FpML Validation

Joint proposal from UBS Warburg, University College London, and Systemwire.

Version:	1.0
Editor:	Bryan Thal (UBS Warburg)
Participants:	Wolfgang Emmerich (UCL) Steven Lord (UBS Warburg) Daniel Dui (UCL) Christian Nentwich (Systemwire)

Change History:

Date	Version	Author	Change description
10/05/02	0.1	B. Thal	First draft release distributed for comments and feedback.
30/05/02	0.95	W. Emmerich	
10/06/02	0.97	D. Dui	
19/06/02	0.99	B. Thal	Final review
25/06/02	1.0	D. Dui	Final changes

Table of Contents

1 Introduction and Background

Annual ISDA Operations Surveys have found that delays and high costs in the processing of complex OTC Derivative trades are to some extent due to the manual nature of trade validation and trade confirmation processing. Trade validation ensures that trades that are exchanged between counterparties or between different systems of the same organization meet a number of consistency constraints. During trade confirmation, details about a trade held by one organization are checked against the details provided by the counterparty. The trade matching process is required to identify inconsistencies between them.

With the trend towards XML based information representation in finance and the resultant need for systematic ways of identifying and reconciling inconsistencies between XML documents, UBS Warburg is sponsoring a PhD studentship at University College London. The aim of the studentship, which started in September 2001, is to investigate the management of inconsistency in processing XML based financial trading data for OTC derivative trades. The associated business goal is to support more effective straight-through-processing of financial trading data.

The aim of this proposal is to put forward a validation rule language for FpML, which can be used to unambiguously describe the rules and also to execute them. It looks at the requirements for validation drawing on work carried out by the PhD student and the FpML Tools workshop in August 2001. It then puts forward the grammar, syntax and operation of the validation rule language based on Xpath before setting out a sample ruleset for FpML 1.0.

The Software Systems Engineering Group at University College London has a long-standing interest in all aspects of inconsistency management. The group has developed algorithms and technologies that support consistency checks across distributed data represented in XML. The IPR of these algorithms and technologies have been transferred to Systemwire, a UCL spin-off company. Systemwire markets a product family called xlinkit, which supports the specification of consistency constraints, their efficient execution and various forms of diagnoses of the results of consistency checks. More details about the company and xlinkit can be found at http://www.systemwire.com.

The work carried out to date has looked at business efficiency issues and at extending the algorithms and consistency checking architeecture of xlinkit to make it suitable for FpML Version 1.0. Following demonstrations of this work to members of the FpML Standards Committee, UBS Warburg, University College London and Systemwire would like to put forward a proposal for FpML Validation.

2 Validation Requirements

Validation is a critical requirement for any organization implemeting FpML. It was identified as a high priority area during the FpML Tools workshop in August 2001. High-level requirements for validation were identified and refined at the workshop (see http://www.fpml.org/tools/toolswork.asp). These requirements are set out below. The PhD student's research has also investigated financial institutions' requirements for checking electronic trade representations. This has resulted in an ordering of sources of inconsistency.

2.1 FpML Tools Workshop Conclusions on Validation Requirements

We reached the following conclusions:

R2.1.1: Validation should focus on semantic or business validation. It is assumed that XML parsers are used for XML syntax validation based on the FpML DTD/schema (for both well-formedness and syntactic validation).

R2.1.2: There was a preference for an XML based predicate or rule definition, as tools can be built to process such rules.

R2.1.3: This includes GUI tools to enable business analysts to formulate and update the rules. The provision of a simple API to allow callout for programmed validation was discussed as an alternative.

R2.1.4: FpML could supply validation rules with each version of FpML for 'community wide' issues. It is hoped that this proposal will help to facilitate this.

In conclusion it was noted by the workshop participants that validation is a critical and timely area for FpML to consider, as all institutions and vendors working with FpML have the requirement.

2.2 Ability to Express and Handle Different Classes of Validation Rules

In refinement of R2.1.4, the following examples of classes or levels of validation rule have been identified:

R2.2.1: Community/Industry Rules: these would come from FpML and include product and market conventions. They could include regulatory requirements for a particular product.

R2.2.2: Company Specific Rules: these would be rules specific to a company and implement policy requirements, such as market or credit risk parameters or collateral requirements.

R2.2.3: Department Specific Rules: these would be rules taking into account how a particular Department processes a document. It could include threshold levels at which a trade would require manual intervention.

R2.2.4: System Specific Rules: these would include rules that are to be enforced by particular trading systems in order to be able to process particular trades.

These classes are not considered to be exhaustive. The requirement is for any number of classes to be handled. Furthermore, we anticipate that the different FpML product working groups will determine a significant number of validation rules (a total of 100-300).

2.3 Comparison to External Data Sources (e.g. FpML Schemes)

R2.3.1: The formalism for specifying validation rules shall not just support validation of single FpML trades, but also comparisons between trades and other external data sources, such as static data, market data, or FpML Schemes.

R2.3.2: It can be assumed that either these external data sources are available in an XML markup language or can be transformed into an XML language. However, the formalism shall support external data sources in non-FpML languages.

R2.3.3: Often such market data is provided from outside a particular organization and this has implications on the distributed checks. However, the notation to express consistency rules should support the specification of consistency rules without assuming any details of where external data sources are located.

2.4 Usability

The validation rule language shall support the formulation of rules at appropriate levels of abstraction.

R2.4.2: To facilitate the conciseness of rules, the formalism shall support the declarative specification of consistency rules.

R2.4.3: At the same time the domain of financial derivative instruments is quite special and the rule language should offer extension mechanisms that can cope with the need for any domain-specific operators, macros and primitives.

R2.4.4: The language should be easily comprehensible and therefore use concepts that members of the FpML community are likely to be familiar with.

R2.4.5: The validation rule approach should be capable of coping with the complexity of a large number of rules (100-300) and in particular offer different structuring mechanisms that support the hierarchical decomposition of the overall set of rules and support the parallel definition of rules by different working groups.

R2.4.6: The formalism should be amenable to automated translation between internal (machine-readable) representations of validation rules and external representations that can be understood and serve as the specification of validation rules across product working groups.

The validation language will not only be used to standardize and specify validation rules, but these rules also need to be executed in order to actually perform the validation.

R2.4.7: The rule validation language should have a compiler or interpreter that can be used in order to execute validation rules.

2.5 Classification of product types

FpML can represent a large number of different products without actually providing language constructs. It might be necessary for purposes of processing and standardizing FpML contracts to classify them into different trades.

R2.5.1: The validation language should support such classification.

2.6 Compliance with XML Standards (XPath and XLink)

FpML demands for all its standards compliance with W3C standards.

R2.6.1: The validation approach adopted for FpML should therefore be compliant with current W3C standards.

2.7 Efficient Execution

It would be desirable if the language chosen for the validation rules could also directly be used to control the execution of validations. In particular, we would like to avoid the need to manually translate the rules into an execution and instead desire that this step be performed by an interpreter or compiler. This motivates the following requirement

R2.7.1: The validation rule language should be efficiently executable.

3 Possible Approaches to Validation

We have evaluated a number of possible approaches for specifying and executing validation rules. In this section, we will briefly present an overview of those approaches. This will then result in the selection of xlinkit as the most promising approach.

3.1 xlinkit

xlinkit is a framework for expressing and checking the consistency of distributed, heterogeneous documents. It comprises a language, based on a restricted form of first order logic, for expressing constraints between elements and attributes in XML documents. The restriction enforces that sets have a finite cardinality, which is not a problem as XML documents only have a finite set of elements and attributes. xlinkit also contains a document management mechanism and an engine that can check the documents against the constraints.

xlinkit has been implemented as a lightweight mechanism on top of XML and creates hyperlinks to support diagnostics by linking inconsistent elements. Because it was built on XML, xlinkit is flexible and can be deployed in a variety of architectures. It has also been applied in a variety of different application areas, including the validation of Software Engineering documents such as the design models and source code of Enterprise JavaBeans-based systems [1].

Sets used in quantifiers of xlinkit rules are defined using XPath. XPath [2] is one of the foundational languages in the set of XML specifications. It permits the selection of elements from an XML document by specifying a tree path in the document. For example, the path `/FpML/trade` would select all `trade` elements contained in the `FpML` element, which is the root element.

XLink [3] is the XML linking language and is intended as a standard way of including hyperlinks in XML documents. XLink goes beyond the facilities provided by HTML by allowing any XML element to become a link; by specifying that links may connect more than two elements, so called *extended* links; and by allowing links to be managed *out-of-bound*, as collections of links termed *linkbases*. These features allow us to capture complex relationships between a multitude of elements that are involved in an inconsistency without altering any of the inconsistent documents.

The linkbases generated by xlinkit form an ideal intermediate representation from which different forms of higher-level diagnoses can be derived. Firstly, xlinkit has a report generator that takes report templates and uses the linkbase to obtain details of the elements involved in an inconsistency to provide a report similar to an error report that a compiler generates. Secondly, xlinkit has a servlet that can read a linkbase and allows users to select a link and it will then open the documents referenced in the link, navigate to elements identified in the link and in that way assist users to understand the links. Xlinkit also has a linkbase processor that folds links back into the documents so that both consistent and inconsistent data can be captured as hyperlinks.

It depends on the application domain which of these higher-level diagnoses mechanisms is most appropriate. For the domain discussed in this paper we found the report generation to have generated most interest among our partners in various investment banks.

3.2 Attribute Grammars

There is a large body of work on validation of constraints of context-free languages in Compiler construction. The constraints that are considered are typically static semantic constraints, such as scoping and typing rules. These constraints are specified using for example attribute grammars [4], which have been shown to be efficiently executable by compilers. Attribute grammars are not very concise specifications of consistency as one constraint is typically spread over a large number of products. On the other hand, they have been shown to be very amenable to efficient execution [5, 6], which is an important property when considering compiling large amounts of source code on slow processors. We have made a slightly different trade-off decision with xlinkit and favour conciseness of the constraint definition over efficiency. This is particularly appropriate given the small size of derivative trade documents, which are in the order of 17KBytes. Moreover it would be difficult to integrate attribute grammars with XML parsers as the attribute grammar approaches assume that an integrated compiler is generated while XML parsers are generic and work in a customized manner.

The work on attribute grammars was then taken on for the construction of syntax-directed editors and software engineering environments, such as Gandalf [7], Synthesizer Generator [8], IPSEN [9], Centaur [10] and GOODSTEP [11].The focus of these environments was to incrementally check constraints during editing. This could only be achieved by translating attribute or graph grammars into efficiently executable code. Our focus is not on supporting the editing of trade representations, but to support the batch validation that occurs when trades are exchanged between organisations or different departments within an organisation. Provision of support for incremental checks is therefore not necessary and instead we favour the flexibility that comes with interpretation of constraints in the xlinkit rule engine.

3.3 OCL

We have also compared xlinkit rules with OMG's Object Constraint Language (OCL) [12]. OCL was defined to declare constraints in UML diagrams or MOF meta models. OCL was not defined with an aim to be executable. In particular, it allows for infinite sets (e.g. integer), which prevents it from being executed efficiently. The focus of xlinkit, however, was to be as expressive as possible, while still being executable in polynomial time.

3.4 Specification of Validation Rules with XSLT

The weakness of expressing constraints in the DTD and XML Schema languages has been recognized for some time now. Various approaches have been reported that use XSLT [13] for validation. In [14], we report about the TIGRA enterprise application integration architecture that uses XML as transport representation for financial trades. In that architecture we have used XSLT stylesheets to express constraints. The expressive power of XSLT stylesheets is considerably lower than that of xlinkit in that xlinkit supports the full power of first-order logic. Moreover, xlinkit carefully separates the concerns of constraint specification, document and rule location, and provision of diagnostic feedback, which would be intertwined in XSLT.

3.5 Schematron

Rick Jeliffe's Schematron [15] also uses XSLT to translate documents into reports about their consistency. However, Schematron manages to conceal the use of XSLT and provides a higher level of abstraction for the definition. Although Schematron works quite well for validating single documents it would not allow us to express constraints across different documents, e.g. to check trades against reference data or workflow representations or to compare two trades in different representations.

3.6 Summary

The following table summarizes the different approaches we investigated. It assesses how well each of the approaches supports the requirements identified in Section 2. The elements of the table identify the extent to which the requirements are addressed. A “++” denotes that the approach fully satisfies the requirement, a “o” denotes that the approach meets the requirement to some extent and a “--”denotes that the requirement is not addressed at all.

As can be seen from the table, the xlinkit approach is the most promising approach and therefore we have selected it for a more detailed assessment. We therefore present xlinkit rules in more detail in Section 4 before we show how it was used to specify validation rules for FpML 1.0

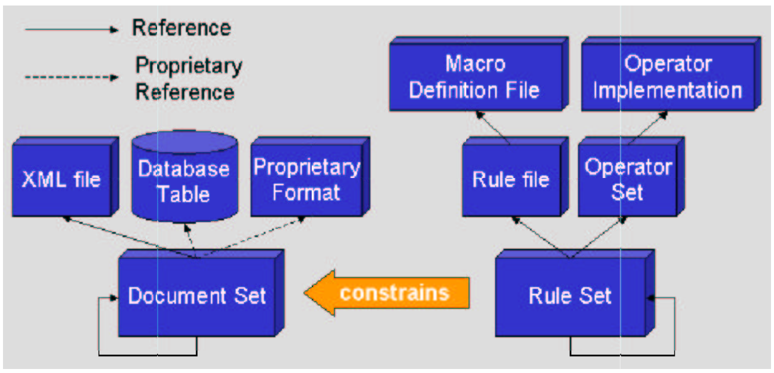
		Xlinkit	Attribute Grammars	OCL	XSLT	Schematron
R2.1.1	Semantic Validation	++	+	+	+	+
R2.1.2	XML-based Definition	++	-	-	++	++
R2.1.3	GUI tools to formulate and update rules	o	-	o	++	-
R2.2	Multiple distributed rule sets	++	-	-	-	-
R2.3.1	Comparison to external data sources	++	-	-	-	-
R2.3.2	Check against non-FpML languages	++	-	-	-	-
R2.3.3	Distributed data sources	++	-	-	+	+
R2.4.1	Declarative rule language	++	--	++	+	+
R2.4.2	Domain-specific operators	++	++	o	++	++
R2.4.3	Ease of comprehension	++	--	++	-	o
R2.4.4	Rule structuring mechanisms	++	--	+	-	-
R2.4.5	human + machine readable representation	+	--	+	-	+
R2.5.1	Classification of FpML product types	+	+	+	o	o
R2.6.1	W3C compliance	+	--	o	++	+
R2.7.1	Efficient Execution	+	++	--	++	++

4 xlinkit Validation Rule Language

This section explains the language of version 5 of xlinkit. The purpose of this explanation is to give a definition of the various data files, artifacts and languages provided by xlinkit. It is *not* intended as a user manual - it does not define how xlinkit works, how to use xlinkit, how to interpret its output, or any kind of best practice. For more information on these topics, please visit the xlinkit web site at http://www.xlinkit.com.

The xlinkit framework consists of many different artifacts and languages that combine to provide the unique benefits of xlinkit: arbitrary distribution of content, abstraction from underlying data formats, specification of complex constraints, integration of heterogeneous data and a flexible approach to consistency management. Figure 1 shows how the various artifacts provided by xlinkit are interrelated:

Figure 1. Language family overview



This document will discuss the artifacts shown in the Figure: Section 1 defines common mechanisms that appear in all artifacts: the referencing mechanism used in the Figure and xlinkit's metadata system; Section 2 defines document sets and Section 3 rule sets; Section 4 defines rule files and the xlinkit constraint language; Section 5 specifies xlinkit's operator plugin mechanisms, operator sets and operator implementations, and Section 6 defines macro files.

4.1 Common Mechanisms

This section summarises the mechanisms that are common to all xlinkit input files: the referencing mechanism for loading data files and the metadata for describing the input files.

4.1.1 Referencing

Many files in the xlinkit family contain references: rule sets reference rule files and further rule sets, operator sets reference operator implementations and so on. xlinkit therefore defines what such a reference should look like. In general, references come in two forms:

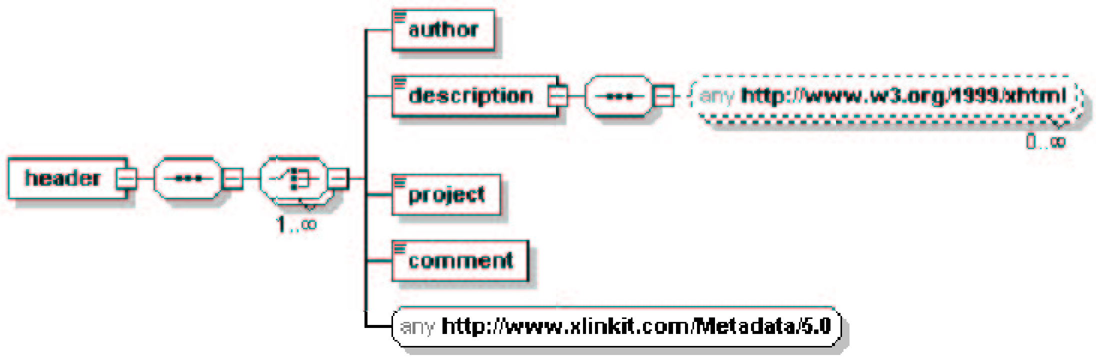
- A local filename, for example `rule.xml`, `C:\Rules\rule.xml` or `../../rule.xml`. A local filename can also take the form of a `file://` URL. In this case, xlinkit will simply remove the protocol section and treat it as a local file. Thus, `file://rule.xml` will be treated as `rule.xml`.
- An HTTP URL, for example `http://www.xlinkit.com/rule.xml`.

The important thing to note is that xlinkit does *not* support relative URLs. It is therefore not possible to mix filenames and URLs freely. For example, if a rule set includes a file `rule.xml` and the rule set is referenced locally, then the current directory will be searched for the file. If it is not there, an error occurs. If the same rule set is referenced using an HTTP URL, the local directory on the referencing host will still be searched for the file. Since this is likely to lead to errors, we recommend that URLs or filenames be used uniformly and not mixed.

4.1.2 Metadata

The xlinkit metadata elements are standardised throughout the different file types and can be used to annotate document sets, rule set, operator sets and individual consistency rules. It is their purpose to provide helpful annotation such as authorship information and documentation for the various artifacts in the xlinkit framework. Figure 1.1 shows a graphical representation of the `header` element, which contains the metadata.

