Condition>
<Operator  value="AND"/>
<Condition expsource="origin().attr(FROM)"
   op="equal"
   dest_ref="SC1"
   expdest="origin().attr(NAME)">
</Condition> </ConsistencyRule>
```

Type 9: *Dependency*

This type of rule is related to the situation in which the existence of an element *e1* expressing a relationship between two elements *e2* and *e3*, where *e2* is related to another element or partial specification *e'2* (*d'2*), and *e3* is related to another element or partial specification *e'3* (*d'3*), demands the existence of an element *e'1*, which expresses a relationship between *e'2* (*d'2*) and *e'3* (*d'3*).

Example: *For every association between two classes e2 and e3 that are included in two different UML packages e2' and e3', there must exist a dependency between these two packages.*

```
<ConsistencyRule     id = "r9"
                     type = "CT">
 <Description> For every association
  between two classes in two packages there
  must exist a dependency between these two
  packages.</Description>
 <Source><XPointer>root().child(all,
   XMI.content).(all,Model_Management.
   Model).descendant(all,Foundation.Core.
   Namespace.ownedElement).child(all,
   Foundation.Core.Association)
 </XPointer></Source>
 <Destination dest_id="DP1"><XPointer>
   root().child(all,XMI.content).(all,
   Model_Management.Model).descendant
   (all,Foundation.Core.Namespace.owned
   Element).child(all,Foundation.Core.
   Dependency)</XPointer></Destination>
 <Condition expsource="id(origin().
   child(all,Foundation.Core.Association.
   connection).(1,Foundation.Core.
   AssociationEnd)…child(1,Foundation.
   Core.ModelElement.name)"
  op="equal"
  dest_ref="DP1"
  expdest="id(origin().child(1,Foundation.
   Core.Dependency.client)….child(1,
   Foundation.Core.ModelElement.name)">
 </Condition>
 <Operator value="AND"/>
 <Condition  expsource="id(origin().child
  (all,Foundation.Core.Association.
   connection).(2,Foundation.Core.
   AssociationEnd)….child(1,Foundation.
   Core.ModelElement.name)"
  op="equal"
  dest_ref="DP1"
  expdest="id(origin().child(1,Foundation.
   Core.Dependency.supplier)….
   child(1,Foundation.Core.ModelElement.
   name)"></Condition>
<Operator value="OR"/>
<Condition expsource="id(origin().child(all,
   Foundation.Core.Association.
   connection).(1,Foundation.Core.
   AssociationEnd)…child(1,Foundation.
   Core.ModelElement.name)"
  op="equal"
  dest_ref="DP1"
  expdest="id(origin().child(1,
   Foundation.Core.Dependency.
   supplier)….child(1,Foundation.Core.
   ModelElement.name)"></Condition>
 <Operator value="AND"/>
 <Condition  expsource="id(origin().child
   (all,Foundation.Core.Association.
   connection).(2,Foundation.Core.
  AssociationEnd)….child(1,Foundation.
  Core.ModelElement.name)"
```

```
 op="equal"
 dest_ref="DP1"
 expdest="id(origin().child(1,Foundation.
  Core.Dependency.client)….child(1,
  Foundation.Core.ModelElement.name)">
</Condition></ConsistencyRule>
```

Type 10: *Associativity*

This type of rule is concerned with the case where the existence of an element *e1* in a partial specification *d1*, demands the existence of either element *e2*, or element *e3*, or element *en (n _ _)* in another partial specification *d2*.

Example: For every association e1 in a UML class diagram d1, there must exist either a schema e2 in a Z partial specification d2, with the same name as the association e1, or a variable e3 in a schema in a Z partial specification d2, with the same name of the association e1, and the variable must be of type relation or cartesian product.

```
<ConsistencyRule   id = "r10"
                   type = "CT">
 <Description>For every association in a
   UML class diagram there must exist
   either a schema in a Z document with
   the same name as the association, or
   a variable in a schema with the same
   name of the association, and the
   variable must be of type relation or
   cartesian product </Description>
 <Source><XPointer>
    root().child(all,Package).(all,
    ClassDiagram).(all,Class).(all,
     Association)</XPointer></Source>
  <Destination dest_id="SC1"><XPointer>
     root().child(all,Zparas).(all,
     Schemadef)</XPointer></Destination>
  <Destination dest_id="VR1"><XPointer>
     root().child(all,Zparas).
