# xlinkit: A Consistency Checking and Smart Link Generation Service[1]

Christian Nentwich, Licia Capra, Wolfgang Emmerich
and Anthony Finkelstein

**xlinkit.com white paper**

**Abstract**

xlinkit is a lightweight application service that provides rule-based link generation and checks the consistency of distributed web content. It leverages standard Internet technologies, notably XML and XLink. xlinkit can be used as part of a consistency management scheme or in applications that require smart link generation, including portal construction and management of large document repositories. In this paper we show how consistency constraints can be expressed and checked. We describe a method for generating links based on the result of the checks and we give an account of our content management strategy. We present the architecture of our service and the results of a substantial 'real-world' evaluation.

[1]Department of Computer Science, University College London, Research Note RN/00/06. Submitted for publication.

## 1 Overview

This paper describes xlinkit, a lightweight application service that provides rule-based link generation and checks the consistency of distributed web resources. The paper is supplemented by the on-line demonstrations at `http://www.xlinkit.com`.

The operation of xlinkit is quite simple. It is given a set of distributed XML resources and a set of potentially distributed rules that relate the content of those resources. The rules express consistency constraints across the resource types. xlinkit returns a set of XLinks, in the form of a linkbase, that support navigation between elements of the XML resources. The precise link generation behaviour is determined by link building annotations on the rules.

xlinkit leverages standard Internet technologies. It supports document distribution and can support multiple deployment models. It has a formal basis and evaluation has shown that it scales, both in terms of the size of documents and in the number of rules.

With this thumbnail description in mind it is easiest to motivate and to explain xlinkit by reference to a simple example. This example is given in Section 3 below. It is preceded by some essential background.

## 2 Background

The paper assumes some familiarity with XML (Extensible Markup Language) [6] and XSLT (Extensible Stylesheet Language Transformations) [7]. It also makes significant reference to technologies related to XML, specifically XLink [10], the XML linking scheme and XPath [8], which supports addressing of the internal structures of an XML resource. We make some reference in the paper to the XML DOM (Document Object Model) [2], the API for XML resources though this paper does not require a detailed understanding of it. For details of XML and related technologies good sources are the World Wide Web Consortium (W3C) and the Organisation for the Advancement of Structured Information Standards (OASIS).

As XLink and XPath are less well known than XML it is worthwhile providing a brief orientation. XLink is an XML markup language that provides additional linking functionality for web resources, it effectively brings much of the power of Open Hypermedia models such as Hytime [20] to the web. HTML links are highly constrained, notably: they are unidirectional and point-to-point; have a limited range of behaviours; link only at the level of files unless an explicit target is inserted in the destination resource; and, most significantly, are embedded within the resource, leading to maintenance difficulties and the familiar 404: Not Found error that results from dangling links. XLink addresses these problems allowing any XML element to act as a link and allowing the user to specify complex link structures and traversal behaviours and to add metadata to links. Most importantly for what follows in this paper XLinks can exist in "linkbases" and such "extended links" can be managed separately from the resources they link. Linkbases can be selectively applied to sets of resources. An XML resource can be viewed as a tree, an XPath expression specifies traversals of the document tree and choice of its internal parts based on properties such as element types, attribute values, character content, and relative position. When combined with XLinks, an XPath expression can address not only a resource but a specified element within that resource.

## 3 Example

We now introduce an example which is used throughout the paper. Wilbur's Bike Shop sells bicycles and makes information about their company available on the Internet and on a corporate intranet. Wilbur's use XML for web publication and information exchange.

The information collected by Wilbur's is spread across several web resources:

- a product catalogue – containing product name, product code, price and description;
- advertisements – containing product name, price and description;
- customer reports – listing the products purchased by particular customers;
- service reports – giving problems with products reported by customers.

Wilbur's has only one product catalogue, but many advertisements, customer reports and service reports. The information is distributed across different web servers.

It should be clear that much of this information, though produced independently, is closely related. For example: the product names in the advertisements and those in the catalogue; the advertised prices and the product catalogue prices; the products listed as sold to a customer and those in the product catalogue; the goods reported as defective in the service reports and those in the customer reports; and so on.

Relationships among independently evolving and separately managed resources can give rise to inconsistencies. This is not necessarily a bad thing but it is important to be aware of such inconsistencies and deal with them appropriately. In view of this, Wilbur's would like to check their resources to establish their position.

For the example which follows we will concentrate on the relationship between the product catalogue and the advertisements. Figure 1 shows an extract from the product catalogue and Figure 2 shows a sample advertisement. Samples of the other resources and can be found in Appendix A.

```
<Catalogue>
     <Product>
         <Name>Haro Shredder</Name>
         <Code>B001</Code>
         <Price currency="sterling">349.95</Price>
     </Product>
     <Product>
         <Name>Dyno NFX</Name>
         <Price currency="sterling">119.95</Price>
         <Code>B003</Code>
     </Product>
</Catalogue>
```

Figure 1: Wilbur's product catalogue extract

This relationship requires a check:

- *Are all the product names in the advertisements the same as in the catalogue?*

```
<Advert>
     <ProductName>Dyno NFX</ProductName>
     <Price currency="sterling">119.95</Price>
     <Description>BMX Bike. Dyno expert frame.
                  Coaster brake or freewheel.
     </Description>
</Advert>
```

Figure 2: Wilbur's sample advertisement

Other checks might include :

- *Do the advertised prices and the product catalogue prices correspond?*
- *Are the products listed as sold to a customer in the product catalogue?*
- *Did we sell the goods reported as problematic to the customer reporting the problem?*

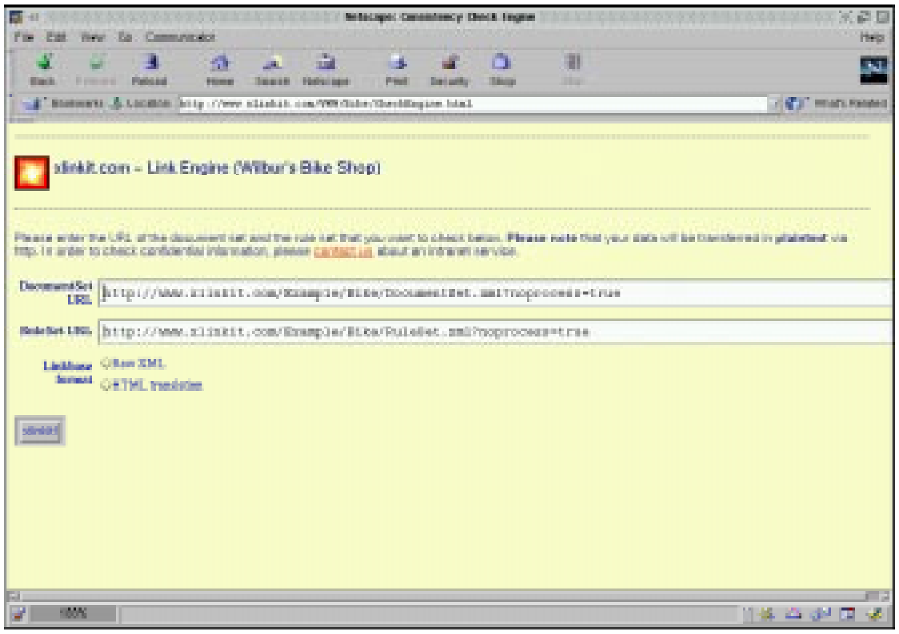
We define these checks as rules and assemble them in a rule set. The rule set could consist of further distributed rule sets. We describe our rule language and the assembly of rule sets in the following sections.

The document set is the collection of documents we want to check against the rules. In the same way as the rule set the document set can consist of further distributed document sets. In this example we have a set of adverts, a set of customers and a set of service reports. Both the document set and the rule set are identified by URLs.

The rules are annotated with information specifying the sort of links we want built when the rule holds or when it does not hold. Thus:

- *Are all the product names in the advertisements the same as in the catalogue?* Result: Links between the product advertised and the corresponding product entry in the catalogue. If there is no corresponding product in the catalogue, the advertisement is linked to the rule for diagnostic purposes.
- *Do the advertised prices and the product catalogue prices correspond?* Result: Links between the advertisement and the rule for diagnostic purposes only when the advertised price does not match.
- *Are the products listed as sold to a customer in the product catalogue?* Result: Links between the product entry in the customer record and the corresponding product entry in the catalogue. If there is no corresponding product in the catalogue, the customer record entry is linked to the rule for diagnostic purposes.
- *Did we sell the goods reported as defective to the customer reporting the problem?* Result: Links between the product with the problem and the product entry in the customer record. If there is no corresponding product entry in the customer record, the product entry is linked to the rule for diagnostic purposes.

The checks are made by submitting the document set and the rule set URLs to the check engine which makes the checks and returns the URL of an XLink linkbase. Figure 3 shows the submission form that is passed to the check engine. Because the linkbase is itself XML we can apply a stylesheet to render it


Netscape: Consistency Check Engine
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location http://www.xlinkit.com/WWW/Bike/CheckEngine.html
What's Related

xlinkit.com – Link Engine (Wilbur's Bike Shop)

Please enter the URL of the document set and the rule set that you want to check below. **Please note** that your data will be transferred in **plaintext** via http. In order to check confidential information, please contact us about an intranet service.

| | |
|---|---|
| DocumentSet URL | http://www.xlinkit.com/Example/Bike/DocumentSet.xml?noprocess=true |
| RuleSet URL | http://www.xlinkit.com/Example/Bike/RuleSet.xml?noprocess=true |
| Linkbase format | Raw XML / HTML translation |

submit

Figure 3: Web submission form

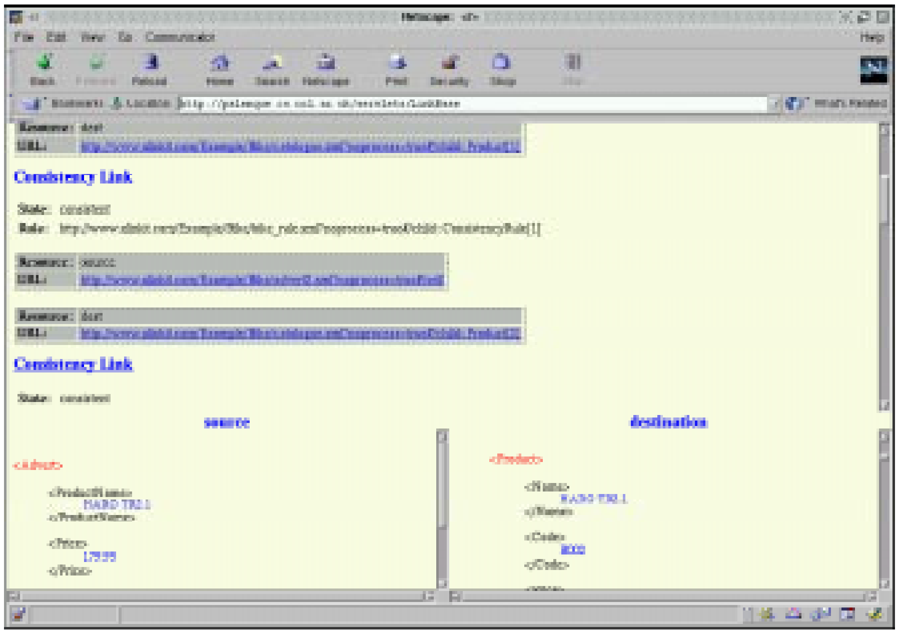
in HTML for review or deliver in source XML. Figure 4 shows an HTML representation of a linkbase. Users can click on the consistency links; a servlet will then retrieve the two XML files that are being linked, convert them to HTML and highlight the linked elements.