Figure 1.1. Metadata Schema


author
description
any http://www.w3.org/1999/xhtml
0..∞
header
1..∞
project
comment
any http://www.xlinkit.com/Metadata/5.0

`author` contains the name of the author of an artifact. If there are multiple authors, one element should be used for each.

`description` can contain a textual description of the resource with which the header has been associated. In addition to text the element can contain elements from the XHTML namespace (`http://www.w3.org/1999/xhtml`), enabling the production of documentation web pages from xlinkit artifacts using stylesheets (see Example 1.2).

`project` can be an arbitrary text string that defines the context of the artifact. In the case of consistency rules, this string may be used as an identifier in future versions (see Section 4.2.2), otherwise it can be used arbitrarily.

`comment` is intended as a means to provide additional information beyond the description, for example on the status of the artifact.

The remaining extension point in the schema allows any element from the `http://www.xlinkit.com/Metadata/5.0` namespace to appear in the header. The content of these elements will not be validated and can be chosen arbitrarily (though it does have to be well-formed XML), for example to meet organisation-specific documentation requirements. See Example 1.3 for an example.

4.1.2.1 Examples

Example 1.1 shows some metadata used to annotate a consistency rule.

Example 1.1. Simple Rule Annotation

```
<consistencyrule id="r1">
  <header>
    <author>Christian Nentwich</author>
    <description>An invalid rule: the forall is empty!!</description>
    <project>xlinkit Language Reference</project>
    <comment>This is broken. Fix it.</comment>
  </header>
  <forall/>
</consistencyrule>
```

Example 1.2 demonstrates the use of additional XHTML elements in a document set description. Notice how the XHTML namespace is bound to the prefix `x:` and the prefix is used on the XHTML elements. Failure to use the prefix would cause a validation error.

Example 1.2. Using XHTML in Descriptions

```
<DocumentSet>
  <header>
    <description xmlns:x="http://www.w3.org/1999/xhtml">
      There is a problem with this <x:tt>DocumentSet</x:tt>:
      The namespace declaration is <x:b>missing</x:b>!
    </description>
  </header>
</DocumentSet>
```

Finally, Example 1.3 demonstrates how to create customised metadata elements. In the example, we create our own `version` element, and a `reviewers` element that lists the developers who have reviewed a particular consistency rule. All the elements are in the metadata namespace, which is bound to the prefix `meta:` at the header element. In files where many extended headers are used, the prefix could be bound at the root element to avoid having to rebind it.

Example 1.3. Customised Metadata

```
<consistencyrule>
  <header xmlns:meta="http://www.xlinkit.com/Metadata/5.0">
    <meta:version>1.0</meta:version>
    <meta:reviewers>
      <meta:reviewer>Wolfgang Emmerich</meta:reviewer>
      <meta:reviewer>Anthony Finkelstein</meta:reviewer>
    </meta:reviewers>
  </header>
  ...
</consistencyrule>
```

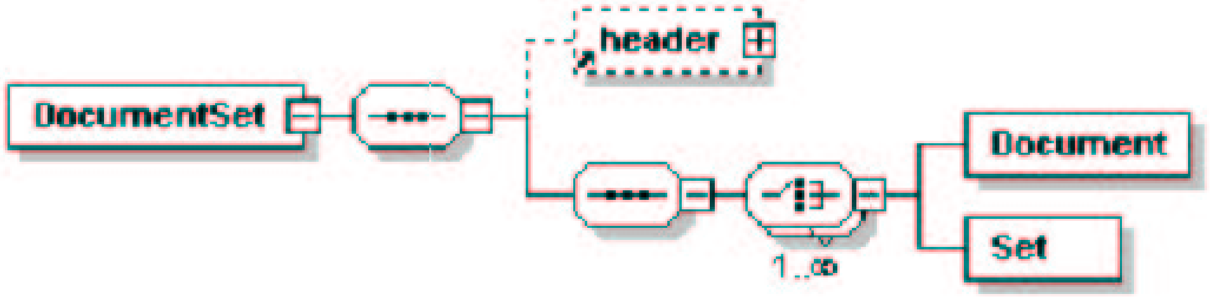
4.2 Document Sets

4.2.1 Definition

Document sets are xlinkit's way of structuring document input. A document in this case means a collection of structured or semi-structured data, but does not necessarily imply storage in a traditional document format. The purpose of a document set is thus to abstract from underlying data storage formats, drawing if necessary on the support of fetcher plugins to translate them into a DOM tree that can be used for checking.

The namespace for document sets is `http://www.xlinkit.com/DocumentSet/5.0`. Figure 2.1 shows a graphical view of the document set schema, leaving the metadata `<header>` collapsed.

Figure 2.1. Document set schema



`header`: A document set can contain the usual metadata, as defined in the common mechanisms. No additional meanings are defined for the metadata in the context of a document set and it can be used freely.

`Document`: The `Document` command imports a document into the document set for checking. It requires an attribute, `href` as a reference - as defined in common mechanisms - to the document. The second, optional, attribute `fetcher` is a string identifying the Fetcher to be used to retrieve the document. By default, documents are loaded using the `FileFetcher`, which is essentially an XML parser. If XML documents are to be loaded, therefore, no fetcher parameter has to be specified.

If alternative fetcher parameters are passed, for example `JDBCFetcher` for loading a database table, two conditions must be met: a fetcher class matching this string has to be registered with xlinkit, and the format of the `Reference` has to be valid for this fetcher (e.g. the JDBCFetcher expects an SQL query rather than a file name).

`Set`: This can be used to import further document sets into the set. The only attribute is `href`, which must point to a valid xlinkit document set. Using this mechanism, it is possible to build up hierarchies of document sets that will be flattened and loaded during a check.

4.2.2 Examples

Example 2.1 shows a simple document set that includes XML files from two different URLS for a check.

Example 2.1. Simple document set

```
<DocumentSet xmlns="http://www.xlinkit.com/DocumentSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/DocumentSet/5.0 DocumentSet.xsd">

  <Document href="http://www.xlinkit.com/Example/documentA.xml"/>
  <Document href="http://www.systemwire.com/Example/documentB.xml"/>
</DocumentSet>
```

Example 2.2 gives a slightly more complex example that specifies some metadata, includes an XML document, imports another document set, and uses a proprietary plugin, called `JavaFetcher` to load a Java source file and make it available for checking. This fetcher must of course be implemented and registered with xlinkit before it can be referred to in a document set like this.

Example 2.2. Document set with custom fetcher

```
<DocumentSet xmlns="http://www.xlinkit.com/DocumentSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/DocumentSet/5.0 DocumentSet.xsd">

  <header>
    <description>A slightly more complex example</description>
    <author>Christian Nentwich</author>
    <comment>Tested and working</comment>
  </header>
  <Document href="http://www.xlinkit.com/Example/documentA.xml"/>
  <Set href="http://www.systemwire.com/Example/MoreDocuments.xml"/>
  <Document href="HelloWorld.java" fetcher="JavaFetcher"/>
</DocumentSet>
```

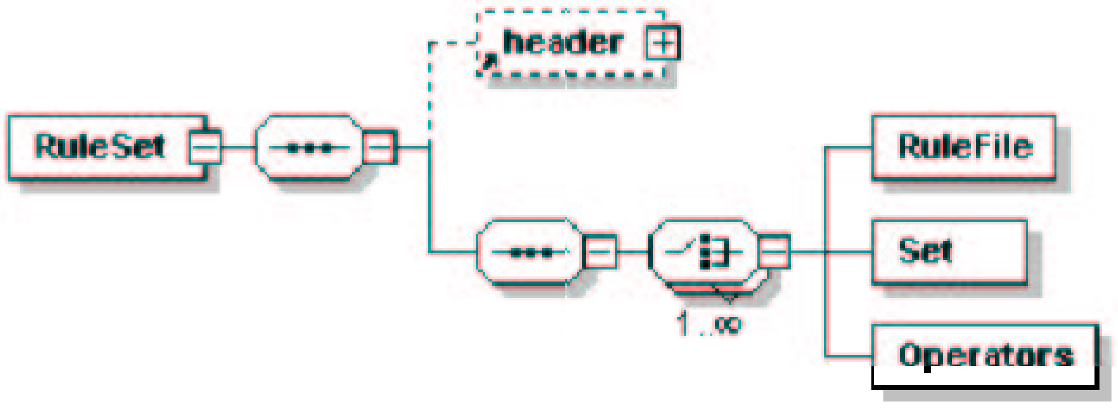
4.3 Rule Sets

4.3.1 Definition

Rule sets are xlinkit's structured consistency rule selection mechanism. They allow the free distribution and reassembly of rules, permit the selection of rules based on workflow, and assist in decoupling the rules from the documents they are applied to.

The namespace for rule sets is `http://www.xlinkit.com/RuleSet/5.0`. Figure 3.1 gives a graphical overview of the schema, leaving the metadata `header` collapsed.

Figure 3.1. Rule set schema



`RuleSet`: In addition to serving as a container element, this has a second function: any namespace prefixes bound at this element can be used in XPath queries to pick out particular rules. For more information, see the definition of `RuleFile` below and Example 3.2.

`header`: A rule set can contain the usual metadata, as defined in the common mechanisms. No additional meanings are defined for the metadata in the context of a rule set and it can be used freely.

`RuleFile`: In the current version of the language, consistency rules must be stored in the xlinkit XML format for consistency rules. The required attribute `href`is therefore a reference that denote a URL or filename.

The optional `xpath` attribute can be used to fine-tune which rules are included from a rule file that contains multiple rules. By default, that is in the absence of the parameter, all rules are included. A parameter such as `/rule:consistencyruleset/rule:consistencyrule[1]`, where `rule` has been bound to the rule file namespace at the root element, can be used to pick out the first rule of the file.

`Operators` includes an operator set into this rule set. If any of the rules in the rule set make use of operators, they have to be included here. The required `href` attribute must identify a valid operator set file.

4.3.2 Examples

Example 3.1 shows a typical rule set that includes a single rule file, picking all rules from the file for checking, and no metadata. It references the file using its filename `rule.xml`, so the file has to be present in the same directory as the rule set.

Example 3.1. Simple rule set

```
<RuleSet xmlns="http://www.xlinkit.com/RuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/RuleSet/5.0 RuleSet.xsd">

  <RuleFile href="rule.xml"/>
</RuleSet>
```

Example 3.2 shows how to refine rule selection in a rule set using XPath expressions to pick out a subset of the rules from a rule file. The expression `//rule:consistencyrule[@id='r3']` matches the consistency rule whose `id` attribute is equal to r3, anywhere in the rule file. Note how the XPath expression makes use of the `rule` namespace prefix that has previously been bound to the rule file namespace. *This is very important* - if the namespace prefix is not properly bound or left out altogether, no rules will be matched and xlinkit will return an error.

Example 3.2. Rule selection

```
<RuleSet xmlns="http://www.xlinkit.com/RuleSet/5.0"
  xmlns:rule="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/RuleSet/5.0 RuleSet.xsd">

  <RuleFile href="rule.xml" xpath="//rule:consistencyrule[@id='r3']"/>
</RuleSet>
```

Finally, Example 3.3 shows a fully-flegded rule set with metadata and operator set inclusion. It retrieves the operator set from a remote server, as well as including another additional rule set from a remote server.

Example 3.3. Operator and rule set inclusion

```
<RuleSet xmlns="http://www.xlinkit.com/RuleSet/5.0"
    xmlns:rule="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xlinkit.com/RuleSet/5.0 RuleSet.xsd">

    <header>
        <description>
            A rule set with operator inclusion
        </description>
            <author>Christian Nentwich</author>
            <comment>Tested and working</comment>
        </header>
    <Operators href="http://www.xlinkit.com/Foo/operators/Operators.xml"/>
    <RuleFile href="rule.xml"/>
    <Set href="http://www.xlinkit.com/Foo/additional-rules.xml"/>
</RuleSet>
```

4.4 Rule Files

Rule files contain the constraints that the xlinkit checker applies to documents. This chapter therefore defines not only the layout of the files, but the grammar of the xlinkit constraint language itself. The constraint language can be displayed in many different formats, in addition to its standard XML encoding. As a consequence we define the language using an abstract syntax, and provide the XML encoding of each construct after the behaviour definition.

From here on, the section is split into three parts: Sub-section 4.4.1 defines the xlinkit constraint language, Sub-section 4.4.2 defines the format of the rule files, including the elements that supplement the constraints, and Sub-section 4.4.3 gives several illustrating examples of complete consistency rule files.

4.4.1 Constraint Language

The xlinkit constraint language is a fairly simple language that is based on first order predicate logic: it allows the use of boolean connectives such as `and` and `or`, quantifiers such as `forall` and `exists` for iteration, and predicates such as `equal` for comparison.

The purpose of this section is firstly to define the abstract syntax and XML encoding of the language, and secondly to specify the behaviour of each construct in the language. The section does *not* discuss the xlinkit link generation semantics, i.e. how to create hyperlinks as a result of the evaluation of formulae in the language - instead we refer you to http://www.xlinkit.com for more information.

This section also does not define any bracketing for formulae, leaving the precedence relationship between logical connectives ambiguous. The reason for this is that bracketing belongs in the concrete syntax - for example the XML syntax is already disambiguated because it is a prefix notation, and does not require any bracketing. Should new concrete syntaxes be defined, they have to introduce bracketing on a case by case basis.

Formula

[1]	Formula	::=	Forall | Exists |	/* Quantifiers */
			And | Or | Implies | Iff | Not |	/* Logical connectives */
			Equal | Notequal | Same | True | Operator	/* Predicates */

Everything in the xlinkit language is a `Formula` - constructs that can contain subformulae, for example the logical connectives, can thus contain any other formula in the language. The only restriction on an xlinkit formula is that it *must* start with a `Forall`, as defined in Sub-section 4.4.2.

4.4.1.1 Forall

[2] Forall `::=` 'forall' Variable 'in' XPath Formula?

`Variable`: This must be a valid XPath variable identifier, as defined in the XPath standard. In addition, if the `Forall` is contained in a parent formula, the identifier must not be declared in a quantifier of one of the parent formulae. *Note:* the variable identifier does *not* include the variable reference character '$', thus `$a` is an illegal variable name whereas `a` is legal.

`XPath`: This must be a valid XPath expression. In addition, the execution of the XPath expression *must* result in a node set. Expressions that return different types of results are *illegal* and will cause a run-time error. The following XPath expressions are examples of legal expressions for `Forall` since they select node sets:

`/foo/bar` selects all elements called `<bar>` that are contained under elements called `<foo>` in any of the documents of the document set.

`/foo/@bar` selects all attribute nodes called `bar` that are attached to elements called `<foo>` in any of the documents of the document set.

And the following examples are *illegal* since they select other types of values:

`54` is illegal because it selects a number.

`substring(/foo/bar/text(),5)` is illegal because it selects a string.

`Forall`'s behaviour is defined as follows: it executes the expression contained in `XPath` on all documents of the document set. The working set of the quantifier is then defined to be the *union of all nodes* returned from *all* documents in the document set. The quantifier binds each node to the variable in turn, calling the subformula to evaluate itself given the new variable binding. It returns `true` if the subformula evaluation returns `true` for all assignments of the variable, else it returns false. Figure 4.1 specifies the behaviour in pseudocode.

The abstract syntax permits the omission of a subformula for the forall operator. If no subformula is present, the subformula will be set to True, and the quantifier will return `true` regardless of where the XPath expression is pointing.

Figure 4.1. Forall evaluation pseudo-code

```
forall (binding)
begin
    workingset=0
    for all doc in DocumentSet do
        workingset=workingset+evaluate(XPath,doc)
    done
    result=true
    for all node in workingset do
        binding=binding+(Variable,node)
        result=result &amp;&amp; Formula.evaluate(binding)
        binding=binding-(Variable,node)
    done
    return result
end
```

XML Representation

Figure 4.2 shows the XML representation for `Forall`. No attributes or elements beyond those already specified in the abstract syntax are required.

Figure 4.2. Forall XML Representation

```
<forall var="variable" in="xpath">
  Formula
</forall>
```

4.4.1.2 Exists

[3] Exists `::=` 'exists' Variable 'in' XPath Formula?

`Variable`: This must be a valid XPath variable identifier, as defined in the XPath standard. In addition, the same restrictions as for Forall apply.

`XPath`: This must be a valid XPath expression. The execution of the XPath expression *must* result in a node set. Expressions that return different types of results are *illegal* and will cause a run-time error. Please refer to Forall for examples of legal and illegal XPath expressions.

`Exists`' behaviour is defined as follows: it executes the expression contained in `XPath` on all documents of the document set. The working set of the quantifier is then defined to be the *union of all nodes* returned from *all* documents in the document set. The quantifier binds each node to the variable in turn, calling the subformula to evaluate itself given the new variable binding. It returns `true` if the subformula evaluation returns `true` for any assignment of the variable. If it is false for all assignments, it returns `false`. Figure 4.3 specifies the behaviour in pseudocode.

Similarly to Forall, the abstract syntax of `Exists` permits the omission of a subformula. If the subformula is omitted, xlinkit assumes it to be True. This is frequently very useful in practice, since it allows a test for the existence of an element without applying any predicates, i.e. we can say "element A exists". Because the subformula is True, the only way an existential quantifier with no subformula can fail is if the XPath expression does not match anything.

Figure 4.3. Exists evaluation pseudo-code

```
exists (binding)
begin
  workingset=0
  for all doc in DocumentSet do
    workingset=workingset+evaluate(XPath,doc)
  done
  result=false
  for all node in workingset do
    binding=binding+(Variable,node)
    result=result || Formula.evaluate(binding)
    binding=binding-(Variable,node)
  done
  return result
end
```

XML Representation

Figure 4.4 shows the XML representation for `Exists`. No attributes or elements beyond those already specified in the abstract syntax are required.

Figure 4.4. Exists XML Representation

```
<exists var="variable" in="xpath">
  Formula
</exists>
```

4.4.1.3 And

[4] And ::= Formula 'and' Formula

The behaviour of `And` matches its definition in classical logic: it returns true if and only if both subformulae evaluate to true, otherwise it returns false. Table 4.1 specifies the behaviour exhaustively. In the table, `FormulaA` refers to the first subformula and `FormulaB` to the second.

Table 4.1. And truth table

FormulaA	FormulaB	FormulaA 'and' FormulaB
true	True	true
true	False	false
false	True	false
false	False	false

XML Representation

Figure 4.5 shows the XML representation for `And`. No attributes or elements beyond those already specified in the abstract syntax are required. Note that the XML syntax for and is *prefix*, i.e. the connective frames its subformulae, whereas the abstract syntax is *infix*, i.e. the connective is located between its parameters. In the XML syntax, note that still *exactly two* subformulae must be present.