     (all,Schemadef).(all,decpart).
     (all,declaration).(all,variable)
  </XPointer></Destination>
 <Condition
    expsource="origin().attr(NAME)"
    op="equal"
    dest_ref="SC1"
    expdest="origin().attr(NAME)">
 </Condition>
 <Operator  value="OR"/>
 <Condition
    expsource="origin().attr(NAME)"
    op="equal"
    dest_ref="VR1"
    expdest="origin().attr(NAME)">
 </Condition>
 <Operator  value="AND"/>
 <Condition expsource="#relation"
    op="equal"
    dest_ref="VR1"
    expdest="origin().ancestor(1,
     declaration).child(all,rel).
     attr(kind)">
 </Condition>
 <Operator  value="OR"/>
 <Condition
    expsource="origin().attr(NAME)"
    op="equal"
    dest_ref="VR1"
    expdest="origin().attr(NAME)">
 </Condition>
 <Operator  value="AND"/>
 <Condition
   expsource="#cartesianproduct"
   op="equal"
   dest_ref="VR1"
   expdest="origin().ancestor(1,
    declaration).child(all,rel).
    attr(kind)">
 </Condition></ConsistencyRule>
```

Type 11: *Fixed values*

This type of rule is concerned with comparisons among elements and partial specifications that have fixed values. The example described in consistency rule Type 10 also illustrates Type 11.

Example: For every method e1 of a class in a UML class diagram d1, there must exist a schema e2 in a Z partial specification d2, with the same name of the method, and the purpose of the schema e2 needs to be of value "operation".

```
<ConsistencyRule   id = "r11"
                   type = "CT">
 <Description>For every method of a class in
  a class diagram, there must exist a schema
  in a Z specification with the same name of
  the method, and the purpose of the schema
  needs to be of value "operation".
</Description>
 <Source><XPointer>root().child(all,
  Package).(all,ClassDiagram).(all,Class).
   (all,Operation)</XPointer></Source>
 <Destination dest_id="SC1"><XPointer>
   root().child(all,Zparas).(all,
   Schemadef)</XPointer></Destination>
 <Condition expsource="origin().attr(NAME)"
   op="equal"
   dest_ref="SC1"
   expdest="origin().attr(NAME)">
</Condition>
<Operator value="AND"/>
<Condition expsource="#operation"
   op="equal"
   dest_ref="SC1"
   expdest="origin().attr(PURPOSE)">
</Condition></ConsistencyRule>
```

5. THE RULE EDITOR

Although XML [5] and XPointer [16] provide a simple and efficient way of describing consistency rules, we recognise that the syntax proposed may not make it straightforward for users to define the rules. The difficulties are related primarily to the following issues: (a) the size of the expressions (this is generally because the DTDs are large

and complex); (b) the XPointer syntax; and (c) the requirement to have a detailed understanding of the DTDs. In order to alleviate these difficulties we developed a rule editor to assist with the partial specification of the consistency rules. The rule editor is implemented in Java, using JDK 1.2.2 and the XML parser for Java from IBM Alphaworks.

Figure 2 presents a screen dump of the rule editor. When using the editor the user selects the type of documents associated with the consistency rule to be specified. Based on this selection the tool presents the respective DTDs for these documents in a tree format. The user defines the `Source` and `Destination` elements and the respective `Conditions` in an interactive way, by selecting elements and attributes from the DTD structures. The rule editor translates the information specified by the user into a consistency rule based on the DTD presented in Figure 1.

Figure 2: A consistency rule specification in the rule editor

6. RELATED WORK

Many approaches have been proposed to support consistency management of multiple perspectives in different formats. In [8][9][11] the authors view inconsistency as a logical concept. They proposed a first-order logic-based approach to inconsistency handling in the ViewPoints framework. This approach has provided us with the conceptual underpinning of the work reported but has not been implemented in a distributed setting. Spanoudakis and Finkelstein [25][26] suggested a method called *reconciliation* to allow detection and verification of overlaps and certain types of inconsistencies between specifications expressed in an object-oriented framework. When managing consistency, overlap detection is an activity that precedes consistency rule construction [12]. We are currently investigating the use of the reconciliation method to support identification of consistency rules.

Van Lamsweerde et al. [28][29] proposed a formal framework for various types of inconsistencies that can occur during requirements engineering process. The idea is to manage conflicts at the goal level in the context of the KAOS requirements engineering methodology [7]. This is achieved by introducing new goals or by transforming specifications of goals into new specifications free of conflicts.

