Consistency Checking of Financial Derivatives Transactions

Daniel Dui and Christian Nentwich
Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK
{d.dui,c.nentwich}@cs.ucl.ac.uk

Wolfgang Emmerich
Zuhlke Engineering Ltd
49 Great Cumberland Place
London, W1H 7TH UK
wje@zuhlke.com
http://www.zuhlke.com

Bryan Thal
UBS Warburg
1 Finsbury Avenue
London EC2M 2PP,UK
bryan.thal@ubsw.com

Abstract. Financial institutions are increasingly using XML as a de-facto standard to represent and exchange information about their products and services. Their aim is to process transactions quickly, cost-effectively, and with minimal human intervention. Due to the nature of the financial industry, inconsistencies inevitably appear throughout the lifetime of a financial transaction and their resolution introduces cost and time overheads.

We give an overview of requirements for inconsistency detection in our particular domain of interest: the Over-The-Counter (OTC) financial derivatives sector. We propose a taxonomy for the different classes of consistency constraints that occur in this domain and present how xlinkit, a generic technology for managing the consistency of distributed documents, can be used to specify consistency constraints and detect transaction inconsistencies. We present the result of an evaluation where xlinkit has been used to specify the evaluation rules for version 1.0 of the Financial Products Markup Language (FpML). The results of that evaluation were so encouraging that they have led the FpML Steering Committee to consider xlinkit as the standard for specifying validation constraints throughout.

1 Introduction

Financial institutions offer their clients a wide range of financial products. Increasingly data about these products are exchanged electronically in ordering, trading and confirmation processes between financial institutions, stock exchanges and institutional clients.

The financial products that are traded on stock exchanges tend to be standard products, such as equity, currency or fixed income products. In addition to these standard security products there are secondary products, whose value is derived from such securities. Examples for such products are futures, swaps, interest rate and energy derivatives. These derivative contracts are often very specialized and not traded in large volumes. Derivatives are therefore mostly traded between institutions without the involvement of an exchange in so called over-the-counter transactions.

To address cost pressures and increased timeliness requirements in trading derivative contracts, financial institutions increasingly rely on electronic trading. To facilitate data interchange between different institutions the International Swaps and Derivatives Association (ISDA) has standardized an XML markup language called Financial Products Markup Language (FpML), which supports many types of derivative products. While successful in standardizing the concrete syntax the FpML consortium has acknowledged that to support effective electronic trading of derivatives it is necessary to also be able to validate constraints within one FpML trade document and between a trade and other FpML documents, which are beyond those that can be expressed using XML Document Type Definitions (DTDs) and Schemas [18].

The novel contribution of this paper is a systematic classification of the different classes of constraints that need to be checked when electronic derivative data is exchanged. We describe how these constraint classes can be expressed using xlinkit, an XML-based first-order rule language [15] that we have developed over the last three years. While doing so, we elaborate on how the different xlinkit concepts, such as plug-in operators, are used in practice. We have used xlinkit to describe all validation constraints for FpML 1.0. Xlinkit rules are executed by an efficient rule engine and we show that validations of FpML 1.0 trades can be performed in about 980 milliseconds, which is fast enough to be able to embed checks in a straight-through-processing architecture.

In the next section we summarize the results of our requirements elicitation exercise, which highlights the different kinds of consistency constraints that financial institutions are interested in. In Section 3, we classify these different classes of constraints based on the primitives that are needed for expressing the required correctness criteria. In Section 4, we briefly present the background of xlinkit to make the paper self-contained. The aim of Section 5 is to show how the different constraint classes are supported by xlinkit in order to give evidence that xlinkit can support all required checks. In Section 6, we report on the experience we made when expressing validation rules for FpML 1.0 and show the performance figures for FpML validation before discussing related work in Section 7 and concluding the paper in Section 8.

2 Validation Requirements

During the initial stages of our research, we have investigated the requirements that financial institutions have for checking electronic trade representations. We discuss these requirements ordered by sources where inconsistencies may occur. Fig. 1 shows an overview of the different sources that we have identified. Alongside the discussion of the validation requirements in each of these areas we endeavour to present rationales that will make the importance of automated validation evident.