Most "off-the-shelf" browsers do not yet implement support for extended links of the sort that xlinkit produces, only limited support is available for simple links, that is XLinks embedded in documents (in-line as distinct from out-of-line) from browsers such as Amaya [9] or the latest releases of Mozilla [21] . One way to make the linkbase navigable is to first "fold" it into the resources. This entails applying an XLink processor to fetch the resources referenced in the linkbase, convert the extended links into simple links and integrate them into the resources in the appropriate place. We use a standard XLink processor, X2X from Empolis [15] for this purpose – there are a number of other similar processors available. The resulting XML resources can then be handled in the familiar manner, that is by applying stylesheets to render a browsable hyperlinked HTML presentation. In the case of our example we deliver a product catalogue site that links to the advertisements.

## 4 Rule Language

This section presents our new set-based rule language – cheXML – which serves to express consistency constraints between distributed documents. We outline a simple formal basis for the language and formalise our example rule.

To do this we use a notation for evaluating XPath expressions and for the formalisation of the DOM which


Resource: dest
URL: [illegible]

Consistency Link

State: consistent
Rule: [illegible]ConsistencyRule[1]

Resource: source
URL: [illegible]

Resource: dest
URL: [illegible]

Consistency Link

State: consistent

source

```
<Advert>
    <ProductName>
        HARD TR2.1
    </ProductName>
    <Price>
        179.99
    </Price>
```

destination

```
<Product>
    <Name>
        HARD TR2.1
    </Name>
    <Code>
        B002
    </Code>
```

Figure 4: Sample linkbase in HTML

is due to Wadler [31]. In order to make the explanation as easy to read as possible, we will introduce some further simplifying notation. For the full grammar and semantics the reader is referred to Appendix B.

- Uppercase letters $A, B, C, \ldots$ correspond to sets of DOM nodes
- Lowercase letters $a, b, c, \ldots$ correspond to DOM nodes
- The function $\mathcal{S}[\![p]\!]_x$ creates a set of nodes by evaluating the path expression $p$ with $x$ as the context node. (For example, $\mathcal{S}[\![Price/@currency]\!]_{/Advert}$ in Figure 2 would return a one-element set containing a text node with the string `''sterling''`). $/$ refers to the root node.

When specifying a rule, we want to express a relationship of one set of nodes with one or more other sets of nodes. For example, the set of all `Advert` elements in Wilbur's advertisements has to be consistent with the set of all `Products` in their product catalogue.

We will rephrase the question *"Are all the product names in the advertisement the same as in the catalogue?"* more formally as an assertion: *"For all Advert elements, there exists a Product element in the Catalogue element where the ProductName subelement of the former equals the Name subelement of the latter".* If this condition holds, a consistent relationship exists between the Advert element and the Product element being considered. Otherwise, the Advert element is inconsistent with respect to our rule.

We can denote the sets of elements to be checked as followed:

- Let $S$ be the set of `Advert` elements: $S = \mathcal{S}[\![/Advert]\!]_/$. $S$ will be our "source" set.

- Let $D$ be the set of `Product` elements: $D = \mathcal{S}[\![/Product]\!]_{/}$. $D$ will be our "destination" set.

| **Source set (S)** | **Destination set(D)** |
|---|---|
| Advert (ProductName="Dyno NFX") | Product (Name="Haro Shredder") |
| Advert (ProductName="Haro Shredder") | Product (Name="Dyno NFX") |
| Advert (ProductName="Phunky Brake") | |

Table 1: Source and destination node sets

Table 1 shows the two sets selected after evaluating the path expressions. For convenience, the relevant subelements of the elements we are going to check are shown in brackets. We can now express the rule formally as $\forall s \in S(|D/(\mathcal{S}[\![ProductName]\!]_s = \mathcal{S}[\![Name]\!]_d)| > 0)$, which reads "For all $s$ in $S$, the size of the set created by restricting $D$ using the formula $\mathcal{S}[\![ProductName]\!]_s = \mathcal{S}[\![Name]\!]_d$ must be greater than zero". $s$ will be bound to the current source node being processed by the quantifier and $d$ will be bound to the current node in the destination set being filtered. If the size of the filtered set is zero, the current source node is inconsistent with respect to this rule, otherwise it is consistent.

As an example of how the approach can be implemented, consider again the documents in Figure 1 and Figure 2 and the source and destination sets shown in Table 1.

The check engine steps through the source set and processes the first entry. According to our rule, the destination set will be filtered by the expression $(\mathcal{S}[\![ProductName]\!]_s = \mathcal{S}[\![Name]\!]_d)$. The resulting filtered set is shown in Table 2. Clearly the cardinality of the filtered set satisfies the condition of being greater than zero, as specified in our rule. As a consequence, the current source element is *consistent* with respect to our rule.

| **Source set** | **Filtered set** |
|---|---|
| $*$ Advert (ProductName="Dyno NFX") | Product (Name="Dyno NFX") |
| Advert (ProductName="Haro Shredder") | |
| Advert (ProductName="Phunky Brake") | |

Table 2: Filtered destination set (* is the current source node)

The check engine then continues to step through the source set and eventually reaches the last Advert element. Again, the filtering expression is applied and the resulting filtered set can be seen in Table 3. This time, the filtered set is empty and the condition $|D/(\mathcal{S}[\![ProductName]\!]_s = \mathcal{S}[\![Name]\!]_d)| > 0$ is not satisfied. The current source element is thus *inconsistent* with respect to our rule.

| **Source set** | **Filtered set** |
|---|---|
| Advert (ProductName="Dyno NFX") | |
| Advert (ProductName="Haro Shredder") | |
| $*$ Advert (ProductName="Phunky Brake") | |

Table 3: Empty filtered destination set (* is the current source node)

Some rules require the added power of a transitive closure operator. For example, if Wilbur's bikeshop were to offer composite products such as bikes made from several components, they might want to check that composite components are not parts of themselves. It is then not enough to check whether a part of a component equals the component itself, since the part may itself be made up from several parts - the transitive closure of the part-whole hierarchy has to be computed.

We specify the operator $closure(x, p_1, p_2)$ that takes a node $x$ which is part of some nodeset $X$ and two xpath expressions $p_1$ and $p_2$. Initially, only $x$ is in the closure. Then, for all nodes $n$ in $X$ and all nodes $c$ in the closure, if $\mathcal{S}[\![p_1]\!]_n = \mathcal{S}[\![p_2]\!]_c$ then $n$ is added to the closure. This process is repeated until the closure set does not change anymore.

A presentation of the complete grammar and semantics for our language is given in Appendix B.

## 5 Link Generation

Having established whether a rule holds or has been violated, we have to record this information. This section explains our handling of consistency information.

Our approach is always to take a tolerant view of inconsistency – inconsistencies are not always accidental or undesirable and we do not force their immediate resolution; instead we provide diagnostic information by creating links. This approach has its roots in previous work which will discussed in the related work section.

Table 2 shows the information obtained previously by filtering a destination set according to some rule. The star next to the `Advert` element in the source set shows the source node currently being processed. As outlined previously, the question of whether an inconsistency or a consistency has been found is answered by comparing the size of the filtered set to some constant value.

In the example rule, if the cardinality of the filtered set is greater than zero we have found a node consistent with our current source node. The strategy is then to link the current source node to all entries in the filtered set. However, if the filtered set was empty and the comparison operator failed, there would be nothing to link to. Furthermore, by the definition of the consistency rule, this would unveil an inconsistency. The strategy is then to link the current source node to the definition of the rule in the rule file – the node is inconsistent with respect to the rule. Using this strategy, we can later provide an overview the elements that are inconsistent with each rule.

Figure 5 shows the XLinks generated after processing the filtered set in Table 2 and the inconsistent link generated after processing the set in Table 3. The first link picks out the root element in the advert file as the source element and links it to the first `Product` element, which it is consistent with. The second link picks the root element in the third advert file as the source and links it to the rule with the `id` attribute $r1$ in the rule file.