Some software development environment projects integrate tools for different languages and incorporate consistency constraints that span across different documents. Examples are found in Arcadia [27], ESF [20], ATOMOSPHERE [3], and GOODSTEP [13]. The approaches used in all of these projects utilise a centralised repository, such as PCTE [4] or an object database for storing documents. However, the use of these repository limits scalability and commitment on the part of both users and tool vendors to a heavyweight integration mechanism. Identification and resolution of semantic and syntactic conflicts are also issues in the multidatabase system domain. Many approaches have been proposed in the literature [6][14][22][23]. A survey of different approaches to detect and resolve conflicts can be found in [2]. We believe that more needs to be done to evaluate these approaches in the context of software engineering.

Although the existing approaches have contributed to a better understanding of the consistency management problem we have only just scratched the surface and considerable further work is required to make consistency management a practical proposition.

7. CONCLUSION AND FUTHER WORK

This paper describes an approach to express consistency rules among distributed specifications. The approach uses XML and related technologies to allow Internet-scale distribution and standardisation of the consistency management process. In order to illustrate our approach and to specify the types of inconsistencies in which we are interested we have presented a classification for different types of consistency rules supported by the approach. The approach is lightweight and can be deployed in a variety of different settings.

We are extending our work to allow other types of consistency rules. In particular, rules involving logical quantifiers ($\forall$, $\exists$) and numeric quantifiers (1, 2, …, n). In addition, we are also exploring rules related to particular domains. We are now using the approach in real applications with some very large specifications, it is out of the scope of the paper to report on these trials, however we believe that they are showing promising results. Once this work is further advanced we plan to look in more detail at inconsistency handling.

ACKNOWLEDGEMENTS

We would like to thank S. Dupuy, E. Ellmer, S. Guerra, C. Nentwich, and T. Revheim for their helpful discussion on the topic. We would also like to acknowledge financial support from the European Union through the RENOIR project and PROMOTER.

REFERENCES

[1] V. Apparo, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A.L. Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification. Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium.

[2] C. Batini, M. Lenzerini, and S.B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computer Surveys*, 18(4), pages 323-364, December 1986.

[3] J. Boarder, H. Obbink, M. Schmidt, and A. Volker. Advanced Techniques and Methods of System Production in a Heterogeneous, Extensible, and Rigorous Environment. In N. Madhavji, W. Schafer, and H. Weber, editors, *Proc. of the 1st Int. Conf. On System Development Environments and Factories*, Berlin, Germany, pages 199-206, London. Pitman Publishing, 1989.

[4] G. Boudier, F. Gallo, R. Minot, and I. Thomas. An Overview of PCTE and PCTE+. *ACM SIGSOFT Software Engineering Notes*, 13(2), pages 248-257. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Mass, 1989.

[5] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language. Recommendation http://www.w3.org/TR/1998/REC-xml-19980210, World Wide Web Consortium.

[6] M.W. Bright, A.R. Hurson, and S. Pakzard. Automated Resolution of Semantic Heterogeneity in Multidatabases. *ACM Transaction on Database Systems*, 19(12), pages 212-253, June 1994.

[7] R. Darimont and A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. *In Proceedings FSE'4 - Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, San Francisco, pages 179-190, October 1996.

[8] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Co-ordinating Distributed ViewPoints: the anatonomy of a consistency check. In *Concurrent Engineering Research & Applications,* CERA Institute, USA 1994.

[9] S. Easterbrook and B. Nuseibeh. Using ViewPoints for Inconsistency Management. *IEE Software Engineering Journal*, November 1995.

[10] E.Ellmer, W. Emmerich, A. Finkelstein, D. Smolko, and A. Zisman. Consistency Management of Distributed Documents using XML and Related Technologies. Submitted for publication. 1999.

[11] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering,* 20(8), pages 569-578, August 1994.

[12] A. Finkelstein, G. Spanoudakis, and D. Till. Managing Interference. Joint Proceedings of the SIGSOFT'96 Workshops – Viewpoints'96: An International Workshop on Multiple Perspectives on Software Development, San Francisco, ACM Press, pages 172-174, October 1996.