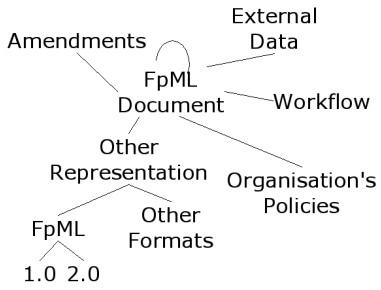

Fig 1. Classification of Consistency Constraints

2.1 Internal Inconsistencies

First, inconsistencies may arise due to internal faults within a trade document. These inconsistencies might occur because the fact that an FpML document is valid with respect to a DTD or XML Schema definition does not necessarily imply that it is meaningful and suitable for being processed by middle- and back-office tools. There are a large number of consistency constraints that cannot be expressed in a DTD or XML Schema, for example:

Example **1:** An FpML trade identifies a number of payment dates when cash flows are exchanged between the parties involved. The list of these payment dates needs to be in chronological order, they need to be subsequent to the date when the trade has been struck, and they need to be valid calendar dates.

Thus there is a need for us to be able to express validation rules for individual trade documents, that go beyond the syntactic correctness that can be established by an XML parser. In many ways this is similar to the static semantic rules that programming language designers introduce to a programming language, because the context free grammar is just not suitable to express the rules that a program must obey so that it has a meaning.

It is important for financial institutions to detect such internal inconsistencies during the exchange of electronic trading data. The later inconsistencies are detected the more effort would have been wasted processing the trade and the more costly the settlement of the trade will become.

2.2 Amendments

Trade amendments are common and are a prime source of inconsistencies. An amendment modifies the terms of an existing trade; therefore the trade document representing the original trade is replaced with a new document that represents the new version of the trade. The new document, which by itself is valid, is not necessarily an acceptable replacement for the first one.

Example **2:** The way of identifying securities in Germany is about to be changed from Wertpapierkennummern to the International Security Identification Number system (ISIN) defined by ISO [9]. As a result trades that have not been completed by the change due date will have to be amended.

It is important to note that not all parts of a trade document can be changed during the lifetime of a transaction. In particular banks would be reluctant (and require higher approval) if the economics of a trade were changed. We therefore have to check that amendments are only made to parts of the trade document that do not affect the economics.

2.3 Reference Data

By "reference data" we mean any information that is outside the control of the organisations involved in the transaction. Reference data therefore subsumes identifications of security products issued by exchanges or market data issued by, for example Reuters or Bloomberg.

Example **3:** An option to sell German Telecom shares identifies the underlying equity share by referring to the international security identification number. Consistency with reference data in this context means that that number has to exist and identify a security instrument that is compatible with the option.

The rationale for demanding consistency of trading documents to reference data is to prevent meaningless trades to be passed on to settlement.

2.4 Workflow

A workflow consists of a collection of activities that support a specific business process: in our case, the execution of a financial transaction. Banks have workflow definitions in place that are, in part, demanded by the banking authorities. Workflow constraints ensure that a trade has a state that is consistent with its prescribed workflow at any point during the trade's life-cycle.

Example **4:** A swap contract specifies that Party A shall transfer a certain sum to Party B's account each month. At the time of making each new payment, Party A wants to check that all due payments up to that point have occurred; if not, an inconsistency should be flagged.

The rationale for workflow constraints in general is that organizations want to ensure that the actions on the trading floor match the workflows prescribed by the institution's compliance department.

2.5 Organisational Policies

Organisational policies are decided by the organisation itself and do not depend on anything from the outside world. They lead to constraints that the organisation imposes on itself and that are meaningful only within the organisation.

Example **5:** A trading desk deals exclusively with standard interest rate products with a face value not greater than $1,000,000 and with clients from EU countries. All trades booked on that desk will be flagged as not consistent if they do not meet the constraints, for example if one the parties is a client from Japan.

Usually these policies define constraints that regard the identity and the geographical location of the parties involved in the trade, the identity of the employee booking or processing the trade, the type and the cost of the products.

2.6 Semantic Equivalence Across Representations

Different organisations inevitably represent the same information in different document formats. Often this also happens for different systems within a single organisation. This class of constraints ensures that the data in one document is consistent with the data in another document in another format.