Figure 4.5. And XML Representation

```
<and>
  Formula
  Formula
</and>
```

4.4.1.4 Or

[5] Or `::=` Formula 'or' Formula

The behaviour of `Or` also matches its definition in classical logic: it returns true if either subformula evaluates to true, otherwise it returns false. Table 4.2 specifies the behaviour exhaustively. In the table, `FormulaA` refers to the first subformula and `FormulaB` to the second.

Table 4.2. Or truth table

FormulaA	FormulaB	FormulaA 'or' FormulaB
true	True	true
true	False	true
false	True	true
false	False	false

XML Representation

Figure 4.6 shows the XML representation for `Or`. No attributes or elements beyond those already specified in the abstract syntax are required. The same comments as for the And XML encoding apply.

Figure 4.6. Or XML Representation

```
<or>
  Formula
  Formula
</or>
```

4.4.1.5 Implies

[6] Implies `::=` Formula 'implies' Formula

We use the classical definition of `Implies`, which is summarised in Table 4.3. In the table, `FormulaA` refers to the first subformula and `FormulaB` to the second. The only way an implication can be `false` if the first subformula is `true` and the second one is `false`, for example "I am strong implies I can lift anything".

Note in particular that if `FormulaA` is false, the outcome of evaluating `FormulaB` is irrelevant - the result will be `true`. This is taking the classical view that anything may follow from a false premise. For example, the sentence "If five divides eleven, I will be king" is considered true even if I will never be king.

Table 4.3. Implies truth table

FormulaA	FormulaB	FormulaA 'implies' FormulaB
true	True	true
true	False	false
false	True	true
false	False	true

XML Representation

Figure 4.7 shows the XML representation for `Implies`. No attributes or elements beyond those already specified in the abstract syntax are required. The same comments as for the And XML encoding apply.

Figure 4.7. Implies XML Representation

```
<implies>
  Formula
  Formula
</implies>
```

4.4.1.6 Iff

[7] Iff `::=` Formula 'iff' Formula

`Iff` is a traditional shorthand for “if and only if”. It is a convenient way of expression a two-way implication. `FormulaA iff FormulaB` is thus equivalent to `(FormulaA implies FormulaB) and (FormulaB implies FormulaA)`.

Table 4.4 gives the truth table for `iff`. It returns true if the result of evaluating both subformulae is equal, otherwise it returns false.

Table 4.4. Iff truth table

FormulaA	FormulaB	FormulaA 'iff' FormulaB
true	True	true
true	False	false
false	True	false
false	False	true

XML Representation

Figure 4.8 shows the XML representation for `Iff`. No attributes or elements beyond those already specified in the abstract syntax are required. The same comments as for the And XML encoding apply.

Figure 4.8. Iff XML Representation

```
<iff>
  Formula
  Formula
</iff>
```

4.4.1.7 Not

[8] Not `::=` 'not' Formula

`Not` computes the logical negation of its subformula. Table 4.5 gives the truth table.

Table 4.5. Not truth table

Formula	'not' Formula
true	False
false	True

XML Representation

Figure 4.9 shows the XML representation for `Not`. No attributes or elements beyond those already specified in the abstract syntax are required.

Figure 4.9. Not XML Representation

```
<not>
  Formula
</not>
```

4.4.1.8 Equal

[9] Equal `::=` XPath '=' XPath

`Equal` compares two *sets* of values for equality. The two sets are constructed by evaluating the two parameter XPath expressions. The two XPath expressions *must* either return primitive values, like strings or numbers, or they must be relative to a variable. Thus, the following expressions are legal, `'foo'`, `5`, `$x/name/text()`, while the following is illegal `/name/text()` (absolute expression).

Value sets

Before defining the behaviour of `Equal` we will define how it evaluates its XPath expressions to produce *value sets*. All entries a value set must be of the same type, determining the type of the set. A value set can thus be:

a set of *strings*

a set of *booleans*

a set of *numbers* (double)

In order to construct the set of values, the two XPath expressions are evaluated and their results converted:

Expressions that result in strings directly are converted into a set of size 1, containing the string. For example, the expression `'foo'` becomes the set `{'foo'}`, the expression `substring($x/name/text(),2)` may become `{'ristian'}`. The XPath 1.0 specification gives further details on which functions return strings.

Expressions that result in numbers directly are converted into a set of nubmers of size 1, containing the number. For example, `20` becomes the set `{20}` and `count($x/b)`may become `{2}`.

Similarly, expressions that result in booleans directly are converted into a set of booleans of size 1, containing only the boolean value. For example, `true` becomes `{true}` and `$x/value > 5` may become `{false}`.

What remains are expressions that produce *node sets*, i.e. expressions like `$x/name/text()`, which produces a list of text nodes, `$x/@name`, which produces a list of attribute nodes, and `$x/name`, which produces a list of element nodes.

Any expression that produces a node set will be converted to a *set of strings*, using the following conversion rules: for every node `n` in the set.

If `n` is a *text node*, *attribute node*, *comment node* or *CDATA node*, the *value* of the node is added to the result set.

If `n` is an *element node*, all text node children of `n` are added to the result set. Thus, an expressions like `$x/name` will be equivalent to `$x/name/text()`. This is equivalent to the behaviour of XSLT. *CAUTION:* While this behaviour is guaranteed by the xlinkit checker, we discourage the use of this shorthand, and recommend the use of the full `text()`

syntax - omission of the explicit text syntax may prevent tools that statically analyse formulae, for example optimisers or repair action generators, from working properly.

If `n` is any other type of node, for example a document node, a runtime error will occur.

We will now go through some examples of the value sets that would be generated from typical XPath expressions. Example 4.1 gives the sample data we will use for the expressions. We will further assume that the variable `$x` has been bound to the `<catalogue>` root element.

Example 4.1. Sample document

```
<catalogue>
  <product>
    <name>Evaluation</name>
  </product>
  <product>
    <name>FpML Validator</name>
  </product>
  <product>
    <name>UML Validator</name>
  </product>
  <number>4</number>
</catalogue>
```

Evaluating `count($x/product)` will result in the set of integers `{3}`. `$x/product/name/text()` will result in the set of strings, `{"Evaluation", "FpML Validator", "UML Validator"}`, `$x/product[2]/name/text()` will result in `{"FpML Validator"}` and `count($x/product) > 5` will result in the boolean set `{false}`.

Casting

Since `Equal` and other predicates that rely on value sets have to compare two or more sets for equality, casting rules have to be defined for those cases where the sets are of a different type. Table 4.6 shows what type both sets will be converted to given their own types - i.e. in practice one set will remain unchanged and the other downcast.

Table 4.6. Value set casting rules

Set1	Set2	Base type
Number	Boolean	Boolean
Number	Strings	Strings
Boolean	Strings	Strings

The base type for all sets is thus a set of strings. The conversion rules for converting between set types are as follows:

Number to Boolean: The first entry in the set of numbers is compared to `0`. If it is non-zero, a boolean set of size 1 with the value `{true}` is returned, else the set `{false} is returned.` For example, the set `{5}` becomes `{true}` and the set `{0}` becomes `{false}`.

Number to Strings: The first number in the set is directly converted into a string and placed into a new set, e.g. `{5}` becomes `{"5"}`.

Boolean to Strings: The first boolean in the set is directly converted into the string `"true"` or `"false"`, depending on its value, and placed into a new set, e.g. `{true}` becomes `{"true"}`.

Using these casting rules it is now possible to compare sets of different types. Taking the data from Example 4.1, we can for example compare `count($x/product)`, a number, to `$x/number/text()`, a string, and the result will be `false`.

Behaviour

The behaviour of `Equal` is quite straightforward, it checks whether two value sets contain exactly the same values. Because it is dealing with sets, however, the order in which the values appear is irrelevant. The pseudocode, assuming the value sets have been constructed and downcast beforehand, is given below:

```
equal (set1,set2)
begin
    if (set1.size != set2.size)
        return false;

    for all entries e in set1
        if (!set2.contains(e))
          return false
        set2=set2-e
    done

    return true
end
```

Given this behaviour, the sets `{"foo"}` and `{"foo"}` are equal, the sets `{"foo","bar"}` and `{"foo"}` are not equal, `{"foo","bar"}` and `{"bar","foo"}` are equal and `{5}` and `{true}` are equal.

XML Representation

Figure 4.10 shows the XML representation for `Equal`. The two XPath expressions must be passed as attributes `op1` and `op2`.

Figure 4.10. Equal XML Representation

```
<equal op1="xpath" op2="xpath"/>
```

4.4.1.9 Notequal

[10] Notequal `::=` XPath '!=' XPath

`Notequal` is really a convenience mechanism, since it can be equivalently represented using the existing `Not` and `Equal` constructs. For details on the restrictions on the XPath expressions, their evaluation, and downcasting rules, please refer to the description of `Equal` in Sub-section 4.4.1.8.

Behaviour

For completeness, the behaviour of `Notequal` is given below in pseudocode. It makes the same comparison as `Equal`, comparing two sets for equality regarless of order, and returns the opposite result.

```
notequal (set1,set2)
begin
    if (set1.size != set2.size)
        return true;

    for all entries e in set1
        if (!set2.contains(e))
          return true
        set2=set2-e
    done

    return false
end
```

XML Representation

Figure 4.11 shows the XML representation for `Notequal`. The two XPath expressions must be passed as attributes `op1` and `op2`.

Figure 4.11. Notequal XML Representation

```
<notequal op1="xpath" op2="xpath"/>
```

4.4.1.10 Same

[11] Same `::=` VariableRef '==' VariableRef

`Same` takes as its parameters two references to variables that must have been bound in a parent formula. It then checks whether the two variables point to exactly the same node - it does *not* compare them by value like `Equal` does. Take

Example 4.2 below: assume that `$x` points to the first `<product>` element, and `$y` to the second. Then `$x == $y` is `false` while `$x = $y` is `true`.

Example 4.2. "Same" example

```
<catalogue>
  <product>xlinkit</product>
  <product>xlinkit</product>
</catalogue>
```

`Same` is most useful in uniqueness checks, for example if one wishes to say “if A and B have the same value, then they must be the same element”.

XML Representation

Figure 4.12 shows the XML representation for `Same`. The two variable references must be passed as attributes `op1` and `op2`.

Figure 4.12. Same XML Representation

```
<same op1="$var" op2="$var"/>
```

4.4.1.11 True

[12]	True	`::=`	'True'	

`True` is different from the other formulae in that it *must not* be used explicitly as a subformula. Instead, it is appended automatically as a child to quantifiers that do not specify a subformula. Please refer to Sub-section 4.4.1.2 for details.

There is no XML encoding for `True`.

4.4.1.12 Operator

[13]	Operator	`::=`	'Operator' String Param*	
[14]	Param	`::=`	'Param' String String	/* Name and Value */

`Operator`: An operator behaves like any other predicate, it takes a number of parameters and returns `true` or `false`. The `String` has to be a valid name for the operator: The rule set in which this rule file is included *must* include an operator set that provides a definition for this operator. Furthermore, the name of the operator must be prefixed with the name of the operator set from which it is loaded. Thus, in order to use the operator `isPrime` in operator set `math`, `math:isPrime` has to be used as the name.

`Param`: The parameters passed as arguments must match those in the operator definition of the operator set both in name, and in order. The value passed as a parameter is a string, but will be converted into the format expected by the operator, for example by treating it as an XPath expression and evaluating it. Please refer to Section 5.1 for details on parameter conversion.

Please refer to Chapter 5 for further details on defining operators.

XML Representation

Figure 4.13 shows the XML representation for `Operator`. The parameters are passed as subelements.

Figure 4.13. Operator XML Representation

```
<operator name="prefix:name">
  <param name="param1" value="val"/>
  <param name="param2" value="val"/>
  ...
</operator>
```

4.4.1.13 Complete Grammar

xlinkit Constraint Language

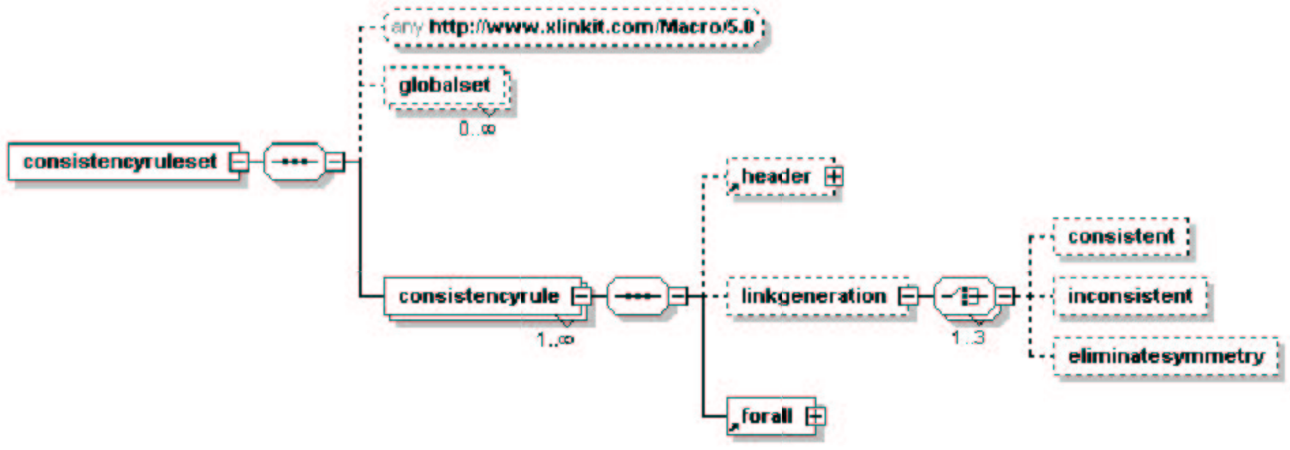
[1]	Formula	::=	Forall | Exists |	/* Quantifiers */
			And | Or | Implies | Iff | Not |	/* Logical connectives */
			Equal | Notequal | Same | True | Operator	/* Predicates */
[2]	Forall	::=	'forall' Variable 'in' XPath Formula?	
[3]	Exists	::=	'exists' Variable 'in' XPath Formula?	
[4]	And	::=	Formula 'and' Formula	
[5]	Or	::=	Formula 'or' Formula	
[6]	Implies	::=	Formula 'implies' Formula	
[7]	Iff	::=	Formula 'iff' Formula	
[8]	Not	::=	'not' Formula	
[9]	Equal	::=	XPath '=' XPath	
[10]	Notequal	::=	XPath '!=' XPath	
[11]	Same	::=	VariableRef '==' VariableRef	
[12]	True	::=	'True'	
[13]	Operator	::=	'Operator' String Param*	
[14]	Param	::=	'Param' String String	/* Name and Value */

4.5 Rule file

A rule file, called a *consistency rule set* for historical reasons - but not to be confused with a *rule set*, consists of namespace declarations, global set declarations, macro inclusion commands, and consistency rules expressed in the xlinkit constraint language. The consistency rules themselves provide additional mechanisms on top of the constraint language, such as metadata.

The namespace for rule sets is `http://www.xlinkit.com/ConsistencyRuleSet/5.0`. Figure 4.14 gives a graphical overview of the schema, leaving the metadata `header` collapsed.

Figure 4.14. Rule file schema



The root element of a rule file is `consistencyruleset`. It does not contain any attributes, but may contain namespace definitions. Any prefix that is bound to a URI at the `consistencyruleset` element can be used inside XPath expressions in constraints. You can find an illustration of this in Example 4.5.

`macro:include` - the rule file may contain macro inclusion elements. The schema allows any element from the `http://www.xlinkit.com/Macro/5.0` namespace, which has been bound to the prefix `macro` in this case, to occur here, but at the moment only the `include` element is processed. The element must provide an `href` attribute, which is a reference to the macro definition file to be applied to the rules. Please refer to Chapter 6 for more details. (*Note:* the current version of xlinkit limits the macro inclusion mechanism to including only a single macro file, this restriction will probably be removed in future versions).

4.5.1 Global sets

A `globalset` is a node set that is created before all rules are evaluated and bound to a certain variable name. The variable can then be used in any of the consistency rules. The element takes two parameters: `id`, which defines the name of the variable and `xpath`, which gives the path to evaluate in order to obtain the value of the variable.

The XPath expression passed as a parameter must have the same characteristics as those for the quantifiers - i.e. it must evaluate to a node set -, please refer to Section 4.1.1 for details. Note that the `Variable` is a variable definition, not a reference, so it must not include the `$` character.

A global set is not only shared between all rules inside one particular rule file, but applies to *all* rules included in a rule set. It is thus possible to refer to elements in documents using a symbolic name such as `$classes` instead of `//Foundation.Core.Class`. Because global sets are shared, whenever a global set is declared under the same `id` in two different files that have been included in the same rule set the `xpath` attribute of the declaration must also match, otherwise the rule set is invalid.

See Example 4.6 for an example of how to use a global set.

4.5.2 Consistency rules

As the schema in Figure 4.14 shows, a consistency rule consists of three parts: a `header` that defines the metadata, a `linkgeneration` element that controls the way xlinkit produces hyperlinks, and the formula itself, which *must* start with a Forall.

Every consistency rule *must* provide an `id` attribute for unique identification in the rule. According to the XML specification, this identifier has to be unique inside the rule file. We recommend that identifiers are made unique within a larger context, for example a set of rule files for a specific markup language.

The `header` contents can be used freely to associate any metadata with the rule. The `project` element, however, is reserved to take up a special role in future versions of xlinkit. It will be used together with the `id` attribute to uniquely identify rules and make it possible to identify rules that have been spread over several files using a `(project,id)` pair. The element may become mandatory in future versions of xlinkit.

4.5.2.1 Link Generation

The optional `linkgeneration` element may contain a number of directives that control the diagnostic output produced by xlinkit: `consistent` controls whether elements that are obey a constraint should be explicitly linked and `inconsistent` controls if elements that violate a constraint should be linked. Both elements are optional, and the default is that consistent is `off` and inconsistent is `on`. Both elements have a `status` attribute that can take the values `on` or `off`.

`eliminatesymmetry` can be used to instruct xlinkit to remove any pairs links whose locators point to the same elements but are permutations of one another. For example, if a pair of links points to `(A,B)` and another to `(B,A)`, the second link will be removed. This functionality is useful in some situations where pairs of elements are compared for equality and inconsistencies cause the elements to be linked twice (because A is inconsistent with B, but B is also inconsistent with A). This behaviour is rare, but does occur with some uniqueness checks. The element takes a `status` attribute whose values can be `on` or `off` - the default is `off`. Turning this function on when it is not necessary will not cause a difference in the result, but will introduce a runtime overhead of `n*n` where `n` is the number of links in a linkbase.