[13] GOODSTEP Team. The GOODSTEP Project: General Object-Oriented Database for Software Engineering Processes. In K. Ohmaki, editor, *Proc. of the Asia-Pacific Software Engineering Conference*, Tokyo, Japan, pages 410-420. IEEE Computer Society Press, 1994.

[14] J. Hammer and D. McLeod. An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems. *International Journal of Intelligent and Cooperative Information Systems*, 2(1), pages 51-83, 1993.

[15] E. Maler and S. DeRose (1998). XML Linking Language (XLink). Technical Report http://www.w3.org/TR/1998/WD-xlink-19980303, World Wide Web Consortium.

[16] E. Maler and S. DeRose (1998). XML Pointer Language (XPointer). Technical Report http://www.w3.org/TR/1998/WD-xptr-19980303, World Wide Web Consortium.

[17] K. Narayanaswamy and N. Goldman. "Lazy" Consistency: A Basis for Cooperative Software Development. In *Proc. of Int. Conf. On Computer-Supported Cooperative Work (CSCW'92),* Toronto, Ontario, Canada, pages 257-264 - ACM SIGHI & SIGOIS.

[18] B. Nuseibeh. To Be and Not to Be: On Managing Inconsistency in Software Development. In *Proc. of 8th IEEE International Workshop on Software Specification & Design (IWSSD-8)*, pages 164-169, March 1996.

[19] OMG (1998). XML Metadata Interchange (XMI) - Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF). Technical Report AD Document AD/98-10-05, Object Management Group,m 492 Old Connecticut Path, Framingham, MA 01701, USA.

[20] W. Schäfer and H. Weber. European Software Factory Plan – The ESF-Profile. In P. A. Ng and R.T. Yeh, editors, *Modern Software Engineering – Foundations and current perspectives*, chapter 22, pages 613-637. Van Nostrand Reinhold, New York, 1989.

[21] R.W. Schwanke and G.E. Kaiser. Living with Inconsistency in Large Systems. *In Proc. of the Int. Workshop on Software Version and Configuration Control,* Grassau, Germany, pages 98-118, B.G. Teubner, Stuttgart.

[22] E.Sciore, M. Siegel, and A. Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2), pages 254-290, June 1994.

[23] M. Siegel and S.E. Madnick. A Metadata Approach to Resolving Semantic Conflicts. *In proceedings of the 17th International Conference on Very Large DataBases,* pages 133-145, Barcelona, Spain, 1991.

[24] J. Suzuki and Y. Yamamoto. Making UML models exchangeable over the Internet with XML: The UXF Approach. In Muller, P.-A. and Bezivin, J., editors, *Proc. Of Int. Worshop on UML'98*, Mulhouse, France, Lecture Notes in Computer Science. Springer.

[25] G. Spanoudakis and A. Finkelstein. Reconciliation: Managing Interference in Software Development. *In Proceedings of the ECAI '96 Workshop on Modelling Conlicts in AI*, Budapest, Hungary, 1996.

[26] G. Spanoudakis and A. Finkelstein. A Semi-automatic Process of Identifying Overlaps and Inconsistencies between Requirements Specifications. *In Proceedings of the 5th International Conference on Object-Oriented Information Systems* (OOIS 98), pages 405-425, 1998.

[27] R.N. Taylor, R.W. Selby, M. Young, F.C. Belz, L.A. Clarce, J.C. Wileden, L. Osterweil, and A.L Wolf. Foundations of the Arcadia Environment Architecture. *ACM SIGSOFT Software Engineering Notes*, 13(5), pages 1-13. Proc. of the 4th ACM SIGSOFT Symposium on Software Development Environments, Irvine, Cal, 1988.

[28] A. van Lamsweerde. Divergent Views in Goal-Driven Requirements Engineering. *In Proceedings of the ACM SIGSOFT Workshop on Viewpoints in Software Development,* San Francisco, pages 252-256. October 1996.

[29] A. van Lamsweerde, R. Darimont, and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transaction on Software Engineering.* November 1998.

[30] Z Standards panel. Z Notation version 1.2. TR, September, 1995. http://www.comlab.ox.ac.uk/oucl/groups/zstandards.

APPENDIX: CASE STUDY

This paper concerns consistency rules and therefore the examples presented are at the level of the particular specification schemes such as UML and Z rather than an *instance* level specification such as the case study. We have a UML specification of the TRMCS and would be able to demonstrate our tools in the context of this case study if this is of interest.