Example **6:** A bank communicates with its clients using FpML, but internally uses another format that captures the same information in the FpML document plus other data about the state of the transaction. We want to be able to define constraints to ensure that the document in the Bank's internal format represents the data in the FpML document accurately.

Example **7:** An FpML document specifies that Party A shall pay $100,000 to Party B every three months. As the trade is executed, every three months Party A will produce another (XML) document that instructs to transfer $100,000 from Party A's account to Party B's account. The data in the FpML document should be consistent with the data in the cash-transfer documents.

In this discussion we are assuming that all file formats are XML-based. In the finance industry, this is acceptable because most organisations already deploy XML internally or are currently in the process of doing so. Other formats are generally unsophisticated and easily convertible to XML. Also interchange formats, which evolve more slowly, are converging to XML. This is the case for the FIX and SWIFT, currently the most widely used.

The Financial Information eXchange (FIX) protocol is a messaging standard developed for the exchange of securities transactions. A FIX message simply consists of an ASCII string that lists an unstructured series of tag-value pairs. FIX 4.3, the current version, defines a dual syntax consisting of the traditional "tag=value" notation and an XML-based equivalent: FIXML.

The Society for Worldwide Interbank and Financial Telecommunication (SWIFT) provides an industry standard messaging service for inter-bank communication since 1977. The messages (known as "FIN" messages) consist of a

simple ASCII string, much like a FIX message, and are delivered by means of a store-and-forward protocol over an X.25 connection. The next generation of the service, SwiftNet, will deliver XML messages over a secure IP network and will become available from the end of 2002. The message format is known as swiftML.

Any standard, no matter how well received, is destined to change over time. This is particularly so in the derivative market where traders invent new and more complicated financial products on a weekly basis. In the case of FpML, version 1.0 has been available since May 2001, a trial recommendation of version 2.0 is currently in use and a first working draft of version 3.0 has been published in January 2002. It is therefore inevitable that organisations will need to handle documents that conform to different versions of the standard specifications. In any case, parties to a derivative trade are often interested in evidence that, although expressed in a different markup language, the trade documents represent the same trade.

3 Constraint Classification

In this section, we classify the consistency constraints that occur in financial transactions. The aim of this classification is to provide a basis for the assessment of the appropriateness of the xlinkit rule language in this setting.

When we classified the constraints a question arose as to where to draw the boundary between constraints that can be expressed in XML Schemas and constraints that one would want to express using xlinkit. Firstly, the above requirements imply a large number of constraints that simply cannot be expressed in a schema (e.g. those that imply more than one document or that are not related to a simple attribute or element).

Moreover one might choose to express constraints in xlinkit even though they could be expressed in an XML schema. For derivative products, ISDA agrees the schema based on consensus among its member organizations. This means that only constraints will be reflected in the schema for which there is wide-spread agreement. To express organization-specific constraints, such as those discussed in Section 2.5 it might be appropriate not to change the schema but to express those constraints in a separate representation.

We now discuss the different classes of constraints. We exemplify these constraints with examples taken from the FpML 1.0 validation rules that we created in order to evaluate our approach.

3.1 Simple Constraints

Existence: Existence constraints impose that a certain element must be present in a trade document. A DTD or schema can enforce this simplest case, but it cannot enforce existence of one element conditional upon the existence of another element.

Example **8:** A trade document should specify reset dates only if it describes a floating rate swap.

Equality: The constraint specifies that that two elements must have the same value.

Example **9:** When comparing two trade documents we want to check that the trade settlement date is the same in both documents.

Uniqueness: Uniqueness constraints impose that, in a set of values, each value should appear exactly once.

Example **10:** An FpML trade document lists a number of business center codes that identify the locations on which date calculations are based. For a document to be meaningful, each business center code must appear in the list only once.

Comparison: These involve the usual $<$, $>$, $\leq$, and $\geq$ operators. In our case it is more common for the operands to be dates and not integers or real numbers.

Example **11:** In a trade contract the *effective date* is the date when the deal is booked, and usually when payments begin, the terminationDate when the contract expires and payments end. For the trade document to be meaningful the *effective date* must be earlier than the *termination date*.