4.5.3 Examples

Example 4.3 shows a very simple constraint that specifies that “for every element A there must be an attribute att”. This rule can now be included in a rule set and checked against a document set that includes any number of files, some of which may have root elements called `A`. The rule does not define any metadata or link generation commands. Consistent cases where the attribute is present will therefore not be specially identified, and inconsistent cases will be linked. The XPath expressions also do not use any namespace prefixes, so the element `A` will only be matched if it is contained in a file without a default namespace.

Example 4.3. Simple Constraint

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd">

  <consistencyrule id="r1">
    <forall var="x" in="/A">
      <exists var="a" in="$x/att"/>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

Example 4.4 shows a slightly more complex constraint, taken from xlinkit's Wilbur's Bike Shop example, that expresses a relationship between two different data formats. It says that for every file with an `Advert` root element, there has to be a `Product` element somewhere else that matches its name.

When applying this rule, it is now possible to feed several files into xlinkit, and all those that have an `Advert` root elements will be checked. Similarly, multiple catalogues could possibly supplied and be checked against.

Note that we have turned on consistent link generation. This will cause elements that obey the constraint to be linked, in this case all adverts will be linked to the correct entry in the catalogue.

Example 4.4. Checking Multiple Files

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd">

  <consistencyrule id="r1">
    <header>
      <description>Every advert must be in the catalogue</description>
      <project>Wilburs</project>
    </header>
    <linkgeneration>
      <consistent status="on"/>
    </linkgeneration>
    <forall var="x" in="/Advert">
      <exists var="y" in="/Catalogue/Product">
        <equal op1="$x/name/text()" op2="$y/name/text()"/>
      </exists>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

Example 4.5 shows the same constraint as the previous example, but assumes that the catalogue elements have been placed in a file that uses `http://www.xlinkit.com/Example/Bike/Catalogue` as the default namespace. If we used the previous constraint on such a catalogue, no elements would be matched and inconsistencies would be detected. Instead, we bind the new namespace to the prefix `cat:` at the root element and make use of that prefix in the XPath expressions.

Example 4.5. Namespaces in XPaths

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd"
  xmlns:cat="http://www.xlinkit.com/Example/Bike/Catalogue">

  <consistencyrule id="r1">
    <forall var="x" in="/Advert">
      <exists var="y" in="/cat:Catalogue/cat:Product">
        <equal op1="$x/name/text()" op2="$y/cat:name/text()"/>
      </exists>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

Example 4.6 demonstrates the use of a global set. The set is defined using the XPath expression from the previous example. The rule itself has now become slightly easier to read - and could become significantly easier to read if more complex expressions were involved. The set `$products` is now been defined globally and can be used in any other rule included in the same rule set as the one in the example.

Example 4.6. Global Sets

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd">

  <globalset id="products" xpath="/Catalogue/Product"/>

  <consistencyrule id="r1">
    <forall var="x" in="/Advert">
      <exists var="y" in="$products">
        <equal op1="$x/name/text()" op2="$y/name/text()"/>
      </exists>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

The final example, Example 4.7 demonstrates an operator invocation. Please refer to Section 5.3 to see the definition of the operator used in this example.

Example 4.7. Operator Invocation

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd">

  <consistencyrule id="r1">
    <forall var="x" in="/Advert">
      <operator name="test:isGreater">
        <param name="stringA" value="$x/name/text()"/>
        <param name="stringB" value="$x/shortname/text()"/>
      </operator>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

4.6 Operators

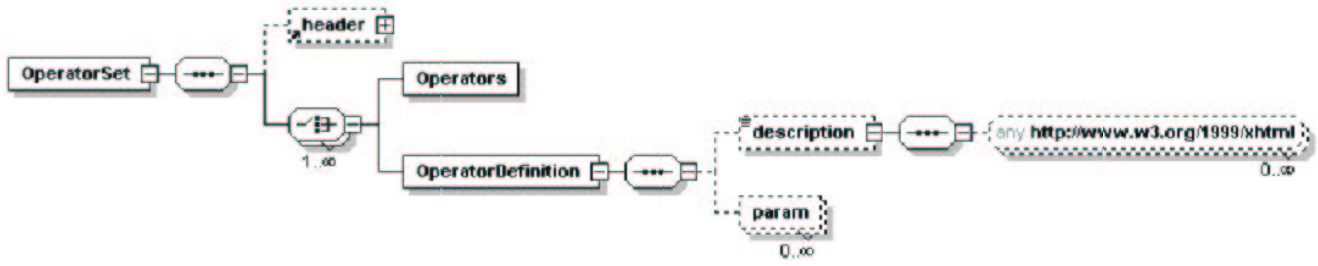
Operators in xlinkit are *plug-in predicates*. Just as the standard xlinkit predicates like `Equal`, they take a number of parameters and return a truth value as a result. In order to create an operator for use in a formula, it has to be defined in an operator set (Section 5.1), and implemented in an implementation language (Section 5.2).

4.6.1 Operator Set

The purpose of an operator set is to define an interface to plug-in operators that are to be used in formulae. It includes metadata, operator interface definitions, and a reference to where the implementation is stored. Operator sets, like the other input mechanisms, can contain further operator sets, permitting the construction and reuse of collections of operators.

The namespace for rule sets is `http://www.xlinkit.com/OperatorSet/5.0`. Figure 5.1 gives a graphical overview of the schema, leaving the metadata `header` collapsed.

Figure 5.1. Operator set schema



`OperatorSet`: This serves as a container element for the operators. It has a *required* attributed called `name`. The `name` defines the prefix that will be applied to all operators in this set in order to avoid name clashes between different sets. Thus the operator `isPrime` in the set whose name is `math` will become `math:isPrime`.

The second required attribute is `impl`, which must contain a reference to an implementation file. The reference can be a URL or file name. See Section 5.2 for more details on implementation files.

`Implementation` must be a valid reference, URL or file, to an operator implementation file. See Section 5.2 for details of implementation files.

`header`: Operator sets can contain the usual metadata, as defined in the common mechanisms. No additional meanings are defined for the metadata in the context of an operator set and it can be used freely.

`Operators` can be used to include a further operator set. There is only one attribute, `href`, which is a reference to the set to be included.

`OperatorDefinition`: This element defines an interface for an operator that has been implemented in the referenced implementation file. The required attribute `name` defines the name of the operator. This name *must* match the name of a function in the implementation file, otherwise the operator set is invalid.

The optional `description` element is defined in exactly the same way as the `description` element in the metadata. It can contain a mixture of text and XHTML elements, provided the elements are in the XHTML namespace. See Section 1.2 for details on how to use this element.

`param`: An operator definition can take zero or more parameters. Each `param` element has two required attributes, `name` and `type`. The name of the parameter can be chosen freely and does not have to match the name of the parameter in the implementation file (however when the operator is referred to in a rule, the name of the parameter in the rule *does* have to match that in the operator set).

The `type` attribute has to be one of the following: `int`, `string`, `nodeList` or `node`. The type of the parameter affects how any values passed in the operator invocation are treated and passed on to the operator implementation. See Section 5.2 for details.

4.6.2 Operator Implementation

An operator implementation file contains the actual implementation of operators referenced in the operator set. The contents of the file are specific to which programming language is being used. In the current version of xlinkit, the only supported language is ECMAScript.

Regardless of which language is used, the implementation file must provide one function for each operator referenced in the operator set. The function will take several parameters that depend on the language, but must return the languages native boolean type.

4.6.2.1 Parameter types

Section 5.1 sets out the possible types for operator parameters: `int`, `string`, `nodeList` or `node`. These types define how parameter values are treated before being passed to the operator invocation:

`int`: The parameter is treated as an integer and converted to the implementation specific integer type. If this conversion is impossible, a run-time error occurs.

`string`: The parameter is converted to the implementation specific string type, and passed on to the implementation as it is.

`nodeList`: The parameter is interpreted as an XPath expression. The expression is evaluated and must result in a node set. This node set is then passed on to the implementation.

`node`: The parameter is interpreted as an XPath expression. The expression is evaluated and must result in a node set with exactly one node in it. This node is then passed on to the implementation.

NOTE: If any XPath expressions are to be evaluated in an argument and passed to the operator, they must be passed as a `nodeList` or `node`, *not* as `string`. For example, passing the expression `/foo/text()` as a string will result in the operator being passed the literal string `"/foo/text()"`. If it is passed as a `node`, the operator will be passed the DOM text node contained in `/foo` and can use the `getNodeValue` method defined by the DOM to get its string value.

4.6.2.2 ECMAScript Implementation

An ECMAScript operator implementation file should simply contain a number of functions that hold the same name as the operators defined in the operator set. Each function must take exactly the same number of parameters as defined in the operator set, and must return `true` or `false` from all possible execution flows. The implementation file may contain additional functions to be used as helper functions, which may return any type.

Type mapping:

`int` parameters are converted into native ECMAScript `ints`.

`string` parameters are converted into native ECMAScript `string` objects. (Caution: these are not the same as Java `String` objects. Please consult your manual or the ECMAScript specification if you need further information).

`nodeList` parameters are passed as Java `Nodelist` classes. `Nodelist` is a class defined in the Document Object Model (DOM) and can be found in the `org.w3c.dom` package of most XML parsers.

`node` parameters are passed as Java `Node` classes. `Node` is also a class defined in the DOM and can be found in the `org.w3c.dom` package of most XML parsers.

4.6.3 Example

Example 5.1 shows an operator set that defines a new operator that can be used to check if its first parameter is a longer string that its second. Because we want to apply the operator using XPath expressions, we have to pass the parameters as `nodes` - they will point to `text` nodes when we make use of the operator.

Because we assigned the name `test` to the set, all operators in the set will have to be prefixed `test:` when they are invoked.

Example 5.1. OperatorSet

```
<OperatorSet name="test" impl="test.es"
  xmlns="http://www.xlinkit.com/OperatorSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/OperatorSet/5.0 OperatorSet.xsd">

  <OperatorDefinition name="isGreater">
    <param name="stringA" type="node"/>
    <param name="stringB" type="node"/>
  </OperatorDefinition>
</OperatorSet>
```

Example 5.2 shows the matching ECMAScript implementation for the operator. It uses the `getNodeValue` method of the two parameters, which must be `Node` objects, to retrieve the text value, and converts them into a Java string. It then returns the result of comparing the strings.

Example 5.2. Operator Implementation

```
function isGreater(stringA, stringB) {
  sA=new java.lang.String(stringA.getNodeValue());
  sB=new java.lang.String(stringB.getNodeValue());

  return ( sA.compareTo(sB) > 0 );
}
```

It would now be possible to invoke the operator in a rule file as in Example 5.3, assuming that `$x` has been bound by a parent formula.

Example 5.3. Operator Invocation

```
<operator name="test:isGreater">
  <param name="stringA" value="$x/foo/text()"/>
  <param name="stringA" value="$x/bar/text()"/>
</operator>
```

4.7 Macros

Macros are xlinkit's preprocessing mechanism for consistency rules. They allow the parameterization of frequently used formulae, which increases reuse. They also make rules easier to read since they can be used to replace complex formulae with a simple macro invocation.

Example 6.1 is a rule that will serve as a motivating example throughout this chapter. It expresses the constraint that all `Product` elements have to have a unique name within a `Catalogue` element. This kind of uniqueness check arises quite frequently and it is tedious to define precisely every time - it is thus a good candidate for replacement with a macro.

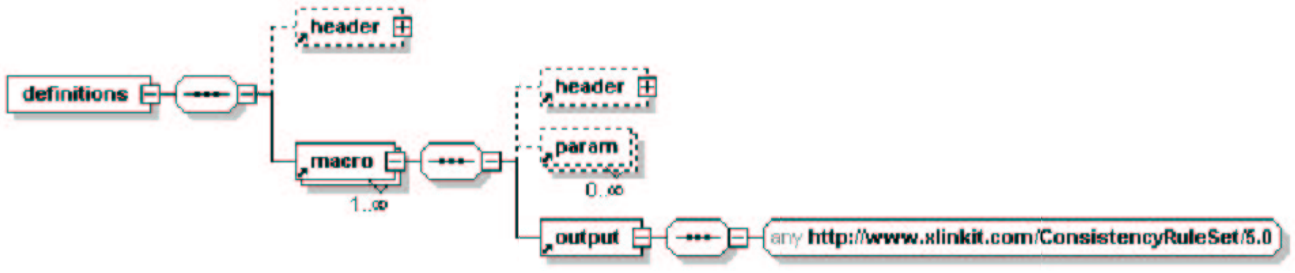
Example 6.1. Sample Rule without Macro

```
<forall var="c" in="/Catalogue">
  <forall var="x" in="$c/Product">
    <forall var="y" in="$c/Product">
      <implies>
        <equal op1="$x/name/text()" op2="$y/name/text()"/>
        <same op1="$x" op2="$y"/>
      </implies>
    </forall>
  </forall>
</forall>
```

4.7.1 Macro Definition

Macros have to be defined in a macro definition file before they can be used. Figure 6.1 shows a graphical representation of the schema for macro definition files.

Figure 6.1. Macro Definition Schema



The `definitions` root element may contain the usual metadata contained in `header`. No special meaning is defined for the metadata and it can be used freely.

`macro`: A definition file must contain at least one `macro` element. The element has a *required* attributed `name` that defines the name of the macro. The name has to be unique within the definition file. Individual macros may also contain the optional `header` element for metadata declaration - again, no special meaning is defined and the metadata may be used freely.

`param`: A macro may take zero or more parameters that can be referred to in XPath expression inside the `output`. Each `param` element has a mandatory `name` attribute that must be unique within the macro.

`output` contains the formula the macro invocation element will be replaced with. The schema allows any element from the consistency rule namespace to occur here, but in practice only formulae may be used otherwise a run-time error will occur.

Formulae in the output may make reference within their XPath attributes to the formally defined parameters of the macro using the notation `{$paramname}`. A parameter referenced using `{$paramname}` will be replaced at macro invocation using the value passed as the parameter `paramname`. See Section 6.2 for details.

4.7.1.1 Example

In this example, we take the fragment of the formula in Example 6.1 that expresses the actual uniqueness criterion, remove the actual XPath expressions and insert parameter references instead. We use the parameter `list` to refer to the set of elements that we wish to compare, and `identifier` for the relative path that we will use to compare the elements for equality.

Note that we bind the default name space to the rule namespace at the `output` element, so that we can use the formula elements without further prefixes. We reference the `list` parameter using `{$list}` in the forall expression. When the macro is invoked, this parameter reference will be replaced with the actual expression. To avoid any name clashes that might arise when inserting the macro into a parent formula, we use `macrox` and `macroy` as our internal variable names, but any variable name is allowed.

Example 6.2. Macro Definition

```
<macro:definitions xmlns:macro="http://www.xlinkit.com/Macro/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/Macro/5.0 Macro.xsd">

  <macro:macro name="unique">
    <macro:param name="list"/>
    <macro:param name="identifier"/>
    <macro:output xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0">
      <forall var="macrox" in="{$list}">
        <forall var="macroy" in="{$list}">
          <implies>
    <equal op1="$macrox/{$identifier}" op2="$macroy/{$identifier}"/>
    <same op1="$macrox" op2="$macroy"/>
  </implies>
        </forall>
      </forall>
    </macro:output>
  </macro:macro>
</macro:definitions>
```

4.7.2 Macro Inclusion and Processing

Macros may be included in a rule file using the `include` element, which is also contained in the macro namespace. Section 4.2 specifies where the element may appear in a rule file.

Once a macro file has been included, macro invocations may appear as a subformula wherever any other formula may appear. Macro invocation is achieved by inserting an element with the name of the macro and a prefix bound to the macro namespaces as a subformula. For example, if we bind `macro` to `http://www.xlinkit.com/Macro/5.0`, we can then include the macro `unique` using the element `macro:unique`.

Parameters to macros are treated similarly: all parameters become attributes, with the same name, of the macro invocation element. Thus, in our example we would have to specify the attributes `list` and `identifier`. The values of these attributes are then used to replace the parameter references in the macro. When the replacement is complete, the instantiated macro with the actual attribute values replaces the macro invocation element in the rule file.

When all macro invocation elements have been replaced with macro instantiation, and the macro inclusion element has been removed, macro processing is complete and the rule file is loaded as normal.

4.7.2.1 Example

Example 6.3 shows the rule file Example 6.1 rewritten using the macro definition from Example 6.2. The `macro:` prefix is bound to the correct namespace URL at the root element. We then use `macro:include` to include the macro file, and replace the uniqueness formula with a macro invocation. The name of the macro invocation elements matches the name of the macro in the definition file. The parameters `list` and `identifier` are passed as attributes to the macro.

Example 6.3. Macro Invocation

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd"
  xmlns:macro="http://www.xlinkit.com/Macro/5.0">

  <macro:include href="macrodef.xml"/>

  <consistencyrule id="r1">
    <forall var="c" in="/Catalogue">
      <macro:unique list="$c/Product" identifier="name/text()"/>
    /forall>
  </consistencyrule>
</consistencyruleset>
```

For completeness, Example 6.4 shows the consistency rule file after macro processing: the macro inclusion has been removed and the macro formula has been inserted, with the actual values replacing the formal parameters. The file is now ready to be loaded and processed by xlinkit as any other rule file.

Example 6.4. Processed Rule File

```
<consistencyruleset xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyruleset.xsd"
  xmlns:macro="http://www.xlinkit.com/Macro/5.0">

  <consistencyrule id="r1">
    <forall var="c" in="/Catalogue">
      <forall var="macrox" in="$c/Product">
        <forall var="macroy" in="$c/Product">
          <implies>
             <equal op1="$macrox/name/text()" op2="$macroy/name/text()"/>
             <same op1="$macrox" op2="$macroy"/>
          </implies>
        </forall>
      </forall>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

5 FpML 1.0 Validation Rules

5.1 History

In Q1 2002 Steven Lord defined a number of FpML 1.0 Validation rules and circulated them within FpML. These authors than formalized these rules using the xlinkit rule language. And during the process of formalization a number of ambiguities were identified and resolved. Furthermore it was found that several rules could be subsumed into single rules thus allowing for a more concise definition. Also during the process we found rules that were missing. The set of rules given in 4.3 does not yet include these missing rules as it was considered more important to give a realistic example for how validation rules could be specified rather than attempt to be complete.

5.2 Rule development process

In order to formalize these rules we defined them in first order logic and defined them in the XML encoding of xlinkit's rule language. We then tested each of these rules using the example trade given in the FpML 1.0 standard.

