Static Consistency Checking for Distributed Specifications

Christian Nentwich, Wolfgang Emmerich and Anthony Finkelstein
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{c.nentwich,w.emmerich,a.finkelstein}@cs.ucl.ac.uk

Abstract

Software engineers building a complex system make use of a number of informal and semi-formal notations. We describe a framework, xlinkit, for managing the consistency of development artifacts expressed in such notations. xlinkit supports distributed software engineering by providing a distribution-transparent language for expressing constraints between specifications. It specifies a semantics for those constraints that permits the generation of hyperlinks between inconsistent elements. We give a formal semantics for link generation, and show how we expressed the rules of the UML Foundation/Core modules in our language. We outline how we implemented xlinkit as a light-weight web service using open standard technology and present the results of an evaluation against several sizeable UML models provided by industrial partners.

1. Introduction

Mainstream software engineering makes use of informal to semi-formal notations for systems specifications, most prominently the UML. With systems growing in complexity and tight constraints on resources and expertise, distributed development often becomes a necessity.

The aim of this work is to support distributed software engineers in managing the consistency of their specifications. Contrary to previous work on constraint checking, we adopt a tolerant approach to inconsistency, shifting the focus from preventing to detecting and pinpointing inconsistent relationships.

We present a formal basis for checking the consistency and linking the elements of distributed, heterogeneous data in XML. We show how this can be applied to semi-formal software engineering content, most notably the UML. We outline a service, xlinkit, that has been developed to support this approach and show evaluation results which demonstrate its practicability. xlinkit leverages open standards such as XML, XLink and XPath in order to bridge heterogeneity problems and allow Internet scale distribution of development activities.

The key contributions of this paper are the adaptation of a simple first-order logic for expressing consistency relationships of elements in XML documents, the formal specification of a semantics that enables the generation of hyperlinks between inconsistent elements, and a light-weight architecture for delivering this as a service via the Web.

We show in particular how we made use of our language to express the constraints of the UML Foundation/Core package and to write rules that relate UML documents to Z specifications. We present an evaluation of the performance of our implementation against several industrial size models. xlinkit may be used at `http://www.xlinkit.com`.

2. Background

The eXtensible Markup Language (XML) [3] has gained acceptance in the business and software development world as an open standard and as a mechanism for bridging data heterogeneity problems.

XML has simplified the creation of domain-specific markup languages. Software engineering is an obvious application area where many languages have been developed. Most importantly for the examples in this paper, the XML Metadata Interchange (XMI) [24] standard supports the storage of MOF [22]-compliant models in XML format, in particular it includes a DTD for the UML as an example application.

XML is accompanied by a set of powerful technologies, which we will briefly review. XPath [6] supports the selection of sets of elements from XML documents by standardising a language for paths in trees. XLink [7] is the linking language for XML. An XLink consists of a set of locators which identify the resources connected by the link. XLink greatly improves the linking facilities available for hypertext authors over those available in HTML anchors: it can link more than two documents, links do not have to be inside the documents being linked (*out-of-line* links) and link traversal behaviour can be specified. When combined with a language like XPath, XLink can be used at a fine-grain level to relate elements rather than simply documents.

3. Related work

In this section we give pointers to immediately related work. It must be appreciated that the problem of managing consistency and of expressing constraints is a fundamental problem and hence appears in different guises in a wide variety of fields, which cannot all be compared in this paper. Instead, we identify those areas of research that have had the most influence on us and those that bear significant similarity.

There is a substantial body of work on programming language environments, for example the Cornell Synthesizer Generator [25] and Gandalf [17]. These systems typically check the consistency of abstract syntax trees by evaluating the static semantic rules of the underlying programming languages. What distinguishes our approach is that documents can be distributed arbitrarily and checked without depending on a central repository. The emphasis here is also on detecting and pinpointing inconsistency rather than enforcing consistency. The tolerant approach to inconsistency that we employ is related to that explored in the area of knowledge based software engineering in [2].

Later work on software development environments such as ESF [26], IPSEN [19], and GOODSTEP [10] integrates tools for different languages and provides consistency checks that span across different documents. Most of the work on these environments has also assumed the existence of a centralised repository, which limits scalability and often requires commitment to a single vendor. We build on open standards such as XML and web transport protocols to obtain access to distributed resources in order to avoid these problems.

Research on federated – or distributed – databases has addressed the problem of managing constraints between heterogeneous, distributed databases. One approach [5] to this problem is to execute queries on remote databases whenever values have changed in order to check and restore inter-database consistency. An immediate drawback of this approach is that constraints are localised, relating one database to other databases. It is thus not straightforward to express the global constraints that we provide. Various protocols [16] for constraint checking in federated databases have been explored. Most work in the area assumes the existence of database access logic on each host. In our scenario of documents on web servers, it cannot be assumed that the server will provide more than just support for streaming documents.

Again, most work in the area of federated databases differs from our approach in that it attempts to prevent inconsistency. Consequently, the focus is on preventing updates that introduce inconsistency rather than detecting and pinpointing it.

More recent work on constraints in semi-structured databases [1] has more immediate relevance to our work. Constraints on paths in semistructured databases are investigated in [4]. The paper proposes a restricted first order logic, similar to ours, for expressing constraints. While the authors claim that the restricted formulae are powerful enough to express a variety of constraints in a database context, the constraints necessary for software engineering, in particular the UML, exceed their power. Consequently, we allow more complex formulae, at the cost of making the implication problem for our formulae undecidable.