3.2 Complex Constraints

In practice most rules are combinations of any of the above. We found examples where simple constraints have to be combined using logical operators (AND, OR, NOT), checks whether an element is included in a set of other elements and checks whether sets are equal. We now give examples for each of these.

Example **12:** For most dates in a trade document, it is necessary to specify a "roll convention", i.e. a date is to be adjusted when it is not a valid business day. This can happen in a number of ways: it can be moved to the following business day, the following Monday, etc.

If the rollConvention is not specified ('NONE') or is the type used on the Sydney Futures Exchange ('SFE'), then the payment period must be expressed in months or years rather than weeks or fortnights.

Example **13:** An FpML trade document defines groups of business center codes that identify the locations on which date calculations are based. Instead of repeating them many times in the same documents, it is possible to reference them. Clearly, for the trade document to be valid, each reference must match with at least one group of business centers.

Example **14:** For two corresponding businessCenters in document A and document B: each businessCenter element in document A shall have the same value of a businessCenter element in document B and each businessCenter element

in document B shall have the same value of a businessCenter element in document A.

3.3 Exotic Constraints

These constraints cannot be meaningfully expressed with the operators sketched above and require the development of new operators. In our domain, date comparison operations tend to be particularly common.

Example **15:** In order to define how interest rate calculation should occur in a swap, an FpML trade document specifies a *calculation period frequency*, the frequency at which calculation of interest rates should occur, and a *reset frequency*. For the document to be meaningful the calculation period frequency should be an integer multiple of the reset frequency.

Thus any constraint language has to be extensible to support domain-specific operators.

4 xlinkit

xlinkit is a framework for expressing and checking the consistency of distributed, heterogeneous documents. It comprises a language, based on a restricted form of first order logic, for expressing constraints between elements and attributes in XML documents. The restriction enforces that sets have a finite cardinality which is not a problem in our application domain as XML documents only have a finite set of elements and attributes. xlinkit also contains a document management mechanism and an engine that can check the documents against the constraints. A full description of xlinkit, including a formal specification of its semantics and its scalability is beyond the scope of this paper and can be found in [15].

xlinkit has been implemented as a lightweight mechanism on top of XML and creates hyperlinks to support diagnostic by linking inconsistent elements. Because it was built on XML, xlinkit is flexible and can be deployed in a variety of architectures. It has also been applied in a variety of different application areas, including the validation of Software Engineering documents such as the design models and source code of Enterprise JavaBeans-based systems [16].

Sets used in quantifiers of xlinkit rules are defined using XPath. XPath [3] is one of the foundational languages in the set of XML specifications. It permits the selection of elements from an XML document by specifying a tree path in the document. For example, the path `/FpML/trade` would select all `trade` elements contained in the `FpML` element, which is the root element.

XLink [4] is the XML linking language and is intended as a standard way of including hyperlinks in XML documents. XLink goes beyond the facilities provided by HTML by allowing any XML element to become a link; by specifying that links may connect more than two elements, so called *extended* links; and by allowing links to be managed *out-of-bound*, as collections of links termed *linkbases*. These features allow us to capture complex relationships between a

multitude of elements that are involved in an inconsistency without altering any of the inconsistent documents.

The linkbases generated by xlinkit form an ideal intermediate representation from which we can derive different forms of higher level diagnoses. Firstly, we have developed a report generator that takes report templates and uses the linkbase to obtain details of the elements involved in an inconsistency to provide a report similar to an error report that a compiler generates. Secondly, we have developed a servlet that can read a linkbase and allows users to select a link and it will then open the documents referenced in the link, navigate to elements identified in the link and in that way assist users to understand the links. We have also developed a linkbase processor that folds links back into the documents so that both consistent and inconsistent data can be captured as hyperlinks. It depends on the application domain which of these higher-level diagnoses mechanisms is most appropriate. For the domain discussed in this paper we found the report generation to have generated most interest among our partners in various investment banks.

5 Constraint Specification with xlinkit

In this section we show how each of the examples of consistency constraints stated in English in Section 3 can be expressed using first order logic expression and therefore xlinkit rules.

5.1 Existence

Example 8 leads to Constraint 1, which can be expressed in the xlinkit rule language as shown below.