If people want to choose different strategies, we offer a choice of what to do when a consistency is found (the CMode) and what to do when an inconsistency is found (the IMode). Table 4 presents all options available. Experience suggests that the most popular strategy is to link consistent elements (CMode=C) and to generate links to the consistency rule for inconsistent elements (IMode= IX). However, other sensible strategies can be used, for example (CMode=, IMode=I), which ignores consistent cases and links inconsistent cases for diagnostic purposes, or (CMode=CX, IMode=IX), which links to the rule in all cases.

```
<LinkBase docset="DocumentSet.xml" ruleset="RuleSet.xml">
    <ConsistencyLink ruleid="bike_rule.xml#/id('r1')">
        <State>consistent</State>
        <Locator xlink:href="advert1.xml#/Advert"
                 xlink:label="source" xlink:title=" "/>
        <Locator xlink:href="catalogue.xml#//Product[1]"
                 xlink:label="dest" xlink:title=" "/>
    </ConsistencyLink>
    <ConsistencyLink ruleid="bike_rule.xml#/id('r1')">
        <State>inconsistent</State>
        <Locator xlink:href="advert3.xml#/Advert"
                 xlink:label="source" xlink:title=" "/>
        <Locator xlink:href="bike_rule.xml#/id('r1')"
                 xlink:label="dest" xlink:title=" "/>
    </ConsistencyLink>
</LinkBase>
```

Figure 5: Wilbur's Bike Shop – Consistency links

| **CMode** | **Consistency found**<br>(consistent link generation) | **IMode** | **Inconsistency found**<br>(inconsistent link generation) |
|---|---|---|---|
| C | Link current source and filter | I | Link current source and filter |
| CX | Link source to rule | IX | Link source to rule |
| | Generate no links | | Generate no links |

Table 4: Link mode table

## 6 XML Implementation

We now present an XML encoding for our language. Encoding the language in XML has the advantage of blending more uniformly into the environment where it is going to be used. It also allows us to treat the rule files as targets which can be checked by other rules.

Presenting the encoding of the whole language is beyond the scope of this paper and the interested reader is referred to Appendix E for the complete DTD. Instead, we will present two example rules expressing constraints for Wilbur's Bike Shop.

Our first example will be the now familiar rule *"For all Advert elements, there exists a Product element in the Catalogue element where the ProductName subelement of the former equals the Name subelement of the latter".* Figure 6 shows a rule file which specifies this rule in XML format.

A rule consists of three main parts: the first entry in a rule is a `Description` element (not shown in the figure) which is a natural language description of the rule that can be used for diagnosis. The following `SetDefinition` elements contain an XPath expression, which will be used to build up node sets from the target documents. Each node set is given a unique identifier, which can be referred to in the actual rule.

The most important part of the rule, the actual consistency constraint, is contained in the `Forall` element. The `setid` parameter specifies which node set will be treated as the source set whose elements will be checked for consistency. Contained inside the `Forall` element is the *s*et operator `SizeNotEqual`. This operator compares the cardinality of its first argument to the cardinality of its second argument. If

```
<ConsistencyRule id="r1">
    <SetDefinition id="source">
        /Advert
    </SetDefinition>

    <SetDefinition id="destination">
        /Catalogue/Product
    </SetDefinition>

    <Forall setid="source">
        <SizeNotEqual cmode="C" imode="IX">
            <Filter setid="destination">
                <Equal>
                    <XPathSource value="ProductName"/>
                    <XPathFilter value="Name"/>
                </Equal>
            <Integer value="0"/>
        </SizeNotEqual>
    </Forall>
</ConsistencyRule>
```

Figure 6: Consistency rule with link modes

they are not equal the current source node is consistent, otherwise it is inconsistent. Since the decision about consistency or inconsistency is made by the `SizeNotEqual` element, the linking strategy attributes `cmode` and `imode` are also specified there. In our sample rule, we link consistent elements (C) and link inconsistent source elements to the rule (IX).

The two arguments contained in the `SizeNotEqual` operator are both filtered sets: the `Filter` argument actually produces a filtered set from a node set while the `Integer` argument contains a fixed size set of cardinality 0. Similar to the `Forall` element, the `Filter` element must also specify which set to filter since there can be multiple destination sets.

Inside the `Filter` operator, any boolean expression can be specified, in this case only an equality operator is used. The two values that will be compared for equality by the filter are retrieved by the `XPathSource` and `XPathFilter` operators. Both `XPathSource` and `XPathFilter` take as an argument a relative XPath expression. `XPathSource` uses the current source node as the context node – the source set is the set specified by the closest `Forall` operator in the current scope. `XPathFilter` uses the current destination node as the context – the destination set is the set specified by the closest `Filter` operator in the current scope.

Our XML implementation includes a further feature called *cursor variables* which can be used to speed up the checking process with large destination sets. Consider again Table 1. For a destination set of size $n$ the filter expression will have to decide $n$ times which nodes to discard. In some cases, we can make use of XPath's ability to apply predicates to pre-filter our destination set in order to make it smaller. We rewrite the rule in Figure 6 as in Figure 7.

When evaluating the `Filtered` expression, the current source node being considered by `Forall` will be bound to the `$source` variable in the destination set path expression. We then regenerate the destination nodeset accordingly, making the XPath processor picking out only those nodes from the tree which are relevant. This leads to considerable time savings with large destination sets but can introduces penalties

```
<ConsistencyRule id="r1">
    <SetDefinition id="source">
        /Advert
    </SetDefinition>

    <SetDefinition id="destination">
        /Catalogue/Product[Name=$source/ProductName]
    </SetDefinition>

    <Forall setid="source">
        <SizeNotEqual cmode="C" imode="IX">
            <Filtered setid="destination"/>
            <Integer value="0"/>
        </SizeNotEqual>
    </Forall>
</ConsistencyRule>
```

Figure 7: Simplification with cursor variables

with small sets.

We have written a stylesheet that transforms the rules from XML to HTML to make them more accessible for browsing. The boolean expressions inside the filter are translated from their XML prefix form back into infix. Figure 8 shows the translated rules.

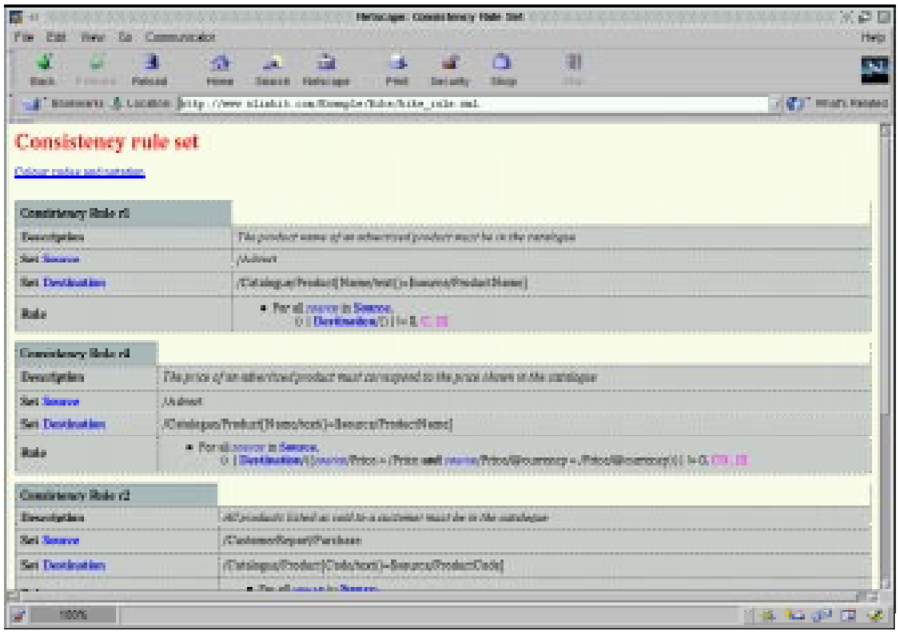
## 7 Content Management

The selection of documents and rules to be checked against each other has to be managed. It is infeasible to check every document against every rule and it is certainly not necessary to check every document every time. Instead, we use document sets, which contain a selection of documents taken from resources, and rule sets which contain several rules. A document set together with a rule set can then be submitted for checking.

Figure 9 shows a sample document set. Document sets form a hierarchy in that they consist of documents and further document sets. In the figure, the `DocFile` directive is used to add a file directly into the set while the `Set` directive includes further sets. At check time, the hierarchy is flattened and resolved into a single set. To find out whether a document needs to be checked against a rule, we check if the XPath expressions in the rule's set definition can be applied.

Our method of retrieval of document information is not limited to XML content stored in files. Instead, we abstract from the underlying data store by providing *fetcher* classes. It is the responsibility of a fetcher to liaise with some data store in order to provide a DOM tree representation of its content. By default, data are retrieved from XML files using the `FileFetcher` class, however user-defined classes can override this behaviour. Using this mechanism, it is possible to read in content that follows a legacy format and translate it into a DOM tree, to read data from network sockets or to construct a DOM tree from a relational or object-oriented database.

As a proof of concept, we provide a JDBC fetcher, which executes a query on a database and translates the resulting table into a DOM tree. Figure 10 shows a version of Wilbur's bikeshop document set where the



# Consistency rule set

Colour codes explanation

| Consistency Rule r1 | |
|---|---|
| Description | *The product name of an advertised product must be in the catalogue* |
| Set Source | /Advert |
| Set Destination | /Catalogue/Product[Name/text()=$source/ProductName] |
| Rule | • For all source in Source, ◦ \| Destination(:) \| != 0. |

| Consistency Rule r4 | |
|---|---|
| Description | *The price of an advertised product must correspond to the price shown in the catalogue* |
| Set Source | /Advert |
| Set Destination | /Catalogue/Product[Name/text()=$source/ProductName] |
| Rule | • For all source in Source, ◦ \| Destination/(:)source/Price = /Price **and** source/Price/@currency = /Price/@currency) \| != 0. |

| Consistency Rule r2 | |
|---|---|
| Description | *All products listed as sold to a customer must be in the catalogue* |
| Set Source | /CustomerReport/Purchase |
| Set Destination | /Catalogue/Product[Code/text()=$source/ProductCode] |
| | • For all source in Source, |

Figure 8: Rules in HTML

service reports have been put into a relational database. The `fetcher` attribute in the `DocFile` directory overrides the default `FileFetcher`, which can be omitted, to select the `JDBCFetcher` class.

The JDBC fetcher class executes the SQL query on the database and transforms the resulting relational table into a DOM tree. Figure 11 shows a sample table of service reports fetched from Wilbur's database by executing the JDBC query from the document set. Shown below the table is the XML representation, containing one `row` element for every row stored in the table and using the column name data from the data dictionary for the element names inside the rows.