The Object Constraint Language [23] can be used to specify static constraints on UML models. OCL is more expressive than xlinkit, allowing the definition of functions and permitting the use of infinite sets such as the integers in constraints. On the other hand, it is much harder to find good diagnostics beyond *true* or *false* for the evaluation of such constraints. OCL also suffers the problem of being undecidable and being specific to the UML – it cannot be used to specify constraints between heterogeneous notations.

The problem of verifying constraints on websites is discussed in [11] and applied in [12]. It is important to distinguish between the goals of these approaches and our approach: Our constraints check if a set of data is consistent, whereas the approaches in the paper check if *any instance* of a schema graph will satisfy the constraint. Since we wish to tolerate inconsistency to introduce flexibility and also have no power to change the standard schemas used by software engineering notations, we cannot adopt this approach but instead focus on detecting inconsistencies in instance documents.

Recent work [27] on graph grammars [30] demonstrates their use in expressing UML class and sequence diagrams and specifying consistency relationships. In this approach specifications have to be translated into graph grammars. In addition, graph grammar systems typically make use of proprietary, centralised data structures, limiting their usefulness in a distributed setting.

The viewpoints framework [15] allows the specification of multiple views of the same system. Multiple viewpoints may describe the same design fragment, leading to the possibility of inconsistency [14]. Our work complements the body of theoretical work on viewpoints with a concrete implementation on top of which such a framework could be built.

Consistency management has become a research area of its own [21, 13]. Several approaches have been presented for dealing with inconsistency in formal models. Inconsistency in state diagrams is explored in [8]. The approach differs from ours in that it locks itself into a particular specification language (state diagrams) in order to allow more powerful constraints (temporal logic). Our focus is on providing support for the vast number of developers that use heterogeneous collections of informal or semi-formal specification languages.

Schematron [18] enables the specification of assertions about the structure of documents and uses XSLT to evaluate the assertions. Schematron is a widely used, lightweight approach to semantic document validation. It does however not posses the expressive power of our language since, by using pure XPath expressions, it essentially builds on a boolean logic. It also does not provide support for checking inter-document relationships and does not generate links.

There is a substantial amount of work in the area of hypertext on automatic link generation [29]. Most approaches in this area focus on textual data, which by its nature exposes a minimal amount of semantics. Several information retrieval techniques such as similarity measures have been applied to link generation. Our approach makes use of the semantics exposed by the structure of XML and the rules in order to provide linkbases that are guaranteed to be meaningful.

Finally, our language and its implementation are the successors of previous prototype schemes [9, 20] and have been considerably improved with respect to expressiveness, by the introduction of first order logic, performance and the removal of link generation annotation.

4. Motivation

The Unified Modeling Language (UML) [23] is widely used in software development. It combines a graphical notation with a semi-formal language. This language can be used together with the notation to provide multiple views of a system.

The UML has a syntax, and models expressed in it have to obey certain static semantic constraints. If any of these constraints are violated, by definition, inconsistency is introduced into the model. Unfortunately, as a result of distribution of development teams and process organisation, it is almost impossible for all constraints to be satisfied at any one time. For example, one constraint prescribes that if a class type is instantiated - e.g. in a sequence diagram - then that class type must have been defined - e.g. in a class diagram. If a developer decides to model some interactions first in order to get a better idea of how the final system could work, the developer may deliberately decide to leave the model in an inconsistent state.

Most modeling tools that implement the UML offer some sort of consistency checking mechanism. More often than not, these tools do not enforce the complete set of constraints in the UML standard [23]. Furthermore, UML models can be split using XMI as an interchange format and maintained in a distributed fashion, further complicating the issue.

Software development is often undertaken by distributed teams who use a variety of tools and specification languages. Interchange formats like XMI can help to alleviate problems of integration between tools that build on the same language, but offer little support otherwise. As an example, in a mission-critical system it may be necessary to introduce Z schemas corresponding to the elements in the UML model. If the Z specification lacks a schema for a certain UML element, the combined model is inconsistent with respect to this regime.

5. Rule definition

We now define our rule language, which is used to assert consistency relationships between document elements.

It may be beneficial, for paedagogic reasons, to start with an example of the kind of relationship we are trying to express. Constraint 1 for associations in the UML Foundation/Core package, *"The AssociationEnds must have a unique name within the association"* can be expressed abstractly as $\forall a(\forall x \in a(\forall y \in a(name(x) = name(y) \rightarrow x = y)))$, where a will be assigned to the associations and x and y refer to the association ends inside the association. This rule can now be evaluated against a set of documents in order to establish their consistency status with respect to the rule.

We use the XPath language to build up sets of elements. In the following, we use a notation for XPath queries due to Wadler [28]: $\mathcal{S}[\![p]\!]_x$ selects all nodes matching pattern p with x as the context node - the context node becomes the relative root for the selection. For example,

$$\mathcal{S}[\![schemadef]\!]_{/z}$$

selects all `schemadef` elements underneath the `z` element, which is also the root of the document.