Constraint **1:** A `resetDates` must exist if and only if there exists also a `floatingRateCalculation` in `calculation`.

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

In this xlinkit rule, `x` refers to all swapStream nodes, `y` refers to the resetDates that are children of swapStream nodes and `x` floatingRateCalculation nodes that are contained in calculation elements, which are contained in calculationPeriodAmount elements of swapStream nodes.

5.2 Equality

Example 9 leads to Constraint 2:

Constraint **2:** The `tradeDate` element in the document A must have the same value of the `tradeDate` element in the document B.

This is translated into an xlinkit rule that compares two pairs of texts that are contained in a tradeDate element and demands that they are the same. Note that the rule itself does not make any assumption on where the trades are located or, indeed, how many trades there are. This is achieved using a document description that is given to the xlinkit rule engine. In this particular example, the document set would contain two elements, with the URLs of document A and B

```
<forall var="x" in /FpML/tradeDate/text()>
  <forall var="y" in /FpML/tradeDate/text()>
    <same op1="$x" op2="$y" />
  </forall>
</forall>
```

5.3 Uniqueness

Example 10 leads to Constraint 3:

Constraint **3:** The values of the `businessCenter` elements in `businessCenters` must be unique.

```
<forall var="a" in="//businessCenters">
  <forall var="x" in="$a/businessCenter">
    <forall var="y" in="$a/businessCenter">
      <implies>
        <not>
          <same op1="$x" op2="$y" />
        </not>
        <notequal op1="$x/text()" op2="$y/text()" />
      </implies>
    </forall>
  </forall>
</forall>
```

This constraint says that for each pair of businessCenter elements that are contained in a businessCenters element somewhere in an FpML trade that if the two nodes are not identical, the text elements that they contain cannot be identical either.

5.4 Comparison

Example 11 leads to Constraint 4:

Constraint **4:** The value of `effectiveDate` must be smaller than the value of `terminationDate`.

```
<forall var="x" in="//calculationPeriodDates">
  <forall var="y" in="$x/terminationDate">
    <forall var="z" in="$x/effectiveDate">
      <greater_than>
```

```
        op1="$x/terminationDate/unadjustedDate/text()"
        op2="$x/effectiveDate/unadjustedDate/text()"
      </greater_than>
    </forall>
  </forall>
</forall>
```

This constraint demands that for each pair of terminationDate and effectiveDate elements that are contained in a calculationPeriodDates element somewhere in an FpML trade, that the effectiveDate has to be greater than the terminationDate.

5.5 Logical Operators

Example 12 leads to Constraint 5

Constraint **5:** If `rollConvention` is not either 'NONE' or 'SFE', then `period` must be either 'M' or 'Y'.

```
<forall var="x" in="//calculationPeriodFrequency">
  <implies>
    <or>
      <notequal op1="$x/rollConvention/text()" op2="'NONE'" />
      <notequal op1="$x/rollConvention/text()" op2="'SFE'" />
    </or>
    <or>
      <equal op1="$x/period/text()" op2="'M'" />
      <equal op1="$x/period/text()" op2="'Y'" />
    </or>
  </implies>
</forall>
```

The above rule is the first example of a complex rule, where logical operators implies and or are used to specify the constraint.

5.6 Compare One Element with a Set

Example 13 leads to Constraint 6

Constraint **6:** In `businessCentersReference`: the value of attribute `href` shall be equal to the value of attribute `id` of exactly one `businessCenters` element.

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

This rule demands that for every businessCentersReference element of an FpML trade there exists a businessCenters element such that the href attribute matches the id attribute.

5.7 Date Modulo (operator)

Example 15 leads to Constraint 7:

Constraint **7:** The `calculationPeriodFrequency` must be an integer multiple of the `resetFrequency`.