The example was modified to force deliberate violations that were then to be identified by the xlinkit rule engine.

Steven Lord and Daniel Dui also evaluated the rules in a number of workshops and now agree that these are meaningful and important rules for FpML

5.3 Sample Rule Set

ID	Description
1	In swapStream: resetDates must exist if and only if a floatingRateCalculation exists in calculation
2	In swapStream: resetDates must not exist if and only if fixedRateSchedule exists in calculation.
3	In %FpML_BusinessDayAdjustments: neither businessCentersReference nor businessCenters must exist if and only if the value of businessDayConvention is 'NONE'. %FpML_BusinessDayAdjustments defines calculationPeriodDatesAdjustments, dateAdjustments, paymentDatesAdjustments, and resetDatesAdjustments.
4	In calculationPeriodDates: firstPeriodStartDate and should not equal effectiveDate.
5	In calculationPeriodDates: terminationDate and lastRegularPeriodEndDate must not be the same.

6 CalculationPeriodFrequency must divide the regular period precisely. This is the period between the following pairs of dates depending on which are present in the document:

- effectiveDate and terminationDate, if neither firstPeriodStartDate nor firstRegularPeriodStartDate nor lastRegularPeriodEndDate exist.
- firstPeriodStartDate and terminationDate, if firstPeriodStartDate exists and neither firstRegularPeriodStartDate nor lastRegularPeriodEndDate exist.
- firstRegularPeriodStartDate and terminationDate, if firstPeriodStartDate does not exist, firstRegularPeriodStartDate exists, and lastRegularPeriodEndDate does not exist.
- effectiveDate and lastRegularPeriodEndDate, if firstPeriodStartDate and firstRegularPeriodStartDate do not exist and lastRegularPeriodEndDate exists.
- firstRegularPeriodStartDate and terminationDate, if firstPeriodStartDate and firstRegularPeriodStartDate exist and lastRegularPeriodEndDate does not exist.
- firstPeriodStartDate and lastRegularPeriodEndDate, if firstPeriodStartDate exists, firstRegularPeriodStartDate does not exist and lastRegularPeriodEndDate exists.
- firstRegularPeriodStartDate and lastRegularPeriodEndDate, if firstPeriodStartDate does not exist, firstRegularPeriodStartDate exists and lastRegularPeriodEndDate exists.
- firstRegularPeriodStartDate and lastRegularPeriodEndDate, If all of firstPeriodStartDate, firstRegularPeriodStartDate, and lastRegularPeriodEndDate exists.

7 In calculationPeriodFrequency: if rollConvention is not either 'NONE' or 'SFE' then the period must be 'M' or 'Y'.

8 In PaymentFrequency and calculationPeriodFrequency: PaymentFrequency must be an integer multiple (could be 1) of the calculationPeriodFrequency.

9 In swapStream: if firstPaymentDate exists in paymentDates, it must match one of the unadjusted calculation period dates.

10 In swapstream: if lastRegularPaymentDate exists in paymentDates, it must match one of the unadjusted calculation period dates.

11 In %FpML_Offset: If the dayType element exists, the period must be 'D'.

%FpML_Offset defines paymentDaysOffset and rateCutOffDaysOffset.

12 In %FpML_Offset: If the dayType is 'Business', the periodMultiplier must be non zero.

%FpML_Offset defines paymentDaysOffset and rateCutOffDaysOffset.

13 In %FpML_RelativeDateOffset: If the dayType is 'Business', then the businessDayConvention should be 'NONE'.

%FpML_RelativeDateOffset defines fixingDateOffset and fixingDates.

14 In resetFrequency: weeklyRollConvention must exist if and only if the period is 'W'.

15 In ResetFrequency and calculationPeriodFrequency: calculationPeriodFrequency must be an integer multiple of the resetFrequency

16 In notionalStepSchedule, fixedRateSchedule, capRateSchedule, floorRateSchedule, and spreadSchedule: if step exists, stepDates in step must match one of the unadjusted calculation period dates.

17 In swapstream: calculationPeriodAmount/calculation/compoundingMethod must exist if and only if paymentDates/paymentFrequency and calculationPeriodDates/calculationPeriodFrequency are different.

18 In swapStream: if initialStub exists in stubCalculationPeriodAmount, at least one of either firstPeriodStartDate or firstRegularPeriodStartDate must exist in the calculationPeriodDates referenced by stubCalculationPeriodAmount.

19 In swapStream: if finalStub exists in stubCalculationPeriodAmount, lastRegularPeriodEndDate must exist in the calculationPeriodDates referenced by stubCalculationPeriodAmount.

20 In swapStream: payerPartyReference and receiverPartyReference must not be the same.

21 In %FpML_Fee: payerPartyReference and receiverPartyReference must not be the same. %FpML_Fee defines additionalPayment and otherPartyPayment .

22 In %FpML_Fee: At least one of paymentDate or adjustedPaymentDate must exist. %FpML_Fee defines additionalPayment and otherPartyPayment .

23 %FpML_Fee: paymentAmount/amount element have non zero value.

%FpML_Fee defines additionalPayment and otherPartyPayment.

24 In swapStream: if calculationPeriodAmount/calculation/compoundingMethod exists, resetDates must exist.

25 In calculationPeriodDates: effectiveDate must be before the terminationDate.

26 In %FpML_Schedule: If there are no step elements, initialValue must be non-zero.

%FpML_Schedule defines capRateSchedule, fixedRateSchedule, floorRateSchedule, and spreadSchedule.

27 In businessCentersReference there shall be a businessCenters element where the href attribute of the businessCentersReference element matches the attribute id of the businessCenters element.

28 In businessCenters: value of businessCenter elements must be unique.

6 Rule Implementation in xlinkit

In this section, we provide the full formalization of the above rules using the xlinkit rule language. In Section 5.1 we show the definition of operators that are used in the rules. The rules themselves are shown in Section 5.2 in XML in the same way as the rule-writer would edit them. As the rule language has a concrete XML syntax, we are able to render the rule using an XSLT stylesheet transformation into a more readable first order language, which we give in Section 5.3 in order to show how rules could be included in FpML standard documentation. In Section 5.4 we provide a reference to the FpML rule implementation that is available on the web for evaluation.

6.1 Operators

As described above xlinkit supports the definition of plugin operators. The rules shown in Section 5.1 use this concept to define operators that are not easily expressed in first order logic. In this section, we show an example of how these operators are defined in a way that xlinkit rules can invoke them and also how they can be implemented in JavaScript

6.1.1 Operator Interfaces

```
<?xml version="1.0" standalone="no"?>
<OperatorSet
  name="fpml"
  impl="operators/fpmlOperators.es"
  xmlns="http://www.xlinkit.com/OperatorSet/5.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xlinkit.com/OperatorSet/5.0 OperatorSet.xsd">

   <OperatorDefinition name="in_unadjusted_period_dates">
      <description xmlns:x="http://www.w3.org/1999/xhtml">
         Check if <x:b>checkdate</x:b> is on a periodic interval
         between <x:b>startdate</x:b> and <x:b>enddate</x:b>:
        <x:b>period</x:b> is added to the start date repeatedly, taking
        into account the unit (which must be D,W,M,Y). One of those dates
        must be the check date.
      </description>
      <param name="startdate" type="node"/>
      <param name="enddate" type="node"/>
      <param name="checkdate" type="node"/>
      <param name="period" type="node"/>
      <param name="periodunit" type="node"/>
    </OperatorDefinition>

    <OperatorDefinition name="unadjusted_period_dates_divides">
      <description xmlns:x="http://www.w3.org/1999/xhtml">
        Check if we can get from <x:b>startdate</x:b> to exactly the
        <x:b>enddate</x:b> by adding the <x:b>period</x:b> a number of
        times. If the enddate is exceeded by adding a period, the operator
        returns false. The unit must be D,W,M or Y.
      </description>
      <param name="startdate" type="node"/>
      <param name="enddate" type="node"/>
      <param name="period" type="node"/>
      <param name="periodunit" type="node"/>
    </OperatorDefinition>

    <OperatorDefinition name="greater_than">
      <description xmlns:x="http://www.w3.org/1999/xhtml">
        Check if <x:b>datea</x:b> is greater than <x:b>dateb</x:b>. The
        dates must be in the YYYY-MM-DD format.
      </description>
      <param name="datea" type="nodeList"/>
      <param name="dateb" type="nodeList"/>
    </OperatorDefinition>

   <OperatorDefinition name="is_period_multiple">
      <description xmlns:x="http://www.w3.org/1999/xhtml">
        Check if <x:b>periodA</x:b> is an integer multiple of
        <x:b>periodB</x:b>, taking into account the units, which must be
        Y,M,W,D. For example, 6 years is an integer multiple of 3 months.
        <x:b>Note:</x:b> This operator can only compare pairs of months and
        years, and weeks and days, respectively. Comparing months to weeks or
       days, etc. is illegal.
      </description>
      <param name="periodA" type="node"/>
      <param name="unitA" type="node"/>
      <param name="periodB" type="node"/>
      <param name="unitB" type="node"/>
    </OperatorDefinition>

</OperatorSet>
```

6.1.2 Operator Definition

```
function greater_than(date_a, date_b) {

  // if one does not exist return false
  if (date_a.getLength != 1 || date_b.getLength != 1) {
    return false ;
  }

  // Date format is YYYY-MM-DD
  s = new java.lang.String( date_a.item(0).getNodeValue() ) ;
  t = new java.lang.String( date_b.item(0).getNodeValue() ) ;

  return ( s.compareTo(t) > 0 ) ;
}

function createCalendar(date) {

    var digits = new Array(0,1,2,3,5,6,8,9);

    for (i=0;i<8;i++)
       if (!java.lang.Character.isDigit(date.charAt(digits[i])))
          return null;

    year=java.lang.Integer.parseInt(date.substring(0,4));
    month=java.lang.Integer.parseInt(date.substring(5,7))-1;
    day=java.lang.Integer.parseInt(date.substring(8,10));


    cal=java.util.Calendar.getInstance();
    cal.set(year,month,day,0,0,0);
    return cal;
}


function calendarEqual(calA,calB) {
    return calA.get(java.util.Calendar.YEAR)==calB.get(java.util.Calendar.YEAR) &&
calA.get(java.util.Calendar.MONTH)==calB.get(java.util.Calendar.MONTH) &&
calA.get(java.util.Calendar.DAY_OF_MONTH)==calB.get(java.util.Calendar.DAY_OF_MONTH);
}

function printcal(cal) {
    java.lang.System.out.print(cal.get(java.util.Calendar.YEAR)+"-");
    java.lang.System.out.print((cal.get(java.util.Calendar.MONTH)+1)+"-");
    java.lang.System.out.println(cal.get(java.util.Calendar.DAY_OF_MONTH));
}

function in_unadjusted_period_dates(startdate,enddate,checkdate,
        period,periodunit) {

    // Create java.util.Calendar objects for each date

    startcal=createCalendar(startdate.getNodeValue());
    endcal=createCalendar(enddate.getNodeValue());
    checkcal=createCalendar(checkdate.getNodeValue());

    // If dates cannot be created bail out

    if (startcal==null || endcal==null || checkcal==null)
       return false;

    period=java.lang.Integer.parseInt(period.getNodeValue());
    unitstring=new java.lang.String(periodunit.getNodeValue());

    unitfield=java.util.Calendar.DAY_OF_YEAR;

    if (unitstring.equals("W"))
       period=period*7;
    else
    if (unitstring.equals("M"))
       unitfield=java.util.Calendar.MONTH;
    else
    if (unitstring.equals("Y"))
       unitfield=java.util.Calendar.YEAR;
    else
    if (!unitstring.equals("D"))
       return false;

    current=startcal;
    while (current.before(endcal)) {
        //printcal(current);
        //printcal(checkcal);

        //java.lang.System.out.println("--");

       if (calendarEqual(current,checkcal))
          return true;

       current.add(unitfield,period);
    }

    if (checkcal.equals(endcal))
       return true;

    return false;
}
```

```
function unadjusted_period_dates_divides(startdate,enddate,period,periodunit) {

    // Create java.util.Calendar objects for each date

    startcal=createCalendar(startdate.getNodeValue());
    endcal=createCalendar(enddate.getNodeValue());

    // If dates cannot be created bail out

    if (startcal==null || endcal==null)
       return false;

    period=java.lang.Integer.parseInt(period.getNodeValue());
    unitstring=new java.lang.String(periodunit.getNodeValue());

    unitfield=java.util.Calendar.DAY_OF_YEAR;

    if (unitstring.equals("W"))
       period=period*7;
    else
    if (unitstring.equals("M"))
       unitfield=java.util.Calendar.MONTH;
    else
    if (unitstring.equals("Y"))
       unitfield=java.util.Calendar.YEAR;
    else
    if (!unitstring.equals("D"))
       return false;

    current=startcal;
    while (current.before(endcal)) {
        //printcal(current);
        //printcal(endcal);
        //java.lang.System.out.println("--");

        if (calendarEqual(current,endcal)) return true;

        current.add(unitfield,period);
    }

    if (calendarEqual(current,endcal))
       return true;
    else
       return false;
}


function is_period_multiple(periodA,unitA,periodB,unitB) {

    periodA=java.lang.Integer.parseInt(periodA.getNodeValue());
    unitA=new java.lang.String(unitA.getNodeValue());

    periodB=java.lang.Integer.parseInt(periodB.getNodeValue());
    unitB=new java.lang.String(unitB.getNodeValue());

    if (unitA.equals("Y") || unitA.equals("M")) {
       if (!(unitB.equals("M") || unitB.equals("Y")))
           return false;

       if (unitA.equals("Y"))
           periodA=periodA*12;
       if (unitB.equals("Y"))
           periodB=periodB*12;


       return (periodA % periodB)==0;
    }
    else
    if (unitA.equals("W") || unitA.equals("D")) {
       if (!(unitB.equals("D") || unitB.equals("W")))
           return false;

       if (unitA.equals("W"))
           periodA=periodA*7;
       if (unitB.equals("W"))
           periodB=periodB*7;

       return (periodA % periodB)==0;
    }

    return false;
}
```