Having selected the elements we are going to relate, we can formally specify constraints between them. Figure 1 shows the abstract syntax for our language - a restricted form of first order logic that uses XPath to select sets. In particular, the language has been restricted to make it decidable – although the implication problem for the language is still undecidable – and, as will be seen later, efficient to evaluate. It can be seen from the language definition that

- No functions are present
- Predicates are restricted to equality

$$\begin{array}{rcl}
rule & ::= & \forall \mathbf{var} \in \mathbf{xpath}(formula) \\
formula & ::= & \forall \mathbf{var} \in \mathbf{xpath}(formula) \mid \\
 & & \exists \mathbf{var} \in \mathbf{xpath}(formula) \mid \\
 & & formula\ \mathbf{and}\ formula \mid \\
 & & formula\ \mathbf{or}\ formula \mid \\
 & & formula\ \mathbf{implies}\ formula \mid \\
 & & \mathbf{not} formula \mid \\
 & & \mathbf{xpath} = \mathbf{xpath} \mid \\
 & & \mathbf{xpath} \neq \mathbf{xpath} \mid \\
 & & \mathbf{same\ var\ var}
\end{array}$$

Figure 1. Rule language abstract syntax

- The model used for evaluation must be finite (since XPath sets are always finite)

Since this language is unable to handle infinite sets, its power is restricted. It will not be possible – without user-defined predicates – to express the constraint "for all elements x, the children of x are prime numbers" since the latter half of the formula would involve the integers. Nevertheless, the language is still powerful enough, through the use of quantifiers, to permit the expression of powerful static semantic constraints.

As a simplified example of a formula in the language, we can express constraint 2 for classifiers in the UML standard, *"No Attributes may have the same name within a Classifier"*. Let C be the set of classifiers. As a simplification, we assume that all classifiers only have attributes as subelements. We can write $\forall c \in C(\forall a \in c(\forall b \in c(\mathcal{S}[\![name]\!]_a = \mathcal{S}[\![name]\!]_b \rightarrow same(a, b))))$, i.e. if two attributes have the same name as a subelement it follows that they are the same. This rule represents a triangular constraint that has to hold between a classifier and each pair of attributes contained within the classifier.

6. Rule checking

A particularly important contribution of our work is the generation of hyperlinks from rules as a meaningful diagnostic for the consistency status of a set of documents. We will explain this link generation strategy by presenting the formal semantics of our language, interspersed with some examples. This semantics is completely transparent to the user writing the rules, who only cares about the quality of the generated links and the straightforward syntax.

A formula in our rule language expresses a desirable or undesirable combination of elements types contained within the document set. When such a formula is applied to actual documents, elements will be found that either conform to it or violate it. Our strategy for highlighting the consistency status of a set of documents is to link together elements that satisfy rules with a *consistent link* and elements that violate rules with an *inconsistent link*.

A link consists of a set of *locators*. Each locator identifies the element it is pointing to. We now introduce some simplified formal notation for link representation: Let Σ be our alphabet and $S = \Sigma^\star$ be the set of strings over Σ. Since a path expression is just a string, and a locator essentially consists of a path expression, the set of strings is also the set of locators. We define the set of sets of locators as $Locators = \wp(S)$. The set of states a link can take is defined as $C = \{Consistent, Inconsistent\}$ and finally the set of consistency links is $L = C \times Locators$.

Before defining an evaluation strategy, we also need to introduce some auxiliary functions, shown in Figure 2: *flip* flips the consistency status of a link to its opposite. *linkcartesian* takes two links, x and y, and produces a new link with the status of x and a set of locators consisting of the union of the sets of locators from x and y. To preserve space, we introduce the infix operator × which takes a link and a set of links and produces a new set by applying *linkcartesian* between the single link and every individual link in the set. Finally, *bind* deals with variable bindings: a binding $B = S \times S$ maps a variable name to a string uniquely identifying a node. *bind* can be used to introduce new variable bindings into a set of bindings.

$$\begin{array}{rcl}
first(x, y) & = & x \\
second(x, y) & = & y \\
 & & \\
flip & : & L \rightarrow L \\
flip((Consistent, y)) & = & (Inconsistent, y) \\
flip((Inconsistent, y)) & = & (Consistent, y) \\
 & & \\
linkcartesian & : & L \rightarrow L \rightarrow L \\
linkcartesian(x, y) & = & (first(x), \\
 & & \ second(x) \cup second(y)) \\
 & & \\
\times & : & L \rightarrow \wp(L) \rightarrow \wp(L) \\
x \times Y & = & \{linkcartesian(x, y) \mid y \in Y\} \\
 & & \\
bind & : & B \rightarrow \wp(B) \rightarrow \wp(B) \\
bind(b, B) & = & \{b\} \cup B
\end{array}$$

Figure 2. Auxiliary functions

We will first define our evaluation function for the $rule$ non-terminal in Figure 1 and then progressively define the