Rule sets are managed in a similar fashion. A rule set contains references to rules and further rule sets. Figure 12 shows a sample rule set. A `RuleFile` element is used to specify a rule file to load and an `xpath` attribute specifies which rules from that file to actually include. The path `/ConsistencyRule-`

```
<DocumentsSet name="BikeDoc">
        <Description>Wilbur’s complete collection</Description>

        <DocFile href="catalogue.xml"/>

        <Set href="Adverts.xml"/>
        <Set href="Customers.xml"/>
        <Set href="Services.xml"/>
</DocumentsSet>
```

Figure 9: Sample document set

```
<DocumentsSet name="BikeDoc">
    <Description>Wilbur’s complete collection</Description>

    <DocFile href="catalogue.xml"/>

    <Set href="Adverts.xml"/>
    <Set href="Customers.xml"/>

    <DocFile fetcher="JDBCFetcher"
        href="jdbc:mysql://www.xlinkit.com/testdb?user=wilbur#select * from report"/>
</DocumentsSet>
```

Figure 10: Document set with SQL resource

```
+------------------------------------+-------------+--------------------------+
| productname                        | productcode | description              |
+------------------------------------+-------------+--------------------------+
| HARO SHREDDER                      | B001        | Found a problem in ...   |
| HARO TR2.1                         | B002        | Found a problem while... |
+------------------------------------+-------------+--------------------------+
<rows>
     <row>
          <productname>HARO SHREDDER</productname>
          <productcode>B001</productcode>
          <description>Found a problem in ...</description>
     </row>
     <row>
          <productname>HARO TR2.1</productname>
          <productcode>B002</productcode>
          <description>Found a problem while...</description>
     </row>
</rows>
```

Figure 11: Relational table XML representation

`Set/ConsistencyRule` will match all `ConsistencyRule` elements included in the rule file. If that is not desired, a more constrained path such as

$$/\texttt{ConsistencyRuleSet}/\texttt{ConsistencyRule}[@\texttt{id} =' \texttt{r1}']$$

could be used, which only loads the rule whose `id` attribute is equal to $r1$.

## 8 Architecture

We have implemented a publicly accessible, free to use Internet service. Our architecture is very simple. Figure 13 shows its basic structure.

We have implemented the check engine as a Java Servlet, which is hosted on an Apache web server running the Apache JServ servlet engine. Users are presented with the form shown in Figure 3 to enter the URL of the document set and rule set to be checked.

```
<RulesSet name="BikeRules">
        <Description>Rules related to the Bike environment</Description>

        <RuleFile href="bike_rule.xml"
          xpath="/ConsistencyRuleSet/ConsistencyRule"/>
</RulesSet>
```

Figure 12: Sample rule set

When the form is submitted, a new servlet instance is created to deal with the request. The servlet itself uses the Xerces XML parser from the Apache XML project to parse the documents and rule files. After checking the rules, the servlet writes an XML file containing the generated links to the web server's local storage. The servlet then generates a result page that contains the URL of the link base and returns it back to the browser client. The input form also gives the user a choice whether to return the raw XML file containing the links or to add a processing directive for it to be translated into HTML using a stylesheet. Please refer back to Figure 4.

## 9 Evaluation

This section presents two sample case studies that we used to evaluate the expressiveness of our rule language and the scalability of our implementation. Our major goal was to find out if xlinkit can be applied to a real-world example. In addition, we also wanted a "stress-test" scenario for performance, scalability and expressiveness.

Our first study checks the consistency of course syllabus information and the second study performs a validation of multiple software engineering documents.

The Department of Computer Science at University College London recently introduced a new curriculum and associated course syllabi. In order to provide high quality information in the wide variety of different representations required, it decided to adopt XML as a common format. The system has to hold a curriculum and provide links to the syllabuses for students, depending on which degree programme they are pursuing. Figure 14 shows a sample abreviated syllabus file for a course. Each course is held in a separate XML file. The curricula for degree programmes are kept in a single file. For each degree programme, the mandatory and optional courses are listed, grouped by the year in which they can be selected. Figure 15 shows a fragment from the curricula file.

The process of syllabus development is highly decentralised, with different people providing additions and corrections to course syllabi. Curriculum files contain information related to the individual syllabus files. For example, course codes mentioned in the curriculum files have to be part of a syllabus definition. Altogether, ten rules where identified as necessary to preserve the consistency of the system. The complete list of rules can be found in Appendix D.

It is desirable for navigation purposes to provide hyper-links from the curriculum to individual courses. However, providing links from the curriculum file to all 48 syllabus files would be error prone as files get deleted and courses renamed. It is preferable to use the semantically equivalent information in the files (e.g. the course codes) to generate the hyperlinks automatically. We used our xlinkit to achieve both goals.

Figure 16 shows the time used for checking each rule against all 52 documents. Checking was performed

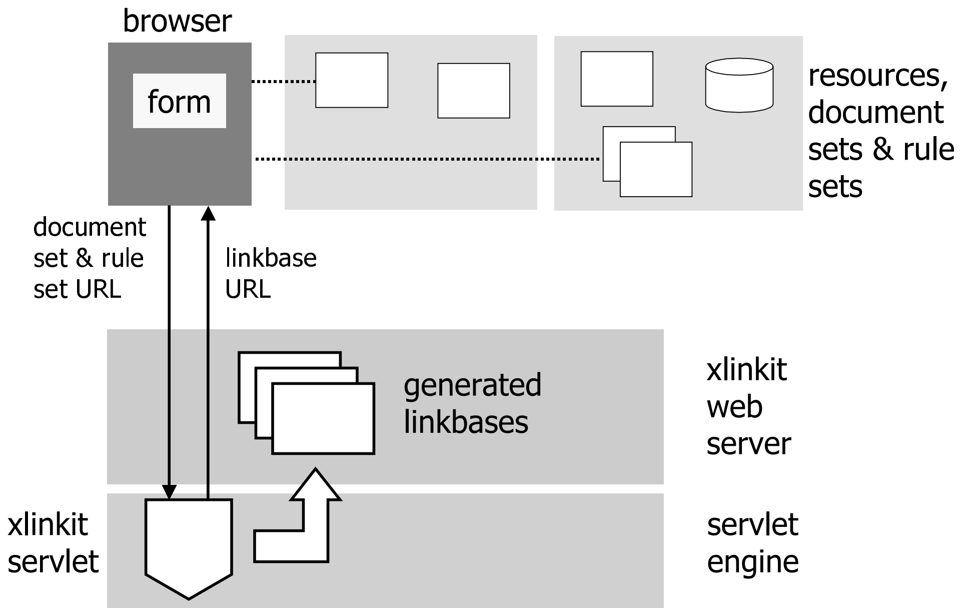


Figure 13: Architecture overview

on a 700 Mhz Intel machine with 128Mb of RAM, running Linux and the IBM JDK 1.3. The total checking time was $5.2$ seconds, with no single rule taking longer than $2$ seconds to check. In total, $410$ consistent and $14$ inconsistent links were generated.

Our second goal was to provide a fully linked HTML version of the department's curriculum to be browsed by staff and students. One of the rules for the curriculum is that every course listed in the curriculum must have a syllabus definition. If the rule is satisfied, a consistent link is generated from the course entry in the curriculum to the syllabus defining the course. We used X2X, as in the example section, to fold all consistent links from this rule back into the XML file containing the curriculum. We then only had to provide a simple XSL stylesheet that transforms the XML file and simple links into an HTML representation, as shown in Figure 17.

Our second case study uses our rule language to express some of the semantic constraints of the UML [24]. The typical scenario for this study is a distributed development team working on the same model and producing their own additions and copies of documents. If frequent merging of the documents is not feasible, for example due to geographical separation, checks can be used to ensure consistency.

We have expressed about half of the constraints from the UML specification in our language. Seven sample rules, some of which require transitive closure were tested against a set of real-world models from a large bank, for whom we have been consulting. The models are stored in the XMI [25] format defined by the OMG, a standard exchange format for models that comply with the Meta-Object Facility [23] specification.

```
<syllabus>
     <identity>
         <title>Concurrency</title>
         <code>3C03</code>
         <summary>The principles of concurrency
                  control and specification</summary>
     </identity>

     <teaching>
         <normal_year>3</normal_year>
         <term>1</term>
         <taught_by>
             <name>Wolfgang Emmerich</name>
             <pct_proportion>100</pct_proportion>
         </taught_by>
     </teaching>

     <subject>
         <prerequisites descr="">
             <pre_code>1B11</pre_code>
         </prerequisites>
     </subject>
</syllabus>
```

Figure 14: Sample shortened syllabus file in XML

In total, there were 19 XMI files, containing 12192 model elements, of which 349 were classes.

The seven rules that were selected as examples to work on are given in Appendix D and were chosen in order to support comparison with previous work, as described below. Figure 18 shows the time taken for checking each rule. Checking was performed on a 650 Mhz Intel machine with 384Mb of RAM, running Linux and the IBM JDK 1.3. The total checking time was 9.3 minutes and maximum RAM use around 250 megabytes. 1958 consistent links and 207 inconsistent links were generated.

As the UML model we were checking was large and complex, we believe that the checking times are reasonable. Full checks on all model files would not always be necessary, enabling users to bring down the checking time further by excluding documents that have not changed from their document sets. Integration with a change management system could help to automate this process.

The rather heavy memory consumption will however turn out to be a problem if much larger amounts of data are to be checked. The present check engine parses all XML files at startup and retains their DOM tree representation in memory throughout the process. As the amount of data increases this becomes less feasible. Proper resource management, parsing files on demand and ordering of the rules to minimise the amount of parsing will be required.

While it was possible to express all the example rules for XMI without problems, we have found problems with the expressiveness of our language when translating a small number of rules from the UML specification. As an example, consider the constraint that "*if an Association has three or more AssociationEnds, then no AssociationEnd may be an aggregation or composition*". Represented in logic, this rule is roughly of the form $\forall a \neg \exists x \exists y \exists z (x \neq y \wedge x \neq z \wedge y \neq z \wedge a \in \{\texttt{composition,aggregation}\})$. However, rules in our language are restricted to the form $\forall x \exists y$. It is not possible to introduce additional quantifiers and to express relationships among more than two nodes at the same time. Therefore, without resorting to

```
<Curricula>
    <Curriculum>
        <Programme>
            <Title>CS</Title>
            <Award>BSc</Award>
        </Programme>
        <Year number="1">
            <Constraint>6 compulsory half-units,
                        2 optional half-units, no more than 1
                        optional half-unit can be non-programme.
            </Constraint>

            <Course value="Standard">
                <Name>Computer Architecture I</Name>
                <Code>1B10</Code>
                <Theme>Architecture</Theme>
                <Type requirement="C" level="F"/>
                <Dept>CS</Dept>
            </Course>
            ...
        </Year>
        ...
    </Curriculum>
</Curricula>
```

Figure 15: Curriculum fragment

introducing additional functions into the language, this rule cannot be expressed in the current version of cheXML.