6.2 Rules defined in XML

```
<?xml version="1.0" encoding="utf-8"?>
<consistencyruleset
xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
xmlns:macro="http://www.xlinkit.com/Macro/5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.xlinkit.com/ConsistencyRuleSet/5.0 consistencyrule.xsd">

  <consistencyrule id="r1">
    <header>
      <description>ResetDates must be present in a swapStream if
      and only if a floatingRateCalculation element is present in
      the calculation element.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x" in="//swapStream">
      <iff>
        <exists var="y" in="$x/resetDates" />
        <exists var="z" in="$x/calculationPeriodAmount/calculation/floatingRateCalculation" />
      </iff>
    </forall>
  </consistencyrule>

  <consistencyrule id="r2">
    <header>
      <description>In swapStream: element ResetDates must not exist
      if and only if element fixedRateSchedule exists in the
      calculation element.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x" in="//swapStream">
      <iff>
        <not>
          <exists var="y" in="$x/resetDates" />
        </not>
        <exists var="z" in="$x/calculationPeriodAmount/calculation/fixedRateSchedule" />
      </iff>
    </forall>
  </consistencyrule>

  <consistencyrule id="r3">
    <header>
      <description>In %FpML_BusinessDayAdjustments: neither
      businessCentersReference nor businessCenters must exist if
      and only if the value of businessDayConvention is 'NONE'. %
      FpML_BusinessDayAdjustments defines
      calculationPeriodDatesAdjustments, dateAdjustments,
      paymentDatesAdjustments, and
      resetDatesAdjustments.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x" in="//calculationPeriodDatesAdjustments|//dateAdjustments|
                        //paymentDatesAdjustments|//resetDatesAdjustments">
      <iff>
        <equal op1="$x/businessDayConvention/text()" op2="'NONE'" />
        <and>
          <not>
            <exists var="y" in="$x/businessCentersReference" />
          </not>
          <not>
            <exists var="y" in="$x/businessCenters" />
          </not>
        </and>
      </iff>
    </forall>
  </consistencyrule>

  <consistencyrule id="r4">
    <header>
      <description>In calculationPeriodDates: firstPeriodStartDate
      and effectiveDate must not be the same.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x" in="//calculationPeriodDates">
      <forall var="y" in="$x/effectiveDate/unadjustedDate">
        <not>
          <exists var="z"
          in="$x/firstPeriodStartDate/unadjustedDate">
            <equal op1="$y/text()" op2="$z/text()" />
          </exists>
        </not>
      </forall>
    </forall>
  </consistencyrule>


  <consistencyrule id="r5">
    <header>
      <description>In calculationPeriodDates: terminationDate and
      lastRegularPeriodEndDate must not be the same.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x" in="//calculationPeriodDates">
      <forall var="y" in="$x/terminationDate">
        <not>
          <exists var="z" in="$x/lastRegularPeriodEndDate">
            <equal op1="$y/unadjustedDate/text()"
            op2="$z/text()" />
          </exists>
        </not>
      </forall>
    </forall>
  </consistencyrule>



  <consistencyrule id="r6">
    <header>
      <author>Christian</author>

      <description>
```

```
      CalculationPeriodFrequency must divide the regular period precisely.
    </description>
  </header>

  <forall var="x" in="//calculationPeriodDates">
    <forall var="s"
    in="$x/effectiveDate[count(../firstRegularPeriodStartDate)=0 and
                         count(../firstPeriodStartDate)=0]|
        $x/firstPeriodStartDate[count(../firstRegularPeriodStartDate)=0] |
        $x/firstRegularPeriodStartDate">

      <forall var="e" in="$x/terminationDate[count(../lastRegularPeriodEndDate)=0] |
                          $x/lastRegularPeriodEndDate">

        <operator name="fpml:unadjusted_period_dates_divides">
          <param name="startdate"
          value="$s/unadjustedDate/text() | $s[count(unadjustedDate)=0]/text()" />

          <param name="enddate"
          value="$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text()" />

          <param name="period"
          value="$x/calculationPeriodFrequency/periodMultiplier/text()" />

          <param name="periodunit"
          value="$x/calculationPeriodFrequency/period/text()" />
        </operator>
      </forall>
    </forall>
  </forall>
</consistencyrule>



<consistencyrule id="r7">
  <header>
    <description>In calculationPeriodFrequency: if rollConvention
    is not either 'NONE' or 'SFE' then the period must be 'M' or
    'Y'.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//calculationPeriodFrequency">
    <implies>
      <or>
        <notequal op1="$x/rollConvention/text()" op2="'NONE'" />

        <notequal op1="$x/rollConvention/text()" op2="'SFE'" />
      </or>

      <or>
        <equal op1="$x/period/text()" op2="'M'" />

        <equal op1="$x/period/text()" op2="'Y'" />
      </or>
    </implies>
  </forall>
</consistencyrule>


<consistencyrule id="r8">
  <header>
    <author>Daniel</author>

    <description>In swapStream: paymentFrequency in paymentDates
    must be an integer multiple of calculationPeriodFrequency in
    calculationPeriodFrequency.</description>
  </header>

  <forall var="x" in="//swapStream">
    <forall var="y" in="$x/paymentDates/paymentFrequency">
      <forall var="z"
      in="$x/calculationPeriodDates/calculationPeriodFrequency">
```

```
          <operator name="fpml:is_period_multiple">
            <param name="periodA" value="$y/periodMultiplier/text()" />

            <param name="unitA" value="$y/period/text()" />

            <param name="periodB" value="$z/periodMultiplier/text()" />

            <param name="unitB" value="$z/period/text()" />
          </operator>
        </forall>
      </forall>
    </forall>
  </consistencyrule>



  <consistencyrule id="r9">
    <header>
      <author>Christian</author>

      <description>
      </description>
    </header>

    <forall var="x" in="//swapStream">
      <forall var="y" in="$x/paymentDates">
        <implies>
          <exists var="l" in="$y/firstPaymentDate" />

<!-- Either the date matches firstPeriodStartDate or it is
               one of the unadjusted period dates -->
          <or>
            <equal
            op1="$x/calculationPeriodDates/firstPeriodStartDate/unadjustedDate/text()"
             op2="$y/firstPaymentDate/text()" />

            <forall var="s" in="$x/calculationPeriodDates/effectiveDate
                                    [count(../firstRegularPeriodStartDate)=0 and
                                    count(../firstPeriodStartDate)=0] |
                              $x/calculationPeriodDates/firstPeriodStartDate
                                    [count(../firstRegularPeriodStartDate)=0] |
                              $x/calculationPeriodDates/firstRegularPeriodStartDate">

              <forall var="e" in="$x/calculationPeriodDates/terminationDate
                                    [count(../lastRegularPeriodEndDate)=0] |
                                $x/calculationPeriodDates/lastRegularPeriodEndDate">

                <operator name="fpml:in_unadjusted_period_dates">
                <param name="startdate"
                value="$s/unadjustedDate/text() | $s[count(unadjustedDate)=0]/text()" />

                <param name="enddate"
                value="$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text()" />

                <param name="checkdate"
                value="$y/firstPaymentDate/text()" />

                <param name="period"
                value="$x/calculationPeriodDates/calculationPeriodFrequency/periodMultiplier/text()" />

                <param name="periodunit"
                value="$x/calculationPeriodDates/calculationPeriodFrequency/period/text()" />
                </operator>
              </forall>
            </forall>
          </or>
        </implies>
      </forall>
    </forall>
  </consistencyrule>


  <consistencyrule id="r10">
    <header>
      <author>Christian</author>
      <description>
        In swapstream: if lastRegularPaymentDate exists in paymentDates, it must
        match one of the unadjusted calculation period dates.
      </description>
    </header>

    <forall var="x" in="//swapStream">
      <forall var="y" in="$x/paymentDates">
        <implies>
          <exists var="l" in="$y/lastRegularPaymentDate" />

<!-- Either the date matches firstPeriodStartDate or it is
               one of the unadjusted period dates -->
          <or>
            <equal
            op1="$x/calculationPeriodDates/firstPeriodStartDate/unadjustedDate/text()"
             op2="$y/lastRegularPaymentDate/text()" />

            <forall var="s" in="$x/calculationPeriodDates/effectiveDate[
                                   count(../firstRegularPeriodStartDate)=0 and
                                   count(../firstPeriodStartDate)=0] |
                                $x/calculationPeriodDates/firstPeriodStartDate[
                                   count(../firstRegularPeriodStartDate)=0] |
                                $x/calculationPeriodDates/firstRegularPeriodStartDate">

              <forall var="e" in="$x/calculationPeriodDates/terminationDate[
                                      count(../lastRegularPeriodEndDate)=0] |
                                  $x/calculationPeriodDates/lastRegularPeriodEndDate">

                <operator name="fpml:in_unadjusted_period_dates">
                <param name="startdate"
                value="$s/unadjustedDate/text() | $s[count(unadjustedDate)=0]/text()" />

                <param name="enddate"
                value="$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text()" />

                <param name="checkdate"
                value="$y/lastRegularPaymentDate/text()" />

                <param name="period"
                value="$x/calculationPeriodDates/calculationPeriodFrequency/periodMultiplier/text()" />

                <param name="periodunit"
                value="$x/calculationPeriodDates/calculationPeriodFrequency/period/text()" />
                </operator>
              </forall>
            </forall>
          </or>
        </implies>
      </forall>
    </forall>
  </consistencyrule>


  <consistencyrule id="r11">
    <header>
      <description>In %FpML_Offset: If the dayType element exists,
      the period must be 'D'. %FpML_Offset defines
      paymentDaysOffset and rateCutOffDaysOffset.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x"
    in="//paymentDaysOffset|//rateCutOffDaysOffset">
      <implies>
        <exists var="y" in="$x/dayType" />

        <equal op1="$x/period/text()" op2="'D'" />
      </implies>
    </forall>
  </consistencyrule>
```

```
<consistencyrule id="r12">
  <header>
    <description>In %FpML_Offset: If the dayType is 'Business',
    the periodMultiplier must be non zero</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x"
  in="//paymentDaysOffset|//rateCutOffDaysOffset">
    <implies>
      <equal op1="$x/dayType/text()" op2="'Business'" />

      <notequal op1="$x/periodMultiplier/text()" op2="'0'" />
    </implies>
  </forall>
</consistencyrule>



<consistencyrule id="r13">
  <header>
    <description>In %FpML_RelativeDateOffset: If the dayType is
    'Business', then the businessDayConvention should be 'NONE'.
    %FpML_RelativeDateOffset defines fixingDateOffset and
    fixingDates.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//fixingDateOffset|//fixingDates">
    <implies>
      <equal op1="$x/dayType/text()" op2="'Business'" />

      <equal op1="$x/businessDayConvention/text()"
      op2="'NONE'" />
    </implies>
  </forall>
</consistencyrule>



<consistencyrule id="r14">
  <header>
    <description>In resetFrequency: weeklyRollConvention must
    exist if and only if the period is 'W'.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//resetFrequency">
    <iff>
      <exists var="y" in="$x/weeklyRollConvention" />

      <equal op1="$x/period/text()" op2="'W'" />
    </iff>
  </forall>
</consistencyrule>




<consistencyrule id="r15">
  <header>
    <author>Daniel</author>

    <description>In swapStream: calculationPeriodFrequency in
    calculationPeriodDates is integer multiple of resetFrequency
    in resetDates.</description>
  </header>

  <forall var="x" in="//swapStream">
    <forall var="y" in="$x/resetDates/resetFrequency">
      <forall var="z"
      in="$x/calculationPeriodDates/calculationPeriodFrequency">
        <operator name="fpml:is_period_multiple">
          <param name="periodA"
          value="$z/periodMultiplier/text()" />

          <param name="unitA" value="$z/period/text()" />

          <param name="periodB"
          value="$y/periodMultiplier/text()" />

          <param name="unitB" value="$y/period/text()" />
        </operator>
      </forall>
    </forall>
  </forall>
</consistencyrule>



<consistencyrule id="r16">
  <header>
    <author>Christian</author>

    <description>
      In notionalStepSchedule, fixedRateSchedule, capRateSchedule, floorRateSchedule,
      and spreadSchedule: if step exists, stepDates in step must match one of the
      unadjusted calculation period dates.
    </description>
  </header>

  <forall var="x" in="//swapStream">
    <forall var="y"
    in="$x//notionalStepSchedule | $x//fixedRateSchedule |
        $x//capRateSchedule | $x//floorRateSchedule |
        $x//knownAmountSchedule | $x/spreadSchedule">

      <forall var="z" in="$y/step/stepDate">
          <!-- Either the step date matches firstPeriodStartDate or it is
             one of the unadjusted period dates -->
        <or>
          <equal
          op1="$x/calculationPeriodDates/firstPeriodStartDate/unadjustedDate/text()"
           op2="$z/text()" />

          <forall var="s" in="$x/calculationPeriodDates/effectiveDate
                                    [count(../firstRegularPeriodStartDate)=0 and
                                     count(../firstPeriodStartDate)=0] |
                              $x/calculationPeriodDates/firstPeriodStartDate
                                    [count(../firstRegularPeriodStartDate)=0] |
                              $x/calculationPeriodDates/firstRegularPeriodStartDate">

            <forall var="e" in="$x/calculationPeriodDates/terminationDate
                                     [count(../lastRegularPeriodEndDate)=0] |
                                $x/calculationPeriodDates/lastRegularPeriodEndDate">

              <operator name="fpml:in_unadjusted_period_dates">
              <param name="startdate"
               value="$s/unadjustedDate/text() | $s[count(unadjustedDate)=0]/text()" />
              <param name="enddate"
               value="$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text()" />

              <param name="checkdate" value="$z/text()" />

              <param name="period"
              value="$x/calculationPeriodDates/calculationPeriodFrequency/periodMultiplier/text()" />

              <param name="periodunit"
              value="$x/calculationPeriodDates/calculationPeriodFrequency/period/text()" />
              </operator>
            </forall>
          </forall>
        </or>
      </forall>
    </forall>
  </forall>
</consistencyrule>



<consistencyrule id="r17">
  <header>
    <author>Daniel</author>

    <description>In swapstream:
    calculationPeriodAmount/calculation/compoundingMethod must
    exist if and only if paymentDates/paymentFrequency and
    calculationPeriodDates/calculationPeriodFrequency are
    different.</description>
  </header>

  <forall var="x" in="//swapStream">
    <iff>
      <exists var="y"
      in="$x/calculationPeriodAmount/calculation/compoundingMethod" />

      <not>
        <and>
          <equal
          op1="$x/paymentDates/paymentFrequency/periodMultiplier/text()"
           op2="$x/calculationPeriodDates/calculationPeriodFrequency/periodMultiplier/text()" />

          <equal
          op1="$x/paymentDates/paymentFrequency/period/text()"
          op2="$x/calculationPeriodDates/calculationPeriodFrequency/period/text()" />
        </and>
      </not>
    </iff>
  </forall>
</consistencyrule>



<consistencyrule id="r18">
  <header>
    <description>In swapStream: if initialStub exists in
    stubCalculationPeriodAmount, at least one of either
    firstPeriodStartDate or firstRegularPeriodStartDate must
    exist in the calculationPeriodDates referenced by
    stubCalculationPeriodAmount.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//swapStream">
    <implies>
      <exists var="y"
      in="$x/stubCalculationPeriodAmount/initialStub" />

      <and>
        <or>
          <exists var="z"
          in="$x/calculationPeriodDates/firstPeriodStartDate" />

          <exists var="z"
          in="$x/calculationPeriodDates/firstRegularPeriodStartDate" />
        </or>

        <equal op1="$x/calculationPeriodDates/@id"
        op2="substring($x/stubCalculationPeriodAmount/calculationPeriodDatesReference/@href, 2)" />
      </and>
    </implies>
  </forall>
</consistencyrule>
```

```
<consistencyrule id="r19">
  <header>
    <description>In swapStream: if finalStub exists in
    stubCalculationPeriodAmount, lastRegularPeriodEndDate must
    exist in the calculationPeriodDates referenced by
    stubCalculationPeriodAmount.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//swapStream">
    <implies>
      <exists var="y"
      in="$x/stubCalculationPeriodAmount/finalStub" />

      <exists var="z"
      in="$x/calculationPeriodDates/lastRegularPeriodEndDate">
        <equal op1="$x/calculationPeriodDates/@id"
        op2="substring($x/stubCalculationPeriodAmount/calculationPeriodDatesReference/@href, 2)" />
      </exists>
    </implies>
  </forall>
</consistencyrule>



<consistencyrule id="r20">
  <header>
    <description>In swapStream: PayerPartyReference and
    receiverPartyReference must not be the same.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//swapStream">
    <forall var="y" in="$x/payerPartyReference">
      <not>
        <exists var="z" in="$x/receiverPartyReference">
          <equal op1="$y/@href" op2="$z/@href" />
        </exists>
      </not>
    </forall>
  </forall>
</consistencyrule>



<consistencyrule id="r21">
  <header>
    <description> In %FpML_Fee: payerPartyReference and
      receiverPartyReference must not be the same.
      %FpML_Fee defines additionalPayment and otherPartyPayment .
    </description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//additionalPayment|//otherPartyPayment">
    <forall var="y" in="$x/payerPartyReference">
      <not>
        <exists var="z" in="$x/receiverPartyReference">
          <equal op1="$y/@href" op2="$z/@href" />
        </exists>
      </not>
    </forall>
  </forall>
</consistencyrule>
```

```
<consistencyrule id="r22">
  <header>
    <description>In %FpML_Fee: At least one of paymentDate or
    adjustedPaymentDate must exist. %FpML_Fee defines
    additionalPayment and otherPartyPayment .</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//additionalPayment|//otherPartyPayment">
    <or>
      <exists var="y" in="$x/paymentDate" />

      <exists var="y" in="$x/adjustedPaymentDate" />
    </or>
  </forall>
</consistencyrule>



<consistencyrule id="r23">
  <header>
    <description>%FpML_Fee: paymentAmount/amount element have non
    zero value. %FpML_Fee defines additionalPayment and
    otherPartyPayment.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//additionalPayment|//otherPartyPayment">
    <forall var="y" in="$x/paymentAmount/amount">
      <notequal op1="$y/text()" op2="'0.00'" />
    </forall>
  </forall>
</consistencyrule>



<consistencyrule id="r24">
  <header>
    <description>In swapStream: if
    calculationPeriodAmount/calculation/compoundingMethod exists,
    resetDates must exist.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>

  <forall var="x" in="//swapStream">
    <implies>
      <exists var="y" in="$x/calculationPeriodAmount/calculation/compoundingMethod" />

      <exists var="y" in="$x/resetDates" />
    </implies>
  </forall>
</consistencyrule>



<consistencyrule id="r25">
  <header>
    <description>In calculationPeriodDates: effectiveDate must be
    before the terminationDate.</description>
  </header>

  <linkgeneration>
    <eliminatesymmetry status="on" />
  </linkgeneration>
```

```
    <forall var="x" in="//calculationPeriodDates">
      <forall var="y" in="$x/terminationDate">
        <forall var="z" in="$x/effectiveDate">
          <operator name="fpml:greater_than">
            <param name="datea"
            value="$x/terminationDate/unadjustedDate/text()" />

            <param name="dateb"
            value="$x/effectiveDate/unadjustedDate/text()" />
          </operator>
        </forall>
      </forall>
    </forall>
  </consistencyrule>



  <consistencyrule id="r26">
    <header>
      <description>In %FpML_Schedule: If there are no step
      elements, initialValue must be non-zero. %FpML_Schedule
      defines capRateSchedule, fixedRateSchedule,
      floorRateSchedule, and spreadSchedule.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x"
    in="//capRateSchedule|//fixedRateSchedule|//floorRateSchedule|//spreadSchedule">

      <implies>
        <not>
          <exists var="y" in="$x/step" />
        </not>

        <forall var="y" in="$x/initialValue">
          <notequal op1="$y/text()" op2="'0.00'" />
        </forall>
      </implies>
    </forall>
  </consistencyrule>


  <consistencyrule id="r27">
    <header>
      <description>In businessCentersReference: value of attribute
      href must be equal to the value attribute id of at least one
      businessCenters.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>

    <forall var="x" in="//businessCentersReference">
      <exists var="y" in="//businessCenters">
        <equal op1="substring($x/@href,2)" op2="$y/@id" />
      </exists>
    </forall>
  </consistencyrule>




  <consistencyrule id="r28">
    <header>
      <description>In businessCenters: value of businessCenter
      elements must be unique.</description>
    </header>

    <linkgeneration>
      <eliminatesymmetry status="on" />
    </linkgeneration>
    <forall var="a" in="//businessCenters">
      <forall var="x" in="$a/businessCenter">
        <forall var="y" in="$a/businessCenter">
          <implies>
            <not>
              <same op1="$x" op2="$y" />
            </not>

            <notequal op1="$x/text()" op2="$y/text()" />
          </implies>
        </forall>
      </forall>
    </forall>
  </consistencyrule>
</consistencyruleset>
```

6.3 Rules rendered in 1st order logic

Consistency Rule r1	
Description	*ResetDates must be present in a swapStream if and only if a floatingRateCalculation element is present in the calculation element.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //swapStream** (**exists** y **in $x/resetDates** () <-> **exists** z **in $x/calculationPeriodAmount/calculation/floatingRateCalculation** ())

Consistency Rule r2	
Description	*In swapStream: element ResetDates must not exist if and only if element fixedRateSchedule exists in the calculation element.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //swapStream** (**not exists** y **in $x/resetDates** () <-> **exists** z **in $x/calculationPeriodAmount/calculation/fixedRateSchedule** ())

Consistency Rule r3	
Description	*In %FpML_BusinessDayAdjustments: neither businessCentersReference nor businessCenters must exist if and only if the value of businessDayConvention is 'NONE'. % FpML_BusinessDayAdjustments defines calculationPeriodDatesAdjustments, dateAdjustments, paymentDatesAdjustments, and resetDatesAdjustments.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //calculationPeriodDatesAdjustments|//dateAdjustments|//paymentDatesAdjustments|//resetDatesAdjustments** ($x/businessDayConvention/text()='NONE' <-> **not exists** y **in $x/businessCentersReference** () **and not exists** y **in $x/businessCenters** ())

Consistency Rule r4	
Description	*In calculationPeriodDates: firstPeriodStartDate and effectiveDate must not be the same.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //calculationPeriodDates** (
	    forall y **in $x/effectiveDate/unadjustedDate** (
	        not exists z **in $x/firstPeriodStartDate/unadjustedDate** (
	            $y/text()=$z/text()
	        )
	    )
	)

Consistency Rule r5	
Description	*In calculationPeriodDates: terminationDate and lastRegularPeriodEndDate must not be the same.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //calculationPeriodDates** (
	    forall y **in $x/terminationDate** (
	        not exists z **in $x/lastRegularPeriodEndDate** (
	            $y/unadjustedDate/text()=$z/text()
	        )
	    )
	)

Consistency Rule r6	
Description	*CalculationPeriodFrequency must divide the regular period precisely*
Link Generation	Consistent **on**