$$
\begin{aligned}
\mathcal{F} &: formula \rightarrow boolean \\
\mathcal{F}[\![\forall \mathbf{var} \in \mathbf{xpath}(formula)]\!]_\beta &= \mathcal{F}[\![formula]\!]_{bind((\mathbf{var},x_1),\beta)} \wedge \ldots \wedge \mathcal{F}[\![formula]\!]_{bind((\mathbf{var},x_n),\beta)} \\
&\quad \mid x_i \in \mathcal{S}[\![\mathbf{xpath}]\!]_/ \\
\mathcal{F}[\![\exists \mathbf{var} \in \mathbf{xpath}(formula)]\!]_\beta &= \mathcal{F}[\![formula]\!]_{bind((\mathbf{var},x_1),\beta)} \vee \ldots \vee \mathcal{F}[\![formula]\!]_{bind((\mathbf{var},x_n),\beta)} \\
&\quad \mid x_i \in \mathcal{S}[\![\mathbf{xpath}]\!]_/ \\
\mathcal{F}[\![formula_1 \ \mathbf{and} \ formula_2]\!]_\beta &= \mathcal{F}[\![formula_1]\!]_\beta \wedge \mathcal{F}[\![formula_2]\!]_\beta \\
\mathcal{F}[\![formula_1 \ \mathbf{or} \ formula_2]\!]_\beta &= \mathcal{F}[\![formula_1]\!]_\beta \vee \mathcal{F}[\![formula_2]\!]_\beta \\
\mathcal{F}[\![formula_1 \ \mathbf{implies} \ formula_2]\!]_\beta &= \mathcal{F}[\![formula_1]\!]_\beta \rightarrow \mathcal{F}[\![formula_2]\!]_\beta \\
\mathcal{F}[\![\mathbf{not} \ formula]\!]_\beta &= \neg \mathcal{F}[\![formula]\!]_\beta \\
\mathcal{F}[\![\mathbf{xpath_1} = \mathbf{xpath_2}]\!]_\beta &= \mathcal{S}[\![\mathbf{xpath_1}]\!] = \mathcal{S}[\![\mathbf{xpath_2}]\!] \\
\mathcal{F}[\![\mathbf{xpath_1} \neq \mathbf{xpath_2}]\!]_\beta &= \mathcal{S}[\![\mathbf{xpath_1}]\!] \neq \mathcal{S}[\![\mathbf{xpath_2}]\!] \\
\mathcal{F}[\![\mathbf{same} \ \mathbf{var_1} \ \mathbf{var_2}]\!]_\beta &= \mathcal{S}[\![\mathbf{var_1}]\!] = \mathcal{S}[\![\mathbf{var_2}]\!]
\end{aligned}
$$

Figure 3. Rule language - truth value semantics

semantics of the various $formula$ productions. Our semantics will be supported by the standard first order logic truth evaluation semantics shown for completeness in Figure 3. We do not define a truth assignment for the top level $rule$ non-terminal since we are not really interested in the overall truth of the formula - we are interested in link generation.

Figure 4 shows the complete link generation semantics for our language. The function $\mathcal{R} : rule \rightarrow \wp(L)$ takes a rule and returns a set of consistency links. Since a rule consists of a universal quantifier, the function will build a set of nodes using a path expression, assign the node in the set to the quantifier variable in turn and ask the subformula to return a set of links. Depending on the truth value of the subformula for the current assignment, the function generates a consistent or inconsistent link by prepending its current variable assignment to all links returned by the subformula.

The quantifiers in the $formula$ productions behave similarly. Both the universal and existential quantifiers will first evaluate their XPath expression - which may now include references to variables bound to some node in a parent formula - and then bind each node in the resulting node set to their variable in turn, calling the subformula evaluation. As far as link generation is concerned, the existential quantifier generates consistent links if the subformula is true for the current assignment, prepending its own current node to the links returned by the subformula. The universal quantifier generates an inconsistent link every time a subformula is false, again prepending its current node to the links returned by the subformula.

We can demonstrate the behaviour of the quantifiers using an example. Suppose we have a formula of the form $\forall x \in X(\exists y \in Y(x = y))$. Suppose also that the sets X and Y consist of the elements shown in Table 1. We use the notation X_i to address the ith element in set X. The rule evaluation will bind $'a'$ to x and call the existential quantifier's evaluation function. Stepping through the destination set, the equality comparison returns false for the first entry, so the entry is ignored. On the second entry, it returns true. The existential quantifier generates a new link of the form $(Consistent, \{Y_2\})$. For the third entry, the subformula returns false so the link generated previously represents the whole set of links returned. The universal quantifier is now notified that the subformula has come out true for the current assignment. It thus prepends its current node X_1 to all links returned by the subformula. The set of consistency links is now $\{(Consistent, \{X_1, Y_2\})$.

X	Y
‘a’	‘c’
‘b’	‘a’
‘c’	‘f’

Table 1. Sample sets for rule evaluation

For node X_2, the existential quantifier will not find any assignment that makes its subformula true. As a consequence, its truth value will be $false$ and it will return an empty set of links. The universal quantifier will obtain this truth value and hence generate a new set of links - prepending its current assignment to the empty set of links returned by the existential quantifier - $\{(Inconsistent, \{X_2\})\}$. Evaluation of the third node will proceed similarly to that of the first node. The result is the union of all sets of links obtained by the universal quantifier:

$$
\begin{gathered}
\{(Consistent, \{X_1, Y_2\}), \\
(Inconsistent, \{X_2\}), \\
(Consistent, \{X_3, Y_1\})\}
\end{gathered}
$$

Intuitively, these links make sense. X_1 and Y_2 form a desirable relationship with respect to this rule and thus have