This constraint demands an unusual check that we address using an xlinkit plug-in operator. We define operator *is_period_multiple*, which takes four parameters: two (period, unit) pairs. For example (2, 'Y', 6, 'M') meaning two years and six months. If the second period is contained an exact number of times in the first one the operator will return true, otherwise it will return false. The operator is made available to xlinkit via the following declaration, again in an XML-based format.

```
<OperatorSet impl="fpmlOperators.es">
<OperatorDefinition name="is_period_multiple">
  <param name="periodA" type="node"/>
  <param name="unitA" type="node"/>
  <param name="periodB" type="node"/>
  <param name="unitB" type="node"/>
</OperatorDefinition>
```

The implementation is an ECMA (Standardized Java)Script function contained in the file `fpmlOperators.es`.

```
<forall var="x" in="//swapStream">
  <forall var="y" in="$x/paymentDates/paymentFrequency">
    <forall var="z" in="$x/calculationPeriodDates/calculationPeriodFrequency">
      <operator name="fpml:is_period_multiple">
        <param name="periodA" value="$y/periodMultiplier/text()" />
        <param name="unitA" value="$y/period/text()" />
        <param name="periodB" value="$z/periodMultiplier/text()" />
        <param name="unitB" value="$z/period/text()" />
      </operator>
    </forall>
  </forall>
</forall>
```

The rule then checks that for each pair of paymentFrequency and calculationPeriodFrequency contained in a swapStream element that the invocation to the is_period_multiple operator returns true.

5.8 Summary

We note that we have been able to formulate examples for each of the different classes of constraints discussed in Section 3. The primitives built into the xlinkit rule language were sufficient to express quite a large number of constraints and we are able to express exotic constraints using plug-in operators.

6 Validation

Above we have shown that analytically xlinkit can express the different classes of constraints that we have identified earlier. In order to also provide empirical evidence that xlinkit is expressive enough to define all classes of consistency constraints that occur in practice in a derivative trading setting, we have elicited a set of 35 consistency rules from product departments in UBS Warburg. We have then formalized these rules and evaluated the performance of the xlinkit rule engine when checking them. The rules can be found at [5]. We report the results in this section.

6.1 On the Benefits of Formalization

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

> BusinessCentersReference must reference a businessCenter element within the document.

We translated that into the following xlinkit rule:

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

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

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

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

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

6.2 Performance

After having formalized the initial set of 35 constraints a total of 28 constraints remained (because some were subsumed) and others were proven to be obsolete. We then checked a 17KByte FpML 1.0 trade document that did not have any constraint violations using the xlinkit rule engine. The rule engine executed on a dual-processor Pentium III with a clock speed of 1.7 GHz and 1GByte of RAM. We show the rule execution times for the different rules in the performance graph in Figure 2.

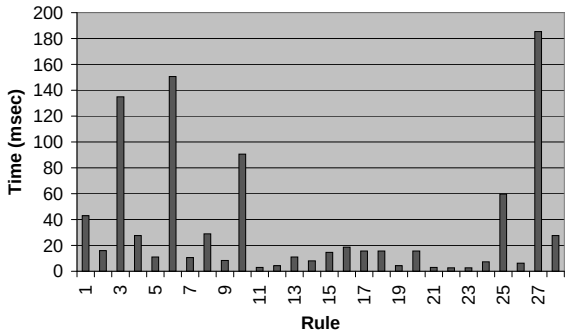

Fig 2. xlinkit performance

Figure 2 shows that some rule require a significantly longer time than others to evaluate. This happens when the evaluation of an XPath expression requires the traversal of the entire DOM tree. For example expression "//businessCentersReference" appears in rule 27.

Expressions of this kind are necessary when the element we are trying to find can appear anywhere in the document, unless we can explicitly identify the position of the element in the document tree. In rule 1 we changed the expression from "//swapStream" to "/FpML/trade/product/swap/swapStream" and the execution time reduced by five folds (the figure in the graph). The only drawback is that long XPath expressions make rules less readable.

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

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

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

7 Related Work

There is a large body of work on validation of constraints of context-free languages in Compiler construction. The constraints that are considered are typically static semantic constraints, such as scoping and typing rules. These constraints are specified using for example attribute grammars [13], which have been shown to be efficiently executable by compilers. Attribute grammars are not very concise specifications of consistency as one constraint is typically spread over a large number of products. On the other hand, they have been shown to be very amenable to efficient execution [11, 12], which is an important property when considering compiling large amounts of source code on slow processors. We have made a slightly different trade-off decision with xlinkit and favour conciseness of the constraint definition over efficiency. This is particularly appropriate given the small size of derivative trade documents, which are in the order of 17KBytes.