	Inconsistent **on**
Rule	**forall** x **in //calculationPeriodDates** (**forall** s **in $x/effectiveDate[count(../firstRegularPeriodStartDate)=0 and count(../firstPeriodStartDate)=0] | $x/firstPeriodStartDate[count(../firstRegularPeriodStartDate)=0] | $x/firstRegularPeriodStartDate** (**forall** e **in $x/terminationDate[count(../lastRegularPeriodEndDate)=0] | $x/lastRegularPeriodEndDate** (**fpml:unadjusted_period_dates_divides**($s/unadjustedDate/text() | $s[count(unadjustedDate)=0]/text(),$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text(),$x/calculationPeriodFrequency/periodMultiplier/text(),$x/calculationPeriodFrequency/period/text()))))

Consistency Rule r7	
Description	*In calculationPeriodFrequency: if rollConvention is not either 'NONE' or 'SFE' then the period must be 'M' or 'Y'.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //calculationPeriodFrequency** ($x/rollConvention/text()**!=**'NONE' **or** $x/rollConvention/text()**!=**'SFE' **implies** $x/period/text()='M' **or** $x/period/text()='Y')

Consistency Rule r8	
Description	*In swapStream: paymentFrequency in paymentDates must be an integer multiple of calculationPeriodFrequency in calculationPeriodFrequency.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //swapStream** (**forall** y **in $x/paymentDates/paymentFrequency** (**forall** z **in $x/calculationPeriodDates/calculationPeriodFrequency** (**fpml:is_period_multiple**($y/periodMultiplier/text(),$y/period/text(),$z/periodMultiplier/text(),$z/period/text()))))

Consistency Rule r9	
Description	
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //swapStream** (**forall** y **in $x/paymentDates** (**exists** l **in $y/firstPaymentDate** () **implies** $x/calculationPeriodDates/firstPeriodStartDate/unadjustedDate/text()=$y/firstPaymentDate/text() **or forall** s **in $x/calculationPeriodDates/effectiveDate[count(../firstRegularPeriodStartDate)=0 and count(../firstPeriodStartDate)=0] | $x/calculationPeriodDates/firstPeriodStartDate[count(../firstRegularPeriodStartDate)=0] | $x/calculationPeriodDates/firstRegularPeriodStartDate** (**forall** e **in $x/calculationPeriodDates/terminationDate[count(../lastRegularPeriodEndDate)=0] | $x/calculationPeriodDates/lastRegularPeriodEndDate** (**fpml:in_unadjusted_period_dates**($s/unadjustedDate/text() | $s[count(unadjustedDate)=0]/text(),$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text() ,$y/firstPaymentDate/text(),$x/calculationPeriodDates/calculationPeriodFrequency/periodMultiplier/text() ,$x/calculationPeriodDates/calculationPeriodFrequency/period/text())))))

Consistency Rule r10	
Description	*In swapstream: if lastRegularPaymentDate exists in paymentDates, it must match one of the unadjusted calculation period dates.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //swapStream** (
	    forall y **in $x/paymentDates** (
	        exists l **in $y/lastRegularPaymentDate** () **implies**
	        $x/calculationPeriodDates/firstPeriodStartDate/unadjustedDate/text()=$y/lastRegularPaymentDate/text() **or forall** s **in**
	        $x/calculationPeriodDates/effectiveDate[count(../firstRegularPeriodStartDate)=0 and
	        count(../firstPeriodStartDate)=0] |
	        $x/calculationPeriodDates/firstPeriodStartDate[count(../firstRegularPeriodStartDate)=0] |
	        $x/calculationPeriodDates/firstRegularPeriodStartDate (
	            forall e **in $x/calculationPeriodDates/terminationDate[count(../lastRegularPeriodEndDate)=0] |**
	            $x/calculationPeriodDates/lastRegularPeriodEndDate (
	                fpml:in_unadjusted_period_dates($s/unadjustedDate/text() |
	                $s[count(unadjustedDate)=0]/text(),$e/unadjustedDate/text() |
	                $e[count(unadjustedDate)=0]/text(),$y/lastRegularPaymentDate/text(),$x/calculationPeriodDates/calculationPeriodF
	                requency/periodMultiplier/text(),$x/calculationPeriodDates/calculationPeriodFrequency/period/text())
	            )
	        )
	    )
	)

Consistency Rule r11	
Description	*In %FpML_Offset: If the dayType element exists, the period must be 'D'. %FpML_Offset defines paymentDaysOffset and rateCutOffDaysOffset.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //paymentDaysOffset|//rateCutOffDaysOffset** (**exists** y **in $x/dayType** () **implies** $x/period/text()='D')

Consistency Rule r12	
Description	*In %FpML_Offset: If the dayType is 'Business', the periodMultiplier must be non zero*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //paymentDaysOffset|//rateCutOffDaysOffset** ($x/dayType/text()='Business' **implies** $x/periodMultiplier/text()**!**='0')

Consistency Rule r13	
Description	*In %FpML_RelativeDateOffset: If the dayType is 'Business', then the businessDayConvention should be 'NONE'. %FpML_RelativeDateOffset defines fixingDateOffset and fixingDates.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //fixingDateOffset|//fixingDates** ($x/dayType/text()='Business' **implies** $x/businessDayConvention/text()='NONE')

Consistency Rule r14	
Description	*In resetFrequency: weeklyRollConvention must exist if and only if the period is 'W'.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //resetFrequency** (**exists** y **in $x/weeklyRollConvention** () <-> $x/period/text()='W')

Consistency Rule r15	
Description	*In swapStream: calculationPeriodFrequency in calculationPeriodDates is integer multiple of resetFrequency in resetDates.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //swapStream** (
	    forall y **in $x/resetDates/resetFrequency** (
	        forall z **in $x/calculationPeriodDates/calculationPeriodFrequency** (
	            fpml:is_period_multiple($z/periodMultiplier/text(),$z/period/text(),$y/periodMultiplier/text(),$y/period/text())
	        )
	    )
	)

Consistency Rule r16	
Description	*In notionalStepSchedule, fixedRateSchedule, capRateSchedule, floorRateSchedule, and spreadSchedule: if step exists, stepDates in step must match one of the unadjusted calculation period dates.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //swapStream** (
	    forall y **in $x//notionalStepSchedule | $x//fixedRateSchedule | $x//capRateSchedule | $x//floorRateSchedule |**
	    $x//knownAmountSchedule | $x/spreadSchedule (
	        forall z **in $y/step/stepDate** (
	            $x/calculationPeriodDates/firstPeriodStartDate/unadjustedDate/text()=$z/text()
	            or forall s **in $x/calculationPeriodDates/effectiveDate[count(../firstRegularPeriodStartDate)=0**
	            and count(../firstPeriodStartDate)=0] |
	            $x/calculationPeriodDates/firstPeriodStartDate[count(../firstRegularPeriodStartDate)=0] |
	            $x/calculationPeriodDates/firstRegularPeriodStartDate (
	                forall e **in $x/calculationPeriodDates/terminationDate[count(../lastRegularPeriodEndDate)=0] |**
	                $x/calculationPeriodDates/lastRegularPeriodEndDate (
	                    fpml:in_unadjusted_period_dates($s/unadjustedDate/text() |
	                    $s[count(unadjustedDate)=0]/text(),$e/unadjustedDate/text() | $e[count(unadjustedDate)=0]/text(),$z/text(),
	                    $x/calculationPeriodDates/calculationPeriodFrequency/periodMultiplier/text(),
	                    $x/calculationPeriodDates/calculationPeriodFrequency/period/text())
	                )
	            )
	        )
	    )
	)

Consistency Rule r17	
Description	*In swapstream: calculationPeriodAmount/calculation/compoundingMethod must exist if and only if paymentDates/paymentFrequency and calculationPeriodDates/calculationPeriodFrequency are different.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //swapStream** (**exists** y **in $x/calculationPeriodAmount/calculation/compoundingMethod** () **<-> not** $x/paymentDates/paymentFrequency/periodMultiplier/text()=$x/calculationPeriodDates/calculationPeriodFrequency/periodMult iplier/text() **and** $x/paymentDates/paymentFrequency/period/text()=$x/calculationPeriodDates/calculationPeriodFrequency/period/text())

Consistency Rule r18	
Description	*In swapStream: if initialStub exists in stubCalculationPeriodAmount, at least one of either firstPeriodStartDate or firstRegularPeriodStartDate must exist in the calculationPeriodDates referenced by stubCalculationPeriodAmount.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //swapStream** (
	    exists y **in $x/stubCalculationPeriodAmount/initialStub** () **implies exists** z **in**
	    $x/calculationPeriodDates/firstPeriodStartDate () **or exists** z **in $x/calculationPeriodDates/firstRegularPeriodStartDate** ()
	    and $x/calculationPeriodDates/@id=substring($x/stubCalculationPeriodAmount/calculationPeriodDatesReference/@href, 2)
	)

Consistency Rule r19	
Description	*In swapStream: if finalStub exists in stubCalculationPeriodAmount, lastRegularPeriodEndDate must exist in the calculationPeriodDates referenced by stubCalculationPeriodAmount.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //swapStream** (
	    exists y **in $x/stubCalculationPeriodAmount/finalStub** () **implies exists** z **in**
	    $x/calculationPeriodDates/lastRegularPeriodEndDate (
	        $x/calculationPeriodDates/@id=substring($x/stubCalculationPeriodAmount/calculationPeriodDatesReference/@href, 2)
	    )
	)

Consistency Rule r20	
Description	*In swapStream: PayerPartyReference and receiverPartyReference must not be the same.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //swapStream** (
	    forall y **in $x/payerPartyReference** (
	        not exists z **in $x/receiverPartyReference** (
	            $y/@href=$z/@href
	        )
	    )
	)

Consistency Rule r21	
Description	*In %FpML_Fee: payerPartyReference and receiverPartyReference must not be the same. %FpML_Fee defines additionalPayment and otherPartyPayment .*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //additionalPayment|//otherPartyPayment** (
	    forall y **in $x/payerPartyReference** (
	        not exists z **in $x/receiverPartyReference** (
	            $y/@href=$z/@href
	        )
	    )
	)

Consistency Rule r22	
Description	*In %FpML_Fee: At least one of paymentDate or adjustedPaymentDate must exist. %FpML_Fee defines additionalPayment and otherPartyPayment .*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //additionalPayment|//otherPartyPayment** (**exists** y **in $x/paymentDate** () **or exists** y **in $x/adjustedPaymentDate** ())

Consistency Rule r23	
Description	*%FpML_Fee: paymentAmount/amount element have non zero value. %FpML_Fee defines additionalPayment and otherPartyPayment.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //additionalPayment|//otherPartyPayment** (**forall** y **in $x/paymentAmount/amount** ($y/text()**!=**'0.00'))

Consistency Rule r24	
Description	*In swapStream: if calculationPeriodAmount/calculation/compoundingMethod exists, resetDates must exist.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //swapStream** (**exists** y **in $x/calculationPeriodAmount/calculation/compoundingMethod** () **implies exists** y **in $x/resetDates** ())

Consistency Rule r25	
Description	*In calculationPeriodDates: effectiveDate must be before the terminationDate.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //calculationPeriodDates** (
	    forall y **in $x/terminationDate** (
	        forall z **in $x/effectiveDate** (
	            fpml:greater_than($x/terminationDate/unadjustedDate/text(),$x/effectiveDate/unadjustedDate/text())
	        )
	    )
	)

Consistency Rule r26	
Description	*In %FpML_Schedule: If there are no step elements, initialValue must be non-zero. %FpML_Schedule defines capRateSchedule, fixedRateSchedule, floorRateSchedule, and spreadSchedule.*
Link Generation	Consistent **on** Inconsistent **on**
Rule	**forall** x **in //capRateSchedule|//fixedRateSchedule|//floorRateSchedule|//spreadSchedule** (**not exists** y **in $x/step** () **implies forall** y **in $x/initialValue** ($y/text()**!=**'0.00'))

Consistency Rule r27	
Description	*In businessCentersReference there shall be a businessCenters element where the href attribute of the businessCentersReference element matches the attribute id of the businessCenters element.*
Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** x **in //businessCentersReference** (
	    exists y **in //businessCenters** (
	        substring($x/@href,2)=$y/@id
	    )
	)

Consistency Rule r28	
Description	*In businessCenters: value of businessCenter elements must be unique.*

Link Generation	Consistent **on** Inconsistent **on**

Rule	**forall** a **in //businessCenters** (
	    forall x **in $a/businessCenter** (
	        forall y **in $a/businessCenter** (
	            not $x===$y **implies** $x/text()!=$y/text()
	        )
	    )
	)

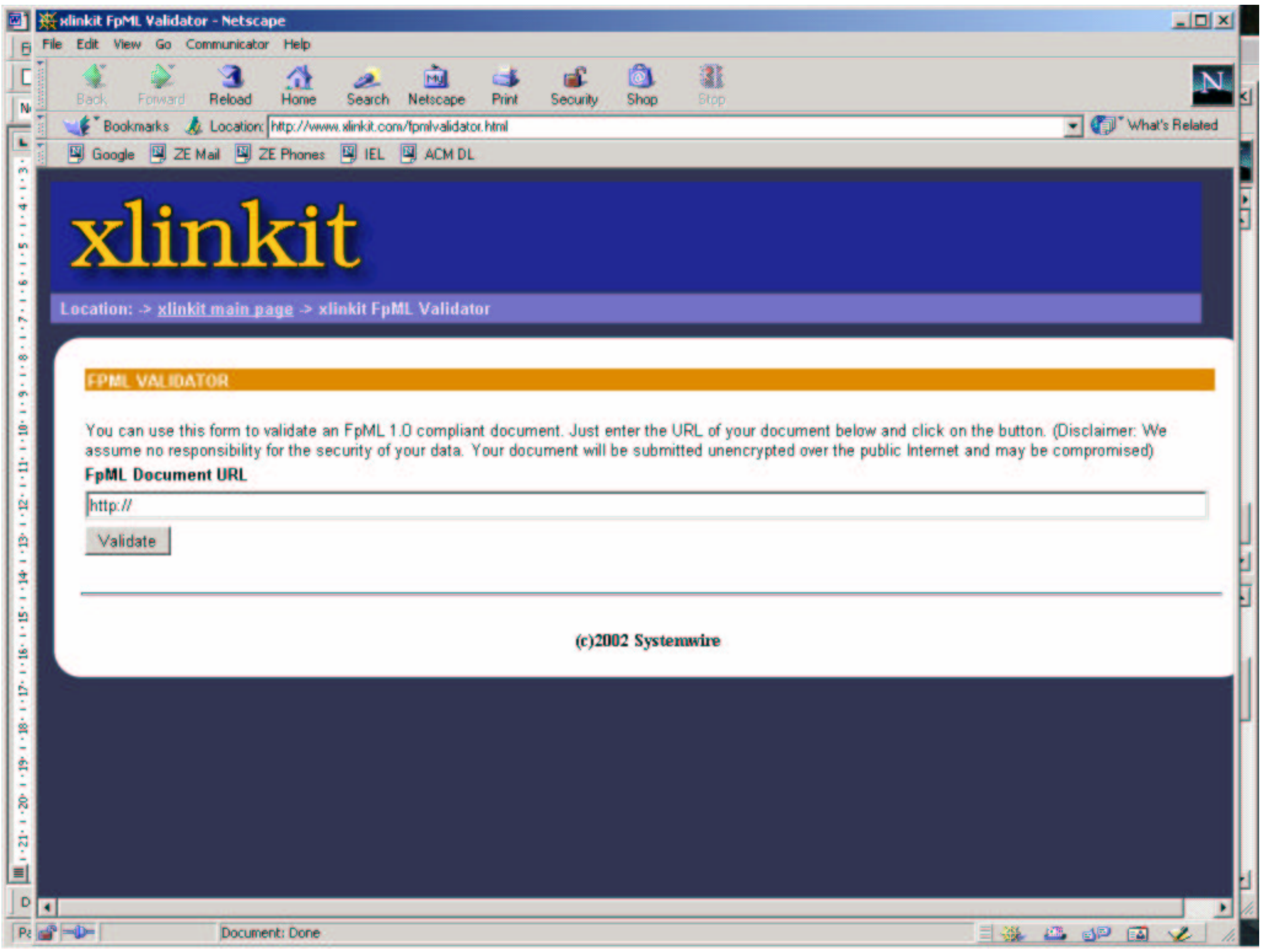
6.4 Reference Implementation

A reference implementation of the above rules is available at http://www.xlinkit.com/fpmlvalidator.html.

The xlinkit FpML Validator validates FpML 1.0 compliant documents against additional integrity constraints. Any FpML document can be submitted to the Validator by entering the document's URL into the form.

The validator will then execute the 28 rules identified in the previous section that check the validity of dates, proper referencing between business centers and more. These hyperlinks will connect elements in the FpML document that are in violation of a constraint, for example two business center elements. These hyperlinks are not intended for human consumption but as an intermediate representation based on which higher-level diagnostic tools can be built very easily.

Because it is based on XLink, xlinkit's hyperlinks can have more than two endpoints. Since there is no browser that can currently display such links, we render the links into HTML and show where each endpoint (or locator) is pointing. Each locator will point into the file and use XPath to highlight which element it is pointing to.


xlinkit FpML Validator - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.xlinkit.com/fpmlvalidator.html
What's Related
Google ZE Mail ZE Phones IEL ACM DL
xlinkit
Location: -> xlinkit main page -> xlinkit FpML Validator
FPML VALIDATOR
Validate
Document: Done

You can use this form to validate an FpML 1.0 compliant document. Just enter the URL of your document below and click on the button. (Disclaimer: We assume no responsibility for the security of your data. Your document will be submitted unencrypted over the public Internet and may be compromised)

FpML Document URL

http://

(c)2002 Systemwire

7 Evaluation

In this chapter we will reflect on the experience that we made when using xlinkit to express the validation rules for FpML 1.0. We will first review the expressiveness of the xlinkit rules and then report the performance measurements.

7.1 Coverage of Requirements

The table below shows the extent to which we have demonstrated in this proposal how well xlinkit addresses the requirements defined in Section 2.