## 10 Applications

xlinkit is a highly generic technology. It can be applied whereever you want to establish links between web resources, broadly construed, where those links reflect relationships between resource types. In particular rather than directly authoring and maintaining links xlinkit can provide semantically aware link generation.

Our principal interest derives from our software engineering background. Thus we have worked on applications largely in this area, most notably managing the consistency of complex development models produced by distributed teams. The UML study reported in the evaluation section is a case in point.

A large range of other applications primarily focusing on link generation and content management have been worked on by us or our partners. For example, information about important customers can be found in many places in sales files, service agreements, problem reports, logistics and supply records. xlinkit can be used to build a web-based customer relationship management system that allows you to navigate between all the pieces of information which reflect the interests of a single customer.

eCRM (e Customer Relationship Management) of this form is an example of a broad class of lightweight intranet portals. Many organisations have information in many different databases scattered across different sites. xlinkit can be used to build portals that can deliver coordinated access to this information and diagnose consistency problems.

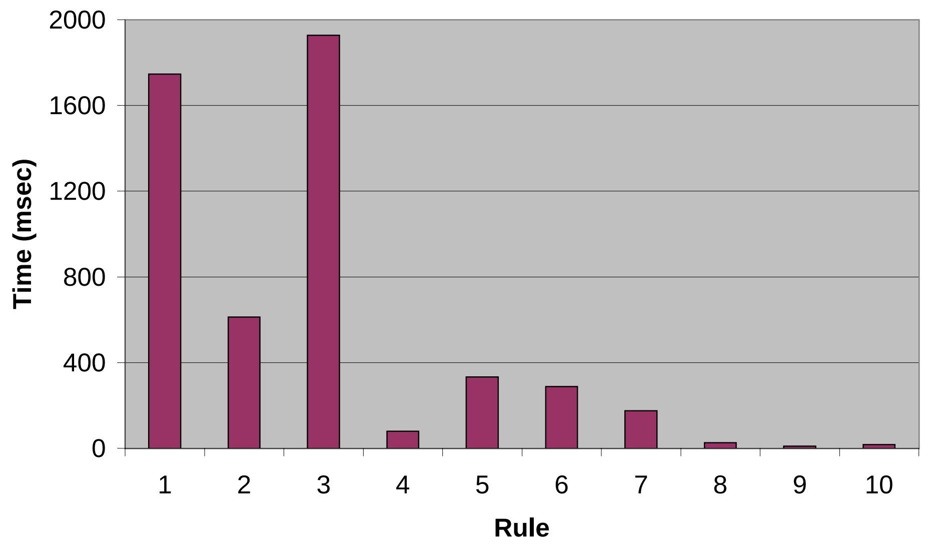


Figure 16: Syllabus study timings

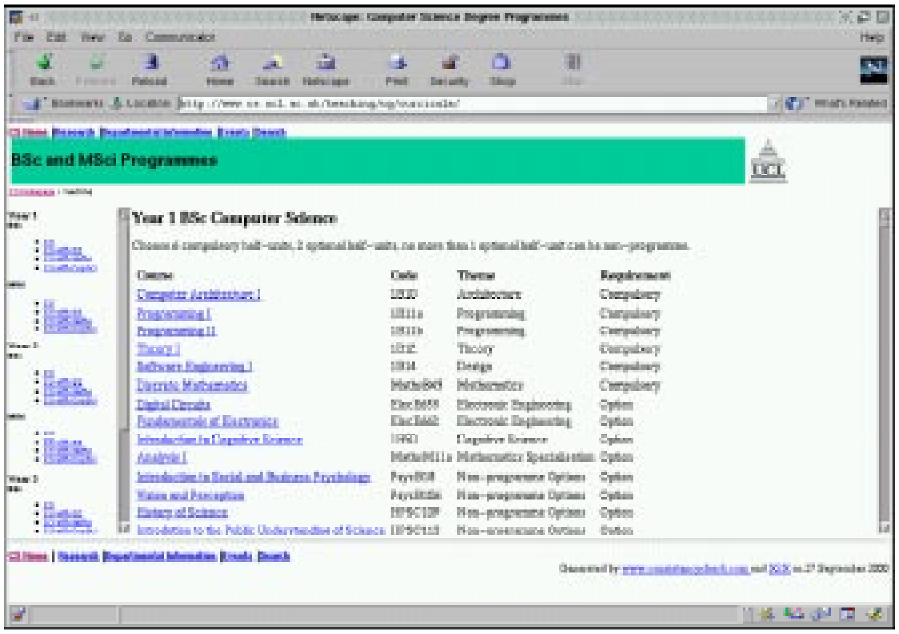
The idea of delivering web content on multiple channels such as web-TV, phones, PDAs etc. is now common. Unfortunately content has to be adapted for each channel to make a high value service. Content adaptation risks inconsistency with its attendant problems. xlinkit can be used to support navigation between information presented in different channels and identify problems. Web sites which aggregate content can use xlinkit to add value by providing content-relevant navigation without directly authoring links.

Other applications which have not been fully evaluated but appear promising are: consistency of information in service-level agreements, security policy and network management policy.

## 11 Related Work

This account of related work is not intended to be a survey of work on consistency management, for which you are referred to [3]. Below we highlight some key comparison points and work which has had a particular influence on xlinkit.

Consistency management has been recognised as an important issue by the programming language and software engineering communities. Early work in this area can be found in publications on programming environments such as the Cornell Synthesizer Generator [26], Gandalf [19] or Centaur [5]. These environments typically provide syntax-directed editors. When the user has finished entering a construct, incremental consistency checks related to the static programming language semantics being used are carried out. These semantic checks are typically carried out on a centralized data structure such as an abstract syntax tree. Later work on Software Development Environments (SDEs) such as IPSEN [22], Arcadia [29], ESF [28], ATMOSPHERE [4] and GOODSTEP [14] raised the complexity by integrating tools for different languages. The latter in particular allowed the specification of *semantic rules.* Checks for semantic integrity between documents could be triggered by user actions. Our approach represents a generalisation in that it



BSc and MSci Programmes

**Year 1 BSc Computer Science**

Choose 4 compulsory half-units, 2 optional half-units, no more than 1 optional half-unit can be non-programme.

| Course | Code | Theme | Requirement |
|---|---|---|---|
| Computer Architecture I | 1B10 | Architecture | Compulsory |
| Programming I | 1B11a | Programming | Compulsory |
| Programming II | 1B11b | Programming | Compulsory |
| Theory I | 1B12 | Theory | Compulsory |
| Software Engineering I | 1B14 | Design | Compulsory |
| Discrete Mathematics | MathB45 | Mathematics | Compulsory |
| Digital Circuits | ElecB655 | Electronic Engineering | Option |
| Fundamentals of Electronics | ElecB662 | Electronic Engineering | Option |
| Introduction to Cognitive Science | 1B60 | Cognitive Science | Option |
| Analysis I | MathM11a | Mathematics Specialisation | Option |
| Introduction to Social and Business Psychology | PsycB08 | Non-programme Options | Option |
| Vision and Perception | PsycB20a | Non-programme Options | Option |
| History of Science | HPSC12F | Non-programme Options | Option |
| Introduction to the Public Understanding of Science | HPSC113 | Non-programme Options | Option |

Generated by [illegible] and [illegible] on 27 September 2000

Figure 17: Automatically generated links in the curriculum

builds on the open model of XML rather than specific programming formalisms. In addition, we allow for the distribution of the documents and provide diagnostics in the form of links.

A viewpoint [18] allows developers to express a design fragment in some specification language, together with additional attributes describing the viewpoint. Multiple viewpoints can describe the same design fragment, leading to overlap and hence the possibility of inconsistency. The issues involved in inconsistency handling of multi-perspective specifications are outlined in [17]. Research in the viewpoints area also introduces the idea of *consistency rules* [12] between distributed specifications. The work on viewpoints has spun off our continuing interest in consistency management and in particular our tolerant view in which consistency is not always enforced. For a detailed discussion see [16]. Although a lot of theoretical work on viewpoints and the associated consistency checking scheme has been done, no generic implementation was ever provided. Our work realises these ideas by providing a concrete implementation on top of which a viewpoint framework can be built.

Graph grammars [33] can be used as a system-specification method. For example, recent work [30] has used graph grammars as a representation for UML class and sequence diagrams. For the approach to work, a translation of the participating specifications into graph grammars is necessary. The algorithm for matching graph morphisms, at least in its diagrammatic form, does not have the expressive power of our boolean filter formulae. More fundamentally, graph grammar systems make use of proprietary formalisms and centralised data structures, whereas we build on open standards and make distribution of participating documents our fundamental assumption.