$$
\begin{array}{lcl}
status & : & bool \rightarrow C \\
status\ \top & = & Consistent \\
status\ \bot & = & Inonsistent \\
 & & \\
\mathcal{R} & : & rule \rightarrow \wp(L) \\
\mathcal{R}\llbracket \forall \mathbf{var} \in \mathbf{xpath}(formula) \rrbracket & = & \{(status(\mathcal{F}\llbracket formula \rrbracket_{bind((\mathbf{var},x),\{\})}), \{x\}) \times \mathcal{L}\llbracket formula \rrbracket_{bind((\mathbf{var},x),\{\})} \\
 & & \ \mid x \in \mathcal{S}\llbracket \mathbf{xpath} \rrbracket_{/} \} \\
 & & \\
\mathcal{L} & : & formula \rightarrow \wp(L) \\
\mathcal{L}\llbracket \forall \mathbf{var} \in \mathbf{xpath}(formula) \rrbracket_{\beta} & = & \{(Inconsistent, \{x\}) \times \mathcal{L}\llbracket formula \rrbracket_{bind((\mathbf{var},x),\beta)} \\
 & & \ \mid x \in \mathcal{S}\llbracket \mathbf{xpath} \rrbracket_{/} \wedge \mathcal{F}\llbracket formula \rrbracket_{bind((\mathbf{var},x),\beta)} = \bot \} \\
\mathcal{L}\llbracket \exists \mathbf{var} \in \mathbf{xpath}(formula) \rrbracket_{\beta} & = & \{(Consistent, \{x\}) \times \mathcal{L}\llbracket formula \rrbracket_{bind((\mathbf{var},x),\beta)} \\
 & & \ \mid x \in \mathcal{S}\llbracket \mathbf{xpath} \rrbracket_{/} \wedge \mathcal{F}\llbracket formula \rrbracket_{bind((\mathbf{var},x),\beta)} = \top \} \\
\mathcal{L}\llbracket formula_1 \ \mathbf{and}\ formula_2 \rrbracket_{\beta} & = & \{x \times \mathcal{L}\llbracket formula_2 \rrbracket_{\beta} \mid x \in \mathcal{L}\llbracket formula_1 \rrbracket_{\beta}\} \\
\mathcal{L}\llbracket formula_1 \ \mathbf{or}\ formula_2 \rrbracket_{\beta} & = & \mathcal{L}\llbracket formula_1 \rrbracket_{\beta} \cup \mathcal{L}\llbracket formula_2 \rrbracket_{\beta},\ if\ \mathcal{F}\llbracket formula_1 \rrbracket_{\beta} = \mathcal{F}\llbracket formula_2 \rrbracket_{\beta} \\
 & & \mathcal{L}\llbracket formula_1 \rrbracket_{\beta},\ if\ \mathcal{F}\llbracket formula_1 \rrbracket_{\beta} = \top \\
 & & \mathcal{L}\llbracket formula_2 \rrbracket_{\beta},\ if\ \mathcal{F}\llbracket formula_2 \rrbracket_{\beta} = \top \\
\mathcal{L}\llbracket formula_1 \ \mathbf{implies}\ formula_2 \rrbracket_{\beta} & = & \mathcal{L}\llbracket formula_2 \rrbracket_{\beta},\ if\ \mathcal{F}\llbracket formula_1 \rrbracket_{\beta} = \top \wedge \mathcal{F}\llbracket formula_2 \rrbracket_{\beta} = \top \\
 & & \{x \times \mathcal{L}\llbracket formula_2 \rrbracket_{\beta} \mid x \in \mathcal{L}\llbracket formula_1 \rrbracket_{\beta}\}, \\
 & & \qquad if\ \mathcal{F}\llbracket formula_1 \rrbracket_{\beta} = \top \wedge \mathcal{F}\llbracket formula_2 \rrbracket_{\beta} = \bot \\
 & & \{flip(x) \mid x \in \mathcal{L}\llbracket formula_1 \rrbracket_{\beta}\},\ otherwise \\
\mathcal{L}\llbracket \mathbf{not}\ formula \rrbracket_{\beta} & = & \{flip(x) \mid x \in \mathcal{L}\llbracket formula \rrbracket_{\beta}\} \\
\mathcal{L}\llbracket \mathbf{xpath_1} = \mathbf{xpath_2} \rrbracket_{\beta} & = & \{\} \\
\mathcal{L}\llbracket \mathbf{xpath_1} \neq \mathbf{xpath_2} \rrbracket_{\beta} & = & \{\} \\
\mathcal{L}\llbracket \mathbf{same}\ \mathbf{xpath_1}\ \mathbf{xpath_2} \rrbracket_{\beta} & = & \{\}
\end{array}
$$

Figure 4. Rule language - link generation semantics

been linked using a consistent link. For X_2, we could not find a matching element and have thus created an inconsistent link. X_2 is inconsistent with the whole of the system rather than a particular element, so it stands alone.

Productions which contain only terminals, such as the definition of **equals** do not introduce new variables nor contain subformulae. Their linking semantics thus is to always return an empty set. We are left with the task of defining the behaviour of the logical connectives.

Because of space limitations we will concentrate on the **and** operator. Suppose we have a formula of the form $\forall x \in X(\exists y \in Y(x = y) \wedge \exists z \in Y(x \neq z))$ and are given the two sets X and Y – note that we are not referring to the sets in Table 1 anymore. Assume that x is currently bound to X_1 and we evaluate the existential quantifiers. Assume furthermore that for this binding the first existential quantifier returns the set of links

$$\{(Consistent, \{Y_1\}), (Consistent, \{Y_2\})\}$$

and the second existential quantifier returns

$$\{(Consistent, \{Y_3\}), (Consistent, \{Y_4\})\}$$

Intuitively, we would like our links to express that the current assignment of x is consistent with respect to both subformulae of the **and** operator at the same time. We achieve this by computing a 'cartesian product' between the links produced by the subformulae: for each link in the first set of links and for each link in the second set of links, we generate a new link containing the union of locators of both links. The result returned by the **and** connective in the example is thus the set:

$$\begin{array}{c}\{(Consistent, \{Y_1, Y_3\}), (Consistent, \{Y_1, Y_4\}) \\ (Consistent, \{Y_2, Y_3\}), (Consistent, \{Y_2, Y_4\})\}\end{array}$$

This set will be passed back to the universal quantifier, which will generate the final set of links (for the current assignment of x):

$$\begin{array}{c}\{(Consistent, \{X_1, Y_1, Y_3\}), \\ (Consistent, \{X_1, Y_1, Y_4\}) \\ (Consistent, \{X_1, Y_2, Y_3\}) \\ (Consistent, \{X_1, Y_2, Y_4\})\}\end{array}$$

The semantics for the remaining connectives were similarly derived to yield meaningful links. The whole semantics was implemented in Haskell and tested against simulation data structures to evaluate its usability.