R2.1.1 Semantic Validation	Xlinkit expresses static semantic validation rules that are beyond the syntactic rules that can be expressed in DTD or XML Schema.
R2.1.2 XML-based Definition	The various xlinkit languages have a concrete syntax defined in XML Schea and shown in Appendix 9.
R2.1.3 GUI tools to formulate and update rules	Systemwire has alpha release GUI tools for editing xlinkit rules, but these have not been discussed in this report.
R2.2 Multiple distributed rule sets	Rules are grouped into rule files and rule files are aggregated in rule sets. URLs are used to reference rules on remote web servers.
R2.3.1 Comparison to external data sources	When expressing rules, xlinkit does not assume any location information. Multiple documents can be checked by including them into a document set. This feature is not yet used in the FpML 1.0 validation rules.
R2.3.2 Check against non-FpML languages	xlinkit assumes that each document references its schema definition. There is no restriction on the number of different schema used.
R2.3.3 Distributed data sources	Document sets can reference distributed data using URLs. Fetchers can be used in order to load documents from sources other than Web servers (e.g. a JDBC Fetcher to load it from a database)
R2.4.1 Declarative rule language	xlinkit uses first order logic rules, a declarative rule language. As a result the rules are very concise.
R2.4.2 Domain-specific operators	Operators can be defined in ECMAscript, a standardized version of Javascript or in Java classes. These operators can then be invoked in rule files.
R2.4.3 Ease of comprehension	The rules are reasonably simple to understand for anyone who understands first-order predicate logic
R2.4.4 Rule structuring mechanisms	There are no restrictions on the number of rule files and rule sets.
R2.4.5 human + machine readable representation	As rules are defined in XML, they can be translated using an XSLT stylesheet into a more readable form.
R2.5.1 Classification of FpML product types	To date, we have not yet demonstrated how to use xlinkit to achieve classification of FpML product types.
R2.6.1 W3C compliance	xlinkit uses only Schemas, XPath and XLink.
R2.7.1 Efficient Execution	28 Validation rules were executed on a relatively small PC within less than a second

7.2 On the Benefits of Formalization

The rule descriptions that we obtained from people in Warburg were given in English, attempting to be as precise as possible. We have then formalized these constraints using the xlinkit rule language. During this process, we have identified many ambiguities and we had to discuss the meaning of some rules with our business contact. Once formalized, we were able to reformulate the original constraint in English, albeit in a more precise way. We give an example now. We were first given the following description of a constraint.

“BusinessCentersReference must reference a businessCenter element within the document.”

We translated that into the following xlinkit rule:

```
<forall var="x" in="//businessCentersReference">
  <exists var="y" in="//businessCenters">
    <equal op1="substring($x/@href,2)" op2="$y/@id" />
  </exists>
</forall>
```

Once we had gone through the formalization, we were able to capture the meaning of the constraint more precisely as:

In `businessCentersReference` there shall be a `businessCenters` element where the `href` attribute of the `businessCentersReference` element *matches* the attribute `id` of the `businessCenters` element.

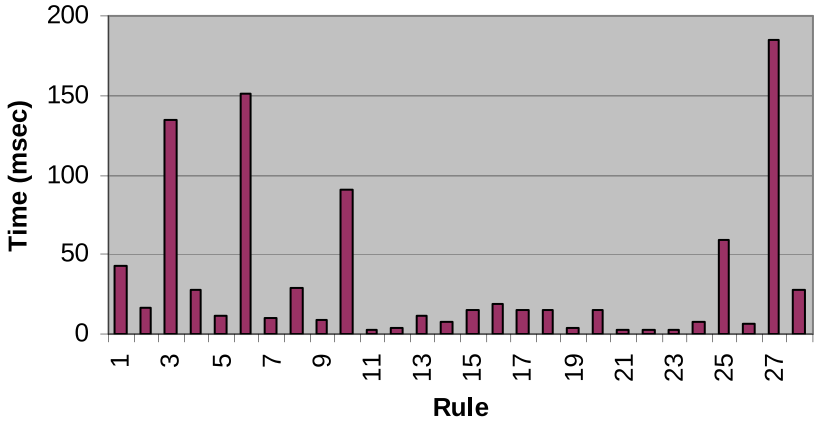
This new description formulation identifies explicitly the attributes to compare, which was unclear in the original formulation. The xlinkit rule also shows exactly what it is meant with ``matches". If the `id` value is `“primaryBusinessCenter”`, then the `href` values referencing it must have value ` ``\#primaryBusinessCenter'' ` (with a leading hash symbol). Hence, using an XPath expression, we impose that the substring starting on the second character of the `href` string must be equal to the entire `id` string.

Another by-product of the formalization process was that the detailed analysis of all the constraints has led us to identify gaps that demanded new constraints that were not evident from the beginning, or to condense several constraints into one. Therefore the whole exercise has led to a more complete and precise formulation of the validation constraints.

7.3 Performance

After having formalized the initial set of 35 constraints a total of 28 constraints remained (because some were subsumed) and others were proven to be obsolete. We then checked a 17KByte FpML 1.0 trade document that did not have any constraint violations using the xlinkit rule engine. The rule engine executed on a dual-processor Pentium III with a clock speed of 1.7 GHz and 1GByte of RAM. The rule angine runs as a single task so it does not use the second processor.

The figure below shows the performance measurements that we. The total for checking all 28 rules is just under 990msec. We also note that at some rules require a significantly longer time than others to evaluate. This happens when the evaluation of an XPath expression requires the traversal of the entire DOM tree. For example expression ``//businessCentersReference" appears in rule 27.



Expressions of this kind are necessary when the element we are trying to find can appear anywhere in the document, unless we can explicitly identify the position of the element in the document tree. In rule 1 we changed the expression from

``//swapStream" to ``/FpML/trade/product/swap/swapStream" and the execution time reduced by five folds (the figure in the graph). The only drawback is that long XPath expressions make rules less readable.

Caching is also important. If a rule uses an XPath expression that was evaluated already for another rule, it will execute much faster. This happens, for example, in rule 2, which uses an expression that the XPath processor had previously evaluated for rule 1.

In general XPath evaluation is the dominant performance factor. We expected the rules that use plug-in operators (for example 8, 9, 10, and 15) to be slower, but their execution time is in line with the other rules.

We are using Apache's Xalan as an XPath processor. Preliminary tests with a beta version of Jaxen indicate that a faster XPath processor can give significant performance improvements. Rule optimisation is also something on which we will be focussing.

8 Summary and Recommendation

In this proposal, we have demonstrated how semantic validations can be achieved using xlinkit. We have argued why xlinkit is superior to other approaches, such as attribute grammars, XSLT or OCL. We have provided evidence that xlinkit is expressive enough to concisely formalize all constraints for FpML 1.0 that we defined. We have customized the xlinkit rule engine to directly check one FpML trade document and have made that FpML validator available as a reference implementation on the systemwire web site. We have done some extensive performance analysis and shown that the 28 FpML 1.0 validation rules can be executed in less than a second. We have also shown that there is considerable benefit in formalizing the static semantics of FpML trades.

These observations lead us to conclude that xlinkit should be adopted as the standard FpML validation language and for FpML Product Working groups to use xlinkit to precisely define the constraints they wish to impose on rules. We would expect that such adoption of xlinkit would have a very positive effect on the efficiency with which financial trade ocuments can be handled both across FpML organizations, but also within different departments of the same organization.

9 APPENDIX

9.1 Rule language syntax (XML Schema)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
           xmlns="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="consistencyruleset">
  <xs:complexType>
   <xs:sequence>
    <xs:any namespace="http://www.xlinkit.com/Macro/5.0"
            processContents="skip" minOccurs="0"/>
    <xs:element name="globalset" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="xpath" type="xs:string" use="required"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="consistencyrule" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="header" minOccurs="0"/>
       <xs:element name="linkgeneration" minOccurs="0">
        <xs:complexType>
         <xs:choice maxOccurs="3">
          <xs:element name="consistent" minOccurs="0">
           <xs:complexType>
            <xs:attribute name="status" use="optional" default="on">
             <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="on"/>
               <xs:enumeration value="off"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
           </xs:complexType>
          </xs:element>
          <xs:element name="inconsistent" minOccurs="0">
           <xs:complexType>
            <xs:attribute name="status" use="optional" default="on">
             <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="on"/>
               <xs:enumeration value="off"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
           </xs:complexType>
          </xs:element>
          <xs:element name="eliminatesymmetry" minOccurs="0">
           <xs:complexType>
            <xs:attribute name="status" use="optional" default="off">
             <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="on"/>
               <xs:enumeration value="off"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
           </xs:complexType>
          </xs:element>
         </xs:choice>
        </xs:complexType>
       </xs:element>
       <xs:element ref="forall"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="header">
  <xs:complexType>
   <xs:sequence>
    <xs:choice maxOccurs="unbounded">
     <xs:element name="author" type="xs:string"/>
     <xs:element name="description">
      <xs:complexType mixed="true">
       <xs:sequence>
        <xs:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="project" type="xs:string"/>
     <xs:element name="comment" type="xs:string"/>
     <xs:any namespace="http://www.xlinkit.com/Metadata/5.0"
             processContents="skip"/>
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="quantifierType">
  <xs:group ref="formulaGroup" minOccurs="0"/>
  <xs:attribute name="var" type="xs:string" use="required"/>
  <xs:attribute name="in" type="xs:string" use="required"/>
  <xs:attribute name="mode" use="optional" default="exhaustive">
   <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="exhaustive"/>
     <xs:enumeration value="instance"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
 <xs:complexType name="binOperatorType">
  <xs:group ref="formulaGroup" minOccurs="2" maxOccurs="2"/>
 </xs:complexType>
 <xs:complexType name="binPredicateType">
```

```
  <xs:attribute name="op1" type="xs:string" use="required"/>
  <xs:attribute name="op2" type="xs:string" use="required"/>
 </xs:complexType>
 <xs:element name="forall" type="quantifierType"/>
 <xs:element name="and" type="binOperatorType"/>
 <xs:element name="or" type="binOperatorType"/>
 <xs:element name="implies" type="binOperatorType"/>
 <xs:element name="iff" type="binOperatorType"/>
 <xs:element name="not">
  <xs:complexType>
   <xs:group ref="formulaGroup"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="exists" type="quantifierType"/>
 <xs:element name="equal" type="binPredicateType"/>
 <xs:element name="notequal" type="binPredicateType"/>
 <xs:element name="same" type="binPredicateType"/>
 <xs:element name="subset">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="binPredicateType">
     <xs:attribute name="size" type="xs:int" use="optional" default="0"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="intersect">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="binPredicateType">
     <xs:attribute name="size" type="xs:int" use="optional" default="0"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="operator">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
       <xs:attribute name="name" type="xs:string" use="required"/>
       <xs:attribute name="value" type="xs:string" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:group name="formulaGroup">
  <xs:choice>
   <xs:element ref="forall"/>
   <xs:element ref="exists"/>
   <xs:element ref="equal"/>
   <xs:element ref="notequal"/>
   <xs:element ref="same"/>
   <xs:element ref="intersect"/>
   <xs:element ref="subset"/>
   <xs:element ref="and"/>
   <xs:element ref="or"/>
   <xs:element ref="implies"/>
   <xs:element ref="iff"/>
   <xs:element ref="not"/>
   <xs:element ref="operator"/>
   <xs:any namespace="http://www.xlinkit.com/Macro/5.0"
           processContents="skip"/>
  </xs:choice>
 </xs:group>
</xs:schema>
```

9.2 Document Set Definition syntax (XML Schema)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.xlinkit.com/DocumentSet/5.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.xlinkit.com/DocumentSet/5.0"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="DocumentSet">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="header" minOccurs="0"/>
    <xs:sequence>
     <xs:choice maxOccurs="unbounded">
      <xs:element name="Document">
       <xs:complexType>
        <xs:attribute name="href" type="xs:string" use="required"/>
        <xs:attribute name="fetcher" type="xs:string" use="optional"
                      default="FileFetcher"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="Set">
       <xs:complexType>
        <xs:attribute name="href" type="xs:string" use="required"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:sequence>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="optional"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="header">
  <xs:complexType>
   <xs:sequence>
    <xs:choice maxOccurs="unbounded">
     <xs:element name="author" type="xs:string"/>
     <xs:element name="description">
      <xs:complexType mixed="true">
       <xs:sequence>
        <xs:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="project" type="xs:string"/>
     <xs:element name="comment" type="xs:string"/>
     <xs:any namespace="http://www.xlinkit.com/Metadata/5.0"
             processContents="skip"/>
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

9.3 Rule Set Definition syntax (XML Schema)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.xlinkit.com/RuleSet/5.0"
           xmlns="http://www.xlinkit.com/RuleSet/5.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="RuleSet">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="header" minOccurs="0"/>
    <xs:sequence>
     <xs:choice maxOccurs="unbounded">
      <xs:element name="RuleFile">
       <xs:complexType>
        <xs:attribute name="href" type="xs:string" use="required"/>
        <xs:attribute name="xpath" type="xs:string" use="optional"
        default="/*[local-name()='consistencyruleset']/*[local-name()
                ='consistencyrule']"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="Set">
       <xs:complexType>
        <xs:attribute name="href" type="xs:string" use="required"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="Operators">
```

```
       <xs:complexType>
        <xs:attribute name="href" type="xs:string" use="required"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:sequence>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="optional"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="header">
  <xs:complexType>
   <xs:sequence>
    <xs:choice maxOccurs="unbounded">
     <xs:element name="author" type="xs:string"/>
     <xs:element name="description">
      <xs:complexType mixed="true">
       <xs:sequence>
        <xs:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="project" type="xs:string"/>
     <xs:element name="comment" type="xs:string"/>
     <xs:any namespace="http://www.xlinkit.com/Metadata/5.0"
             processContents="skip"/>
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

9.4 Operator Definition Syntax (XML Schema)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.xlinkit.com/OperatorSet/5.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.xlinkit.com/OperatorSet/5.0"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="OperatorSet">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="header" minOccurs="0"/>
    <xs:choice maxOccurs="unbounded">
     <xs:element name="Operators">
      <xs:complexType>
       <xs:attribute name="href" type="xs:string" use="required"/>
      </xs:complexType>
     </xs:element>
     <xs:element name="OperatorDefinition">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="description" minOccurs="0" maxOccurs="1">
         <xs:complexType mixed="true">
          <xs:sequence>
           <xs:any namespace="http://www.w3.org/1999/xhtml"
                   processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required"/>
          <xs:attribute name="type" type="ParameterType" use="required"/>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
       <xs:attribute name="name" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:unique name="ParamUnique">
       <xs:selector xpath="./param"/>
       <xs:field xpath="@name"/>
      </xs:unique>
     </xs:element>
    </xs:choice>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="impl" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="header">
  <xs:complexType>
   <xs:sequence>
    <xs:choice maxOccurs="unbounded">
     <xs:element name="author" type="xs:string"/>
     <xs:element name="description">
      <xs:complexType mixed="true">
       <xs:sequence>
        <xs:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="project" type="xs:string"/>
     <xs:element name="comment" type="xs:string"/>
     <xs:any namespace="http://www.xlinkit.com/Metadata/5.0" processContents="skip"/>
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:simpleType name="ParameterType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="int"/>
   <xs:enumeration value="string"/>
   <xs:enumeration value="nodeList"/>
   <xs:enumeration value="node"/>
   <xs:enumeration value="var"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

9.5 Macro Language Syntax (XML Schema)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.xlinkit.com/Macro/5.0"
           xmlns="http://www.xlinkit.com/Macro/5.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="definitions">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="header" minOccurs="0"/>
    <xs:element ref="macro" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="macro">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="header" minOccurs="0"/>
    <xs:element ref="param" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="output"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="output">
  <xs:complexType>
   <xs:sequence>
    <xs:any namespace="http://www.xlinkit.com/ConsistencyRuleSet/5.0"
            processContents="skip"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="param">
  <xs:complexType>
   <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="header">
```

```
  <xs:complexType>
   <xs:sequence>
    <xs:choice maxOccurs="unbounded">
     <xs:element name="author" type="xs:string"/>
     <xs:element name="description">
      <xs:complexType mixed="true">
       <xs:sequence>
        <xs:any namespace="http://www.w3.org/1999/xhtml"
                processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="project" type="xs:string"/>
     <xs:element name="comment" type="xs:string"/>
     <xs:any namespace="http://www.xlinkit.com/Metadata/5.0"
             processContents="skip"/>
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Refences

[1] C. Nentwich, W. Emmerich, A. Finkelstein. Flexible Consistenc Checking. Research Note, University College London, Dept. of Computer Science, 2001. Submitted for Pubblication.

[2] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0 W3C Recommendation, 16 November 1999, http://www.w3.org/TR/xpath.

[3] S. DeRose and E. Maler and D. Orchard. XML Linking Language (XLink) Version 1.0. W3C Recommendation http://www.w3.org/TR/xlink/, World Wide Web Consortium, June 2001.

[4] D. E. Knuth. Semantics of Context-Free Languages. *Mathematical Systems Theory*, 2(2):127-145, 1968.

[5] U. Kastens. Ordered Attributed Grammars. *Acta Informatica*, 13(3):229-256, 1980.

[6] U. Kastens and W. M. Waite. Modularity and reusability in attribute grammars. *Acta Informatica*, 31:601-127, 1991.

[7] A. N. Habermann and D. Notkin. Gandalf: Software Development Environments. *IEEE Transactions on Software Engineering*, 12(12): 1117-1127, 1986.

[8] T. W. Reps and T. Teitelbaum. The Synthesizer Generator. *ACM SIGSOFT Software Engineering Notes*, 9(3): 42-48, 1984.

[9] M. Nagl, editor. *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, volume 1170 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[10] P. Borras, D. Clement, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: The System. *ACM SIGSOFT Software Engineering Notes*, 13(5):14-24, 1988, ACM Press.

[11] An Object-Oriented Language for Specification of Syntax Directed Tools. Proc. of the 8th Int. Workshop on Software Specification and Design, 26-35, 1996. IEEE Computer Society Press.

[12] J. B. Warmer and A. G. Kleppe. The Object Constraint Language: Precise Modeling with UML. Addison Wesley, 1999.

[13] J. Clark, XSL Transformations (XSLT). Technical Report http://www.w3.org/TR/xslt, World Wide Web Consortium, November, 1999.

[14] W. Emmerich and E. Ellmer and H. Fieglein, TIGRA -- An Architectural Style for Enterprise Application Integration. In Proc. of the 23rd Int. Conf. on Software Engineering, pages 567-576. IEEE Computer Society Press, 2001.

[15] R. Jelliffe. The Schematron Assertion Language 1.5. Technical Report, GeoTempo Inc., October 2000.

[16] FpML Version 1.0 Recommendation, May 14, 2001, http://www.fpml.org/spec/2001/rec-fpml-1-0-2001-05-14/

[17] XML Schema Part 0: Primer W3C Recommendation, 2 May 2001, http://www.w3.org/TR/xmlschema-0/