The hypertext community has worked on the problem of automatic link generation. For a survey of this topic we refer you to [32]. Work in the area has traditionally focused on textual documents and many

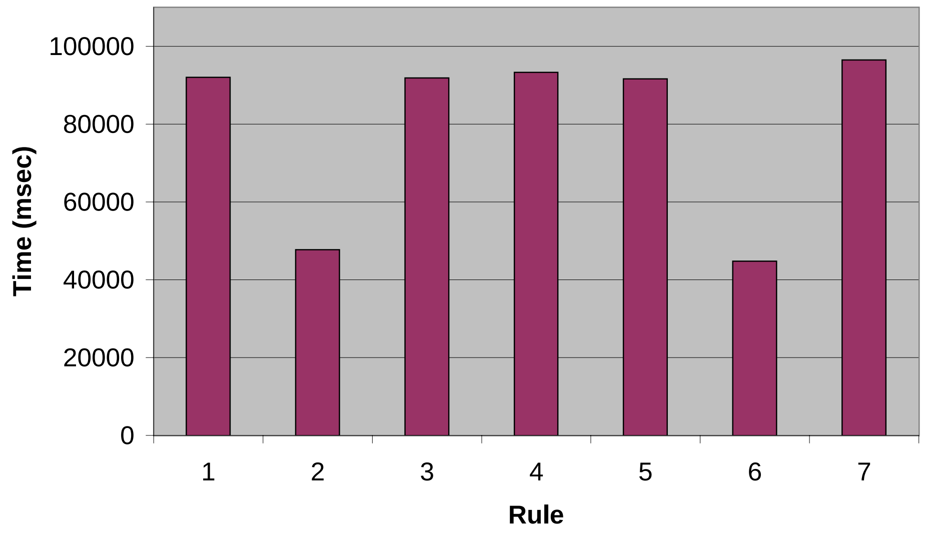


Figure 18: XMI study timings

approaches based on information retrieval techniques such as similarity measures can be found. We exploit the structure afforded by XML, and its widespread use for storing data rather than textual information in our approach to provide a much richer and more fine-grained expression of linking semantics.

There is a growing body of work concerned with applications of hypertext in software engineering. The CHIMERA project [1] demonstrates multiple document views and the capability of separating linking information from the underlying documents. It does not support consistency checking. CHIME [11] provides a framework for folding links into legacy software documents using information from software analysis tools. The work provides a strong case for the sort of browsing which our approach provides.

Our work has some analogies with Schematron [27], an XML structure validator which employs XSL and XPath to traverse documents and check constraints. Though simple and elegant, Schematron lacks the expressiveness of xlinkit and does not provide link generation or scalable support for distributed documents.

Finally, xlinkit builds on two previous prototype consistency checking schemes [13] which have substantially influenced the ideas on which it is based. In both cases these were standalone applications and used a rule language based on a restricted form of first-order logic. The language, architecture and content management framework are novel and the genericity, scaleability and performance of the xlinkit approach distinguish it from the earlier prototypes. For a comparison of runtime efficiency when checking rules on XMI files, please refer to Figure 19 in Appendix D.

## 12 Future Work

In this section we present aspects of our approach which need further research. Our work in this area has raised many problems, both technical and theoretical. Below we give what we think is a realistic agenda of issues to be tackled in the near future.

Our immediate goal will be to extend the expressiveness of our language to allow an arbitrary number of quantifiers. The biggest problem with this will be to find an adequate and understandable semantics for link generation – early investigation has however shown some promising results.

A “static” application service such as ours does more work than is really necessary because it has to recheck all documents against all rules upon request. When documents are changed, we would like to recheck only those rules that are affected by the changes. Such an *incremental checking* scheme is certainly a barrier we have to overcome if our approach is to scale to very large datasets. We do not think that this is a major problem and have a draft algorithm which implements this functionality.

Conflict resolution is a logical back-end of a consistency check and has not been discussed in this paper. It is assumed that the user will refer to our linkbases as a diagnostic tool and then take action in accordance with some real-world process. While we believe that conflict resolution can never be fully automated, it should still be possible to set certain default actions for handling trivial inconsistencies. Integration with a workflow management system may prove valuable in this respect and we will investigate this option. Achieving this goal without compromising the light-weight characteristics of xlinkit will however be a challenge.

The linkbases themselves are currently displayed rather statically, limiting their usefulness for diagnostic purposes. We are working on more interactive stylesheets which will use a multi-frame layout to allow a quick overview of which items are being linked. Prototypes for this can be found on our website.

## 13 Conclusion

This paper has described xlinkit, a lightweight application service that provides rule-based link generation and checks the consistency of distributed web resources. xlinkit leverages standard Internet technologies. It supports document distribution and can support multiple deployment models. It has a formal basis and evaluation has shown that it scales, both in terms of the size of documents and in the number of rules. We have identified some important applications and pointed to future directions for our work. xlinkit is the product of long-standing research looking at consistency management. It is available for use now and we are keen to see it applied in several areas.

## Acknowledgements

We would like to thank Giulio Carlone, who wrote the linkbase stylesheet, and our colleagues from earlier incarnations of this project, Ernst Ellmer, Andrea Zisman and Torbjoern Revheim, for their contributions. We would also like to thank the Apache Software Foundation and its volunteers for its continued and free provision of high-quality tools such as Xerces and Xalan, which have greatly simplified our work. The XLink working group also deserve thanks, in particular we are grateful to Eve Maler for technical feedback.

xlinkit is covered by PCT 9914232.5. Licenses for xlinkit can be obtained free for research and evaluation purposes. The technology and interactive examples can be found at `http://www.xlinkit.com`.

## References

[1] ANDERSON, K. M., TAYLOR, R. N., AND WHITEHEAD, E. J. Chimera: Hypertext for Heterogeneous Software Environments. In *Proc. of the European Conference on Hypermedia* (Edinburgh, UK, Sept. 1994).

[2] APPARAO, V., BYRNE, S., CHAMPION, M., ISAACS, S., JACOBS, I., HORS, A. L., NICOL, G., ROBIE, J., SUTOR, R., WILSON, C., AND WOOD, L. Document Object Model (DOM) Level 1 Specification. W3C Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium, Oct. 1998.

[3] B. NUSEIBEH AND S. EASTERBROOK AND A. RUSSO. Leveraging Inconsistency in Software Development. *IEEE Computer 33*, 4 (April 2000), 24–29.

[4] BOARDER, J., OBBINK, H., SCHMIDT, M., AND VÖLKER, A. Advanced techniques and methods of system production in a heterogeneous, extensible, and rigorous environment. In *Proc. of the 1st Int. Conf. on System Development Environments and Factories* (Berlin, Germany, 1989), N. Madhavji, W. Schäfer, and H. Weber, Eds., Pitman Publishing, pp. 199–206.

[5] BORRAS, P., CLÉMENT, D., DESPEYROUX, T., INCERPI, J., KAHN, G., LANG, B., AND PASCUAL, V. CENTAUR: The System. *ACM SIGSOFT Software Engineering Notes 13*, 5 (1988), 14–24. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, MA, USA.

[6] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., AND MALER, E. Extensible Markup Language. Recommendation http://www.w3.org/TR/2000/REC-xml-20001006, World Wide Web Consortium, Oct. 2000.

[7] CLARK, J. XSL Transformations (XSLT). Tech. Rep. http://www.w3.org/TR/xslt, World Wide Web Consortium, Nov. 1999.

[8] CLARK, J., AND DEROSE, S. XML Path Language (XPath) Version 1.0. Recommendation http://www.w3.org/TR/1999/REC-xpath-19991116, World Wide Web Consortium, Nov. 1999.

[9] CONSORTIUM, W. W. W. Amaya. http://www.w3.org/Amaya/, 2000.

[10] DEROSE, S., MALER, E., ORCHARD, D., AND TRAFFORD, B. XML Linking Language (XLink) Version 1.0. Candidate Recommendation http://www.w3.org/TR/2000/CR-xlink-20000703, World Wide Web Consortium, July 2000.

[11] DEVANBU, P., CHEN, Y.-F., GANSNER, E., MULLER, H., AND MARGIN, J. CHIME – Customizable Hyperlink Insertion and Maintenance Engine for Software Engineering Environments. In *Proc. of the* $21^{st}$ *Int. Conf. on Software Engineering* (Los Angeles, CA, USA, May 1999), ACM Press, pp. 473–482.

[12] EASTERBROOK, S., FINKELSTEIN, A., KRAMER, J., AND NUSEIBEH, B. Coordinating Distributed ViewPoints: The Anatomy of a Consistency Check. *Int. Journal of Concurrent Engineering: Research & Applications 2*, 3 (1994), 209–222.

[13] ELLMER, E., EMMERICH, W., FINKELSTEIN, A., SMOLKO, D., AND ZISMAN, A. Consistency Management of Distributed Documents using XML and Related Technologies. Research Note 99-94, University College London, Dept. of Computer Science, 1999. Submitted for Publication.

[14] EMMERICH, W. GTSL — An Object-Oriented Language for Specification of Syntax Directed Tools. In *Proc. of the 8th Int. Workshop on Software Specification and Design* (1996), IEEE Computer Society Press, pp. 26–35.

[15] EMPOLIS. X2X. http://www.empolis.co.uk, 2000.

[16] FINKELSTEIN, A. A Foolish Consistency: Technical Challenges in Consistency Management. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA)* (London, UK, September 2000), Springer, pp. 1–5.

[17] FINKELSTEIN, A., GABBAY, D., HUNTER, H., KRAMER, J., AND NUSEIBEH, B. Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering 20*, 8 (1994), 569–578.

[18] FINKELSTEIN, A., KRAMER, J., NUSEIBEH, B., FINKELSTEIN, L., AND GOEDICKE, M. Viewpoints: a framework for integrating multiple perspectives in system development. *Int. Journal of Software Engineering and Knowledge Engineering 2*, 1 (1992), 21–58.

[19] HABERMANN, A. N., AND NOTKIN, D. Gandalf: Software Development Environments. *IEEE Transactions on Software Engineering 12*, 12 (1986), 1117–1127.

[20] ISO 10744. Hypermedia/Time-based Structuring Language (HyTime). International standard, International Standards Organisation, 1997.

[21] Mozilla. http://www.mozilla.org, 2000.

[22] NAGL, M. Building Tighly Integrated Software Development Environments: The IPSEN Approach. *Lecture Notes in Computer Science 1170* (1996).

[23] OBJECT MANAGEMENT GROUP. *The Meta Object Facility*. Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, 1997.

[24] OBJECT MANAGEMENT GROUP. *Unified Modeling Language Specification*, March 2000.

[25] OBJECT MANAGEMENT GROUP. *XML Metadata Interchange (XMI) Specification 1.1*. 492 Old Connecticut Path, Framingham, MA 01701, USA, Nov. 2000.

[26] REPS, T. W., AND TEITELBAUM, T. The Synthesizer Generator. *ACM SIGSOFT Software Engineering Notes 9*, 3 (1984), 42–48. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, PA, USA.

[27] RICK JELLIFFE. The Schematron Assertion Language 1.5. Tech. rep., GeoTempo Inc., October 2000.