The work on attribute grammars was then taken on for the construction of syntax-directed editors and software engineering environments, such as Gandalf [8], Synthesizer Generator [17], IPSEN [14], Centaur [1] and GOODSTEP [6]. The focus of these environments was to incrementally check constraints during editing. This could only be achieved by translating attribute or graph grammars into efficiently executable code. Our focus is not (yet) on supporting the editing of trade representations, but to support the batch validation that occurs when trades are exchanged between organisations or different departments within an organisation. Provision of support for incremental checks is therefore not necessary and instead we favour the flexibility that comes with interpretation of constraints in the xlinkit rule engine.

The weakness of expressing constraints in XML DTDs and Schemas has been recognized for quite some time now. Various approaches have been reported that use XSLT [2] for validation. In [7], we report about the TIGRA enterprise application integration architecture that uses XML as transport representation for financial trades. In that architecture we have used XSLT stylesheets to express constraints. The expressive power of XSLT stylesheets is considerably lower than that of xlinkit in that xlinkit supports the full power of first-order logic. Moreover, xlinkit carefully separates the concerns of constraint specification, document and rule location, and provision of diagnostic feedback, which would be intertwined in XSLT.

Rick Jeliffe's Schematron [10] also uses XSLT to translate documents into reports about their consistency. However, Schematron manages to conceal the use of XSLT and provides a higher level of abstraction for the definition. Although Schematron works quite well for validating single documents it would not allow us to express constraints across different documents, e.g. to check trades against reference data or workflow representations or to compare two trades in different representations.

We have also compared xlinkit rules with OMG's Object Constraint Language (OCL) [19]. OCL was defined to declare constraints in UML diagrams or MOF meta models. OCL was not defined with an aim to be executable. In particular, it allows for infinite sets, which prevents it from being executed effi-

ciently. The focus of xlinkit, however, was to be as expressive as possible, while still being executable in polynomial time.

8 Conclusions

We have shown in this paper that xlinkit can express consistency constraints between financial documents at a very high level of abstraction. We have presented the requirements and classified the different kinds of constraints. We have used this classification to show completeness of our rule language in the support for defining consistency constraints. We have used xlinkit to implement a full suite of validation rules for FpML 1.0 and used this implementation to evaluate performance. As a result, we are confident that it is feasible to employ xlinkit in straight through processing architectures to validate trades that are exchanged between different organisations or different departments in the same organisation.

The success of the evaluation has led the FpML steering committee to consider a proposal to use xlinkit as the standard language to express validation constraints for FpML documents. The high-level of abstraction provided by xlinkit makes it particularly suitable. Moreover, FpML will use the xlinkit rule language as the reference implementation for an FpML validation engine against which the industry can compare their proprietary implementations.

Finally, we believe that beyond derivative trading, xlinkit has many different potential application areas. Problems similar to the ones described in this paper occur whenever semi-structured information is exchanged between different organizations. This is the case in e-commerce and electronic business. We have, for example also investigated the use of xlinkit in procurement processes, such as those standardized by RosettaNet. Moreover, we have successfully applied xlinkit to detecting inconsistencies in software engineering documents.

References

[1] P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: The System. *ACM SIGSOFT Software Engineering Notes*, 13(5):14–24, 1988. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, MA, USA.

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

[3] 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, November 1999.

[4] 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.

[5] D. Dui. Fpml 1.0 validation rules. http://www.cs.ucl.ac.uk/staff/D.Dui/FpML/Rules1.0/validation.html, 2002.

[6] 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.

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

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

[9] ISO 6166. Securities and related financial instruments – International securities identification numbering system (ISIN). Technical report, International Standards Organisation, 2001.

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

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

[12] U. Kastens and W. M. Waite. Modularity and reusability in attribute grammars. *Acta Informatica*, 31:601–627, 1991.

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

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

[15] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2002. To appear.

[16] C. Nentwich, W. Emmerich, and A. Finkelstein. Flexible Consistenc Checking. Research note, University College London, Dept. of Computer Science, 2001. Submitted for Publication.

[17] 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.

[18] B. Thal. Fpml tools workshop. http://www.fpml.com/tools/toolswork.asp, August 2001.

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