We conclude the section with some observation about the complexity of our link generation semantics. First of all we note that the evaluation function will always terminate since the quantifiers which introduce looping into the scheme only execute their loops n times for a node set of size n. Secondly, the run-time complexity of the system is mainly influenced by the maximum nesting of quantifiers, i.e. it is $O(n^k)$ where k is the maximum level of quantifier nesting. Though this exponential behaviour sounds problematic, it is not a problem in practice. Most rules from the UML Core, which represent a complex scenario by our standards, require at most 3 levels of nesting. In addition, empirical results show that the evaluation is fast enough for the theoretical complexity to be ineffectual.

7. XML deployment

Our language depends on the ability to select sets of elements for inclusion into checks and on an output format for the resulting links. In practice, we employ XPath to achieve the former and XLink for the latter. This makes it possible to provide a consistency checking environment which is based entirely on open and extensible standards.

We provide an encoding of our entire rule language in XML. Using XML for writing rules allows users to use the same tool environment for editing that they would be using for processing their documents in XML format. In addition, since elements of the rules can be selected using XPath, it will be possible to write meta-rules that check other rules should the need arise. Figure 5 shows an example rule which states that we have to create a Z schema somewhere for each class in our UML model. The syntax of the XMI paths in the figure has been abbreviated for clarity.

```
<globalset id="$classes"
    xpath="//Foundation.Core.Class[@xmi.id]"/>
<globalset id="$stateschemas"
    xpath="/z/schemadef[@purpose='state']"/>

<consistencyrule id="r1">
  <description>
    Every class in the UML model must have a
    state schema in a Z specification
  </description>

  <forall var="c" in="$classes">
    <exists var="s" in="$stateschemas">
     <equal op1="$c/ModelElement.name/text()"
          op2="normalize-space($s/text()[1])"/>
    </exists>
  </forall>
</consistencyrule>
```

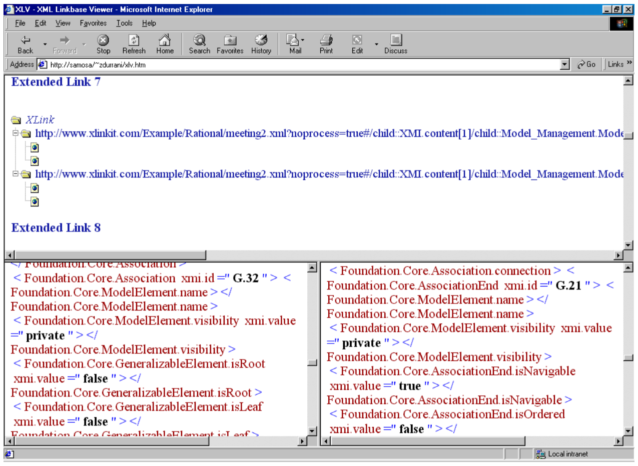
Figure 5. Example rule in XML

The rule will be parsed and checked by our implementation. The resulting set of links will be stored in an XLink *linkbase*, an example of which can be seen in Figure 6 - the XMI paths have been abbreviated again. This file now contains the complete consistency status of the participating documents with respect to the set of rules that were checked.

As they stand, the linkbases are not directly usable. We have developed several options for making the results of the consistency more accessible. The out-of-line links in the linkbases can be *folded* back into the files they are pointing to. The processor makes local copies of the files before putting in the additional required elements. For example, the link shown in Figure 6 would cause one link to be inserted into a copy of the UML model, pointing to the Z schema, and one link to be created into a copy of the Z schema, pointing to the UML model.

```
<xlinkit:LinkBase
  xmlns:xlinkit="http://www.xlinkit.com"
  docSet="file://DocumentSet.xml"
  ruleSet="file://RuleSet.xml">

  <xlinkit:ConsistencyLink
    ruleid="zrules.xml#/id('r1')">

    <xlinkit:State>consistent</xlinkit:State>
    <xlinkit:Locator
      xlink:href="meeting2.xml#//Class[1]"/>
    <xlinkit:Locator
      xlink:href="meetingz.xml#/z/schemadef[2]"/>

  </xlinkit:ConsistencyLink>

</xlinkit:LinkBase>
```

Figure 6. Sample result linkbase



http://samosa/~zdurrani/xlv.htm

Extended Link 7

XLink

http://www.xlinkit.com/Example/Rational/meeting2.xml?noprocess=true#/child::XMI.content[1]/child::Model_Management.Mode

http://www.xlinkit.com/Example/Rational/meeting2.xml?noprocess=true#/child::XMI.content[1]/child::Model_Management.Mode

Extended Link 8

```
</ Foundation.Core.Association >
 < Foundation.Core.Association  xmi.id =" G.32 " > <
Foundation.Core.ModelElement.name > </
Foundation.Core.ModelElement.name >
 < Foundation.Core.ModelElement.visibility  xmi.value
=" private " > </
Foundation.Core.ModelElement.visibility >
 < Foundation.Core.GeneralizableElement.isRoot
 xmi.value =" false " > </
Foundation.Core.GeneralizableElement.isRoot >
 < Foundation.Core.GeneralizableElement.isLeaf
 xmi.value =" false " > </
Foundation.Core.GeneralizableElement.isLeaf >
```

```
 < Foundation.Core.Association.connection > <
Foundation.Core.AssociationEnd  xmi.id =" G.21 " > <
Foundation.Core.ModelElement.name > </
Foundation.Core.ModelElement.name >
 < Foundation.Core.ModelElement.visibility  xmi.value
=" private " > </
Foundation.Core.ModelElement.visibility >
 < Foundation.Core.AssociationEnd.isNavigable
 xmi.value =" true " > </
Foundation.Core.AssociationEnd.isNavigable >
 < Foundation.Core.AssociationEnd.isOrdered
 xmi.value =" false " > </
```