[28] SCHÄFER, W., AND WEBER, H. European Software Factory Plan – The ESF-Profile. In *Modern Software Engineering – Foundations and current perspectives*, P. A. Ng and R. T. Yeh, Eds. Van Nostrand Reinhold, NY, USA, 1989, ch. 22, pp. 613–637.

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

[30] Tsiolakis, A. Consistency Analysis of UML Class and Sequence Diagrams bas ed on Attributed Typed Graphs and their Transformation. Tech. Rep. 2000/3, Technical University of Berlin, March 2000. ISSN 1436-9915.

[31] Wadler, P. A formal semantics of patterns in XSLT. Markup Technologies, December 1999.

[32] Wilkison, R., and Smeaton, A. F. Automatic Link Generation. *ACM Computing Surveys 31*, 4es (December 1999). Article No. 27.

[33] Zündorf, A. *PROgrammierte GRaphErsetzungsSysteme – Spezifikation, Implementierung und Anwendung einer integrierten Entwicklungsumgebung*. PhD thesis, University of Aachen, 1996.

## A Wilbur’s Bike Shop sample files

*Product catalogue sample*

```
<Catalogue>

    <Product>
        <Name>HARO SHREDDER</Name>
        <Code>B001</Code>
        <Price currency="sterling">349.95</Price>
        <Description>Freestyle Bike.</Description>
    </Product>

    <Product>
        <Name>HARO TR2.1</Name>
        <Code>B002</Code>
        <Price currency="sterling">179.95</Price>
        <Description>BMX / Trail Bike.</Description>
    </Product>

</Catalogue>
```

*Sample advert file*

```
<Advert>

    <ProductName>HARO SHREDDER</ProductName>
    <Price currency="sterling">349.95</Price>
    <Description>Freestyle Bike. Super versatile frame for dirt,
        street, vert or flat. New full cromoly frame.
        Fusion MegaTube axle extenders.
    </Description>

</Advert>
```

*Sample service report file*

```
<ServiceReport>

    <CustomerIdentity reg_number="3645"/>

    <Report>
        <ProductName>HARO SHREDDER</ProductName>
        <ProductCode>B001</ProductCode>
        <ProblemDescr>Found a problem in ...</ProblemDescr>
    </Report>

</ServiceReport>
```

*Sample customer report file*

```
<CustomerReport>

    <CustomerIdentity>
        <FirstName>Licia</FirstName>
        <FamilyName>Capra</FamilyName>
        <Reg_Number>3645</Reg_Number>
    </CustomerIdentity>

    <Purchase>
        <ProductName>HARO SHREDDER</ProductName>
        <ProductCode>B001</ProductCode>
    </Purchase>

    <Purchase>
        <ProductName>Shimano LX Mountain Bike Crank Set</ProductName>
        <ProductCode>A102</ProductCode>
    </Purchase>

</CustomerReport>
```

# B Rule language semantics

## B.1 Definitions

The following definitions are due to Wadler [31]. They are included here for a quick overview. $Node$ is the basic datatype, referring to a DOM node. $Set(Node)$ denotes a set of nodes and $Set_1(Node)$ is the set containing exactly one node.

The following function is defined for the DOM:

$$value \quad : \quad Node \rightarrow String$$

The result of $value$ depends on the type of node being passed. The value of an element node is the value of its children, the value of a text node is its content, etc.

The abstract syntax of XPath patterns is defined as follows (the semantics will be omitted but should be fairly clear from the syntax for readers familiar with XPath).

n : Name
s : String
p : Pattern $\quad p ::= p_1|p_2 \mid /p \mid //p \mid p_1/p_2 \mid p_1//p_2 \mid p[q] \mid$
$\quad\quad n \mid * \mid @n \mid @* \mid$ `text()` $\mid$ `comment()` $\mid$ `pi(`$n$`)` $\mid$ `pi()` $\mid$
$\quad\quad$ `id(`$p$`)` $\mid$ `id(`$s$`)` $\mid$ `ancestor(`$p$`)` $\mid$ `ancestor-or-self(`$p$`)` $\mid$ `.` $\mid$ `..`

And the following semantic function is defined to assign meaning to patterns: $\mathcal{S}[\![p]\!]_x$ selects a set of nodes using pattern $p$ with $x$ as the context node. (We use $x$ as a subscript because it improves the readability of the following sections considerably).

$$\mathcal{S} \quad : \quad Pattern \rightarrow Node \rightarrow Set(Node)$$

## B.2 Rule language syntax

In the syntax, $n$, $p$ and $s$ are used as described in the previous section. For a concrete syntax, please refer to the DTD in Appendix E.

| | | | | | |
|---|---|---|---|---|---|
| $i$ | : | $Number$ | | | |
| $N$ | : | $Set(Node)$ | | | |
| $forall$ | : | $Forall$ | $forall$ | ::= | $N\ setop$ |
| $setop$ | : | $Setoperator$ | $setop$ | ::= | $setop$ `and` $setop$ \| $setop$ `or` $setop$ \| `sizeeq` $f\ f$ \| `sizenoteq` $f\ f$ |
| $f$ | : | $FilteredSet$ | $filter$ | ::= | $N\ e$ \| $i$ |
| $e$ | : | $Evaluator$ | $eval$ | ::= | $v = v$ \| $v \neq v$ \| `intersect` $v\ v\ i$ \| `subset` $v\ v\ i$ |
| | | | | | $e$ `and` $e$ \| $e$ `or` $e$ |
| $v$ | : | $Valueset$ | $v$ | ::= | `xpathsource` $p$ \| `xpathfilter` $p$ \| $i$ |

## B.3 Denotational semantics

This section defines the semantics of the abstract syntax specified above. The semantics will define how to check the consistency of a pair of documents. The main function in the semantic model will be $\mathcal{A}$, which defines the meaning of the $forall$ statement. The function will return a set of consistent nodes. It should be noted that the real implementation of $forall$ does not really return any nodes and this adjustment was made in order to make the semantics easier to express. $forall$ will use a $SetOperator$ to determine whether consistent of inconsistent links should be generated. The meaning of $SetOperator$ is defined by the function $\mathcal{O}$.

A set operator typically examines the size of a filtered set and returns $true$ or $false$ depending on whether a condition holds. The meanining of $FilteredSet$ will be defined using the $\mathcal{F}$ function. Filters themselves use an $Evaluator$ on each node to determine whether to discard it or include it in the filtered set. $\mathcal{E}$ is used to define the semantics of evaluators. Finally, in order to compare properties of source and destination nodes, xpath expressions relative to them have to be created. Those XPath expressions are grouped into $Valueset$s and $\mathcal{V}$ defines the semantics of the latter. The semantic functions are thus:

$$\begin{aligned}
\mathcal{A} &: Forall \rightarrow Set(Node) \\
\mathcal{O} &: SetOperator \rightarrow Node \rightarrow Boolean \\
\mathcal{F} &: FilteredSet \rightarrow Node \rightarrow Set(Node) \\
\mathcal{E} &: Evaluator \rightarrow Node \rightarrow Node \rightarrow Boolean \\
\mathcal{V} &: Valueset \rightarrow Node \rightarrow Node \rightarrow Set(String)
\end{aligned}$$

Before defining the semantic functions, helper-functions have to be defined. $setvalue$ takes as an input a set of DOM nodes and returns a set of strings corresponding to the value of the nodes (as defined in section B.1). (To be precise, it returns a multiset of strings).

$setequal$ returns true if two sets of strings have the same content, conversely for $setnotequal$. $setofsize$ takes an integer $i$ as a parameter and returns a set of cardinality $i$, filled with random strings. The functions are defined as follows:

$$\begin{array}{rcl}
setvalue & : & Set(Node) \rightarrow Set(String) \\
setvalue(N) & = & \{x \mid n \in N \wedge x = value(n)\} \\
\\
setequal & : & Set(String) \rightarrow Set(String) \rightarrow Boolean \\
setequal(X, Y) & = & |X| = |Y| \wedge \forall x \in X(\exists y \in Y \mid x = y) \\
\\
setofsize & : & Number \rightarrow Set(Node) \\
setofsize(i) & = & \{s \mid x = [1..i]\} \\
\\
intersection & : & Set(String) \rightarrow Set(String) \rightarrow Set(String) \\
intersection(X, Y) & = & X \cap Y
\end{array}$$

We can now proceed to implement the semantic functions introduced above. $\mathcal{A}$ evaluates a forall expression and returns a set of consistent nodes. It goes through all nodes $n$ in the set it is processing and checks if the child set operator evaluates to $true$ for $n$.

$$\begin{array}{lcl}
\mathcal{A} & : & Forall \rightarrow Set(Node) \\
\mathcal{A}[\![N setop]\!] & = & \{n \mid n \in N \ \wedge \ \mathcal{O}[\![setop]\!]_n\}
\end{array}$$

$\mathcal{O}$ has been passed the current source node $\sigma$ in its environment and needs to compute a set of node depending on the results its children return. The functions currently supported are checking if the size of two filtered sets matches and returning $true/false$ accordingly.

$$\begin{array}{lcl}
\mathcal{O} & : & SetOperator \rightarrow Node \rightarrow Boolean \\
\mathcal{O}[\![setop_1 \texttt{ and } setop_2]\!]_\sigma & = & \mathcal{O}[\![setop_1]\!]_\sigma \ \wedge \ \mathcal{O}[\![setop_2]\!]_\sigma \\
\mathcal{O}[\![setop_1 \texttt{ or } setop_2]\!]_\sigma & = & \mathcal{O}[\![setop_1]\!]_\sigma \ \vee \ \mathcal{O}[\![setop_2]\!]_\sigma \\
\mathcal{O}[\![\texttt{sizeeq } f_1 \ f_2]\!]_\sigma & = & |\mathcal{F}[\![f_1]\!]_\sigma| \ = \ |\mathcal{F}[\![f_2]\!]_\sigma| \\
\mathcal{O}[\![\texttt{sizenoteq } f_1 \ f_2]\!]_\sigma & = & |\mathcal{F}[\![f_1]\!]_\sigma| \ \neq \ |\mathcal{F}[\![f_2]\!]_\sigma|
\end{array}$$

$\mathcal{F}$ is supposed to return a filtered set depending on the value of the current source node $\sigma$. Such a set can be generated from an actual node set or be of constant size. In the former case, the filter will go through each node in the set to be filtered and use an evaluator to determine whether the node should be included. To that effect, the filter binds each node it is examining into the $\delta$ variable of the $Evaluator$ environment.

$$
\begin{array}{lcl}
\mathcal{F} & : & FilteredSet \rightarrow Node \rightarrow Set(Node) \\
\mathcal{F}[\![N\ e]\!]_\sigma & = & \{n \mid n \in N\ \wedge\ \mathcal{E}[\![e]\!]|_{\sigma,n}\} \\
\mathcal{F}[\![i]\!]_\sigma & = & setofsize(i)
\end{array}
$$

The function $\mathcal{E}$ has in its environment the current source node $\sigma$ and the current destination node $\delta$ and returns $true$ or $false$ depending on whether a certain property relative to the two nodes holds.

$$
\begin{array}{lcl}
\mathcal{E} & : & Evaluator \rightarrow Node \rightarrow Node \rightarrow Boolean \\
\mathcal{E}[\![e_1\ \texttt{and}\ e_2]\!]_{\sigma,\delta} & = & \mathcal{E}[\![e_1]\!]_{\sigma,\delta}\ \wedge\ \mathcal{E}[\![e_2]\!]_{\sigma,\delta} \\
\mathcal{E}[\![e_1\ \texttt{or}\ e_2]\!]_{\sigma,\delta} & = & \mathcal{E}[\![e_1]\!]_{\sigma,\delta}\ \vee\ \mathcal{E}[\![e_2]\!]_{\sigma,\delta} \\
\mathcal{E}[\![v_1 = v_2]\!]_{\sigma,\delta} & = & setequal(\mathcal{V}[\![v_1]\!]_{\sigma,\delta}, \mathcal{V}[\![v_2]\!]_{\sigma,\delta}) \\
\mathcal{E}[\![v_1 \neq v_2]\!]_{\sigma,\delta} & = & \neg setequal(\mathcal{V}[\![v_1]\!]_{\sigma,\delta}, \mathcal{V}[\![v_2]\!]_{\sigma,\delta}) \\
\mathcal{E}[\![\texttt{intersect}\ v_1\ v_2\ i]\!]_{\sigma,\delta} & = & |intersect(\mathcal{V}[\![v_1]\!]_{\sigma,\delta}, \mathcal{V}[\![v_2]\!]_{\sigma,\delta})|\ = i \\
\mathcal{E}[\![\texttt{subset}\ v_1\ v_2]\!]_{\sigma,\delta} & = & \mathcal{V}[\![v_1]\!]_{\sigma,\delta} \subseteq \mathcal{V}[\![v_2]\!]_{\sigma,\delta}
\end{array}
$$

$\mathcal{V}$ examines the current source node $\sigma$ and the current destination node $\delta$ and returns a valueset relative to either, depending on the syntax used. It can also return a set of constant size.

$$
\begin{array}{lcl}
\mathcal{V} & : & Valueset \rightarrow Node \rightarrow Node \rightarrow Set(String) \\
\mathcal{V}[\![\texttt{xpathsource}\ p]\!]_{\sigma,\delta} & = & \mathcal{S}[\![p]\!]_\sigma \\
\mathcal{V}[\![\texttt{xpathfilter}\ p]\!]_{\sigma,\delta} & = & \mathcal{S}[\![p]\!]_\delta \\
\mathcal{V}[\![i]\!]_{\sigma,\delta} & = & setvalue(setofsize(i))
\end{array}
$$

## C Case study rules

| | |
|---|---|
| 1 | Each course (of the CS department) must have a syllabus |
| 2 | The year of the course in the curriculum corresponds to the year in the syllabus |
| 3 | There must not be two courses with the same code |
| 4<br>5 | Each course listed as a pre-requisite in a syllabus must have a syllabus definition<br>A course cannot be a pre-requisite of itself |
| 6 | Each course in a studyplan is identified in the curricula |
| 7 | A student cannot take the same course twice |
| 8 | 1st year BSc/CS and MSci: 6 compulsory half-units |
| 9 | 1st year BSc/CS and MSci: 2 optional half-units |
| 10 | 1st year BSc/CS and MSci: no more than 1 optional half-units can be Non-programme |

Table 5: Curriculum study rules

| | |
|---|---|
| 1 | Classes with the same name in different class diagrams are considered to be identical |
| 2 | For every package P1 in a UML document UD1, there must exist an associated UML document D2 that refines P1 |
| 3 | A use case UC1 in a UML document UD1, and a document D2 containing an informal structured text describing the flow of events of a use case UC1 are considered to be related |
| 4 | For every use case UC1 that has a specialised use case UC2 in a UML document UD1, if there exist a sequence diagram SD1 specifying the main flow of UC2, then SD1 must include elements related to the main flow of use case UC1 |
| 5 | For every classifier C1 there must not be a classifier C2 that is both a subtype and supertype of C1 in any package diagram of the same UML model, for any level of nesting. |
| 6 | For every UML model, there must exist a package P1 |
| 7 | For every link L1 between two instances I1 and I2 in a collaboration, there must exist an association A1 or an aggregation AG1 between two classifiers C1 and C2, where the names of C1 and C2 equal the types of I1 and I2 respectively |

Table 6: XMI study rules

## D Case study results

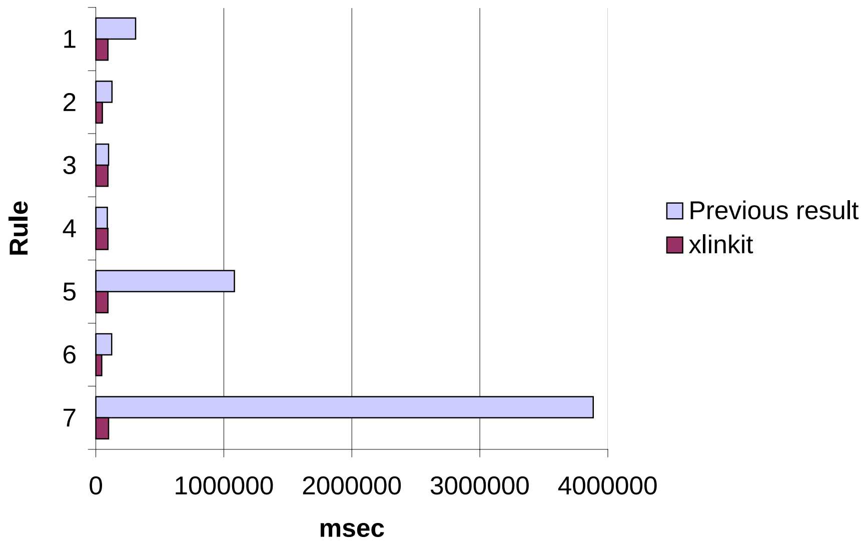


Figure 19: Comparison with previous results

## E Rule language XML DTD

```
<!ENTITY % Boolean "And,Or">

<!ELEMENT XPathSource EMPTY>
<!ATTLIST XPathSource
    value CDATA #REQUIRED>

<!ELEMENT XPathFilter EMPTY>
<!ATTLIST XPathFilter
    value CDATA #REQUIRED>

<!ELEMENT Constant EMPTY>
<!ATTLIST Constant
    value CDATA #REQUIRED>
```

```
<!ELEMENT Integer EMPTY>
<!ATTLIST Integer
    value CDATA "0">


<!-- ConsistencyRule -->

<!ELEMENT ConsistencyRuleSet (ConsistencyRule)*>

<!ELEMENT ConsistencyRule (Description,SetDefinition*,Forall)>
<!ATTLIST ConsistencyRule
id ID #REQUIRED>

<!ELEMENT Description (#PCDATA)>

<!ELEMENT SetDefinition (#PCDATA)>
<!ATTLIST SetDefinition
    id ID #REQUIRED>

<!-- Forall -->


<!ELEMENT Forall (AndOperator|OrOperator|SizeEqual|SizeNotEqual)>
<!ATTLIST Forall
    setid IDREF #REQUIRED
    mode (oneSet|multipleSet) "multipleSet">

<!ELEMENT AndOperator (SizeEqual*,SizeNotEqual*,AndOperator*,OrOperator*)>
<!ATTLIST AndOperator
    cmode (C|CX) "C"
    imode (I|IX) "">

<!ELEMENT OrOperator (SizeEqual*,SizeNotEqual*,AndOperator*,OrOperator*)>
<!ATTLIST OrOperator
    cmode (C|CX) "C"
    imode (I|IX) "">

<!ELEMENT SizeEqual (Filter*,Filtered*,Integer*)>
<!ATTLIST SizeEqual
    cmode (C|CX) "C"
    imode (I|IX) "">

<!ELEMENT SizeNotEqual (Filter*,Filtered*,Integer*)>
<!ATTLIST SizeNotEqual
    cmode (C|CX) "C"
    imode (I|IX) "">

<!-- Filters -->
```

```
<!ELEMENT Filter (%Boolean;,Equal,NotEqual,IsIntersect,Subset)>
<!ATTLIST Filter
    setid IDREF #REQUIRED
    mode (oneSet|multipleSet) "multipleSet">

<!ELEMENT Filtered (%Boolean;,Equal,NotEqual,IsIntersect,Subset)?>
<!ATTLIST Filtered
    setid IDREF #REQUIRED
    mode (oneSet|multipleSet) "multipleSet">

<!ELEMENT Equal (XPathSource,XPathFilter,Constant*)>
<!ELEMENT NotEqual (XPathSource,XPathFilter,Constant*)>
<!ELEMENT IsIntersect (XPathSource,XPathFilter,Constant*)>
<!ATTLIST IsIntersect size CDATA #REQUIRED>
<!ELEMENT Subset (XPathSource,XPathFilter,Constant*)>
<!ATTLIST Subset size CDATA #REQUIRED>

<!ELEMENT And (Equal*,NotEqual*,IsIntersect*,Subset*,And*,Or*,Not*)>
<!ELEMENT Or (Equal*,NotEqual*,IsIntersect*,Subset*,And*,Or*,Not*)>
<!ELEMENT Not (Equal|NotEqual|IsIntersect|Subset|And|Or|Not)>
```