Figure 7. Interactive linkbase servlet

Alternatively, we can take the XML representation of the linkbase and make the linkbase itself more user-friendly by transforming it into HTML and making it interactive. Figure 7 shows a screenshot of a servlet we have developed for this purpose. The servlet shows the links contained in the linkbase in the top frame. Users can click on a pair of locators in the link. The linked XML documents are then rendered into HTML and the linked elements are framed and centered in the respective bottom frame. The user can then easily compare the inconsistent elements and decide whether and how to take action.

8. Document Management

We provide a general framework for managing the submission of documents and rules to our check engine. It is infeasible to check every document against every rule and it is certainly not necessary to check every document every time. The aim of this framework is to provide a structured mechanism to select which rules are supposed to be applied to which documents.

```
<DocumentsSet name="UMLandZ">
  <Description>
    A couple of UML models and Z schemas
  </Description>

  <Document href="catalogue.xml"/>

  <Set href="Zschemas.xml"/>
</DocumentsSet>
```

Figure 8. Sample document set

Figure 8 shows a *document set*. It includes a `Document` element which directly adds a document into the set and a `Set` element which includes a further document set. The latter can be used to form a hierarchy of document sets, perhaps representing a hierarchy of subsystems. At run-time, the hierarchy is flattened and all documents are loaded.

```
<RuleSet name="ZIFRule">

  <Description>XMI vs. ZIF rules</Description>

  <RuleFile href="zifrules.xml"
            xpath="//consistencyrule[1]"/>

</RuleSet>
```

Figure 9. Sample rule set

Rules are treated similarly, they are stored in rule sets. Figure 9 shows a sample rule set. There is a `RuleFile` element for including a rule file directly and an `xpath` attribute for specifying precisely which rules to extract from that file. Although it is not shown in the figure, a `Set` element can again be used to include further rule sets. The rule set mechanism can be used to select different rulesets depending on the developer's area of interest and stages in the lifecycle.

9. Architecture

We have implemented our technology as an openly accessible web service, Figure 10 shows its basic architecture. Users assemble their files in a document set and their rules in a rule set. They then type the URL of their sets into a browser form and submit it to the web server, which invokes a servlet.

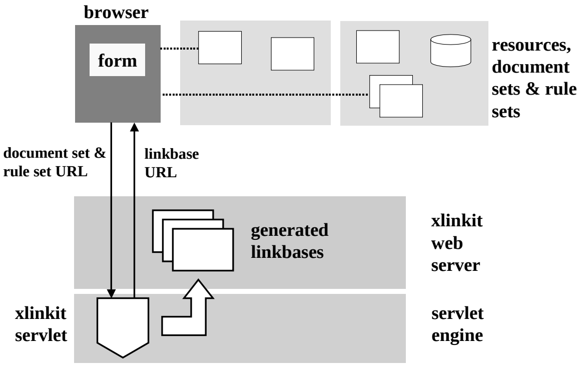


Figure 10. Web service architecture

The check engine executes the consistency rules and computes a set of links. These links are then stored locally in a linkbase with a unique identifier and a result page is generated which contains instructions for downloading or viewing the linkbase.

10. Evaluation

We have evaluated our implementation in several case studies. This section presents the results we obtained by checking the UML Core constraints against a set of sample models: a small design model of a meeting scheduler, a medium size model taken from Rational Rose and 19 industrial size models provided by an investment bank.

We use the number of ModelElement objects contained in each model as a measure of scale since almost everything in the UML meta-model derives from ModelElement. Our small model contains 93 elements, the medium size model has 610 elements. The number of elements contained in the industrial models ranges from 64 to 2834 elements. In terms of file size, the models range from around 100 kilobytes to 6 megabytes.

The rules that were checked are rules from the UML Foundation/Core package. We have expressed all but eight rules. Of those rules, some are enforced by the XMI DTD and do not have to be checked, some require transitive closure, which we have implemented in the past [20] but not yet ported to our new implementation, and some cannot be checked because the required information is not exported by XMI.

All results listed below were obtained by executing our check engine on a single-processor Intel machine running at 750 Mhz and using the IBM JDK 1.2 for Linux. We will first discuss the results obtained by checking the UML Core constraints in each file individually. Figure 11 shows the total time in seconds taken by each rule for all files. The longest time taken to check all rules against any individual file was 2.38 minutes, for the largest file in the set. The most amount of RAM consumed was 60 megabytes, again for the largest file in the set.

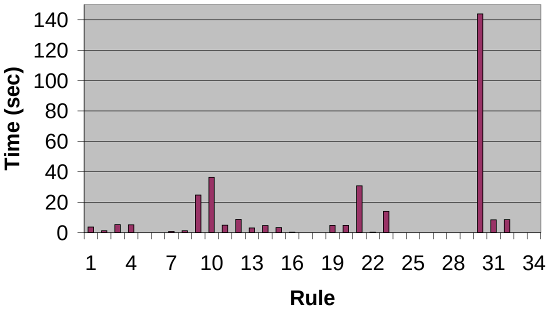


Figure 11. Rule totals for UML Core rules

We can observe several interesting properties from the figure. There is a large variance in the time taken by the different rules. This is due to two factors: Some rules apply to more files than others, for example almost every model has classes whereas few have association classes, i.e. the association class rules do not apply in many cases. Secondly, the complexity of the XPath expressions in the rules varies greatly. Some expressions use straightforward tree paths whereas others require sophisticated functions like id lookup. This is a feature of the rather complex design of XMI. XPath selection is the single most expensive process in rule checking and hence the complexity of the path has the greatest impact - far greater than the complexity of the formula in terms of nested quantifiers!

In total, over all files, 8101 inconsistent links were generated. Consistent link generation was turned off since we were only interested in finding inconsistencies. Although the number seems large given that the models were exported from a case tool it can be explained. Some of the models included in the check were analysis or high-level design models, so they were incomplete with respect to definition of fundamental data types, had operation parameter types missing and similar problems.

11. Conclusion and Future work

We have presented the formal foundations of xlinkit, a generic technology for consistency checking and link generation, its deployment as an open, light-weight web service using XML, and its application to software engineering.

Our evaluation has demonstrated that we were able to express the constraints of the UML Foundation/Core package as consistency rules and check those rules against sizeable UML models in a reasonable amount of time. Nevertheless, there are several problems that we could not address in this paper.

Once inconsistencies have been detected, the problem of resolution arises. Several questions have to be answered when dealing with inconsistency, for example: Which rules are more important than others? Similarly, which inconsistencies are more severe than others? Our ruleset mechanism may help to overcome these problems by allowing developers to select different rule sets depending on their preferences. We are also considering process integration and automated techniques for dealing with inconsistencies.

Our current approach is static, meaning that it checks all documents against all rules in the rule set when a check is invoked. We have prototyped an incremental algorithm which performs a tree-diff operation between documents and computes a set of rules which need to be rechecked. We plan to integrate this algorithm into our web-based architecture and evaluate how it improves performance.

Our memory management strategy at the moment is to load all documents from a document set into memory as DOM trees in order to make them available for checking. This approach will not scale for the very large volumes of data typical in e-commerce applications, though less common in software engineering. We will investigate options such as ordering our rules in order to minimise the amount of trees required in memory. Alternatively, it may be possible to exploit the caching mechanisms of XML databases, which can provide DOM trees directly without parsing, to circumvent the problem.

12. Availability

We encourage experimentation and use of our work, which has already been taken up in business and financial applications, and by several research groups. An OSI certified open source implementation [1] is available, as is a free web service with interactive examples, including those referred to in this paper.

13. Acknowledgements

We would like to thank our students who produced the linkbase processor and the interactive linkbase viewer. We also gratefully acknowledge financial support from Zuhlke Engineering UK Ltd. for Christian Nentwich.

[1] xlinkit is protected by PCT 9914232.5

References

[1] S. Abiteboul. Querying Semi-Structured Data. In *Proceedings of the 5th International Conference on Database Theory*, pages 1–18, 1997.

[2] R. Balzer. Tolerating Inconsistency. In *Proceedings of the 13th International Conference on Software Engineering*, pages 158–165, Austin, TX USA, May 1991. IEEE Computer Society Press.

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language. Recommendation http://www.w3.org/TR/2000/REC-xml-20001006, World Wide Web Consortium, Oct. 2000.

[4] P. Buneman, W. Fan, and S. Weinstein. Path Constraints in Semistructured Databases. *Journal of Computer and System Sciences*, 61(2):146–193, 2000.

[5] S. Ceri and J. Widom. Managing Semantic Heterogeneity with Production Rules and Persistent Queues. In *Proceedings of the 19th VLDB Conference*, pages 108–119, Dublin, Ireland, 1993.

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

[7] S. DeRose, 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.

[8] S. Easterbrook and M. Chechik. A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, pages 411–420, Toronto, Canada, May 2001. ACM Press.

[9] E. Ellmer, W. Emmerich, A. Finkelstein, D. Smolko, and A. Zisman. Consistency Management of Distributed Documents using XML and Related Technologies. Research Note 99-94, University College London, Dept. of Computer Science, 1999.

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

[11] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying Integrity Constraints on Web Sites. In *Proceedings of the 16th International Joint Conference on Articial Intelligence*, pages 614–619, 1999.

[12] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Declarative Specification of Web Sites with Strudel. *VLDB Journal*, 9(1):38–55, 2000.

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

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

[15] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. *Int. Journal of Software Engineering and Knowledge Engineering*, 2(1):21–58, 1992.

[16] P. Grefen and J. Widom. Protocols for Integrity Constraint Checking in Federated Databases. *Distributed and Parallel Databases*, 5(4):327–355, 1997.

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

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

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

[20] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a Consistency Checking and Smart Link Generation Service. Research Note RN/00/66, University College London, Dept. of Computer Science, 2000. Submitted for Publication.

[21] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging Inconsistency in Software Development. *IEEE Computer*, 33(4):24–29, April 2000.

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

[23] Object Management Group. *Unified Modeling Language Specification*, March 2000.

[24] Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA. *XML Metadata Interchange (XMI) Specification 1.1*, Nov. 2000.

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

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

[27] A. Tsiolakis. Consistency Analysis of UML Class and Sequence Diagrams based on Attributed Typed Graphs and their Transformation. Technical Report 2000/3, Technical University of Berlin, March 2000. ISSN 1436-9915.

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

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

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