Transformation and Reaction Rules for Data on the Web

James Bailey

NICTA Victoria Laboratories
Department of Computer Science and Software Engineering
University of Melbourne
Victoria, 3010
Email: `jbailey@csse.unimelb.edu.au`

Abstract

The transformation and manipulation of XML is an increasingly important research topic. This paper examines a number of issues with regard to languages for transforming and reacting to changes on XML data. On the transformation side, we focus on XSLT, a powerful language for converting XML data into other formats. We look at analysis and optimisation issues for XSLT, as well as support for debugging and automatic generation. On the reactivity side, we focus on an event-condition-action rule approach, which is a natural candidate for the support of reactive functionality on XML repositories.

Keywords: XML, XSLT, active rules

1 Introduction

The Extensible Markup Language, XML, has recently emerged as a new standard for information storage, representation and exchange on the World Wide Web. To support the use of XML, many kinds of languages and standards are needed. In this paper, we focus on two important and closely related aspects of the manipulation XML data, namely XML transformations and XML reactivity. The former is concerned with methods for conversion of XML documents from one form to another and we focus on the *Extensible Stylesheet Transformations* language known as XSLT (World Wide Web Consortium 1999). We see there are a number of important research issues for XSLT, with regard to its analysis, optimisation, automatic generation and debugging. The latter aspect, XML reactivity, concerns integrating rules with XML documents, in order to provide the capability for automatic reaction to important events. We examine the active rule approach to XML, which aims to leverage past work in the area of active databases and make it suitable for the XML context. We see that while many techniques can be reused, a number of new challenges also arise.

1.1 Outline

This paper is divided into two main parts. In the first part, we review XSLT in section 2 and then examine research themes for its analysis (Section 3), optimisation (Section 4), automatic generation (Section 5) and debugging (Section 6). The second part, in section 7, looks at reaction rules for XML. Section 8 provides a summary and conclusion.

2 XML Transformations

XSLT (World Wide Web Consortium 1999) is a language with two main applications. The first of these deals with the presentation of XML data, which involves displaying XML documents in a sensible layout. The XSLT program or *stylesheet*, stores a set of rules that can be used for determining how the content of elements in an XML document should be displayed in a browser. This is increasingly important as the Web begins to include more semantic information and user pages migrate from HTML to XML.

The second main application of XSLT is for data exchange and conversion of one XML document into another XML document having a different structure. For example, in the publishing industry, it is common to need multiple versions of the same document. Traditionally, a document would be manually duplicated into all the required media, such as a newspaper article, Web page and magazine. However, by applying transformation rules in XSLT stylesheets, a single XML document containing the contents of the material can be straightforwardly converted into other formats. A similar application exists in the area of data integration, where XSLT can be used when merging two or more different documents into another, single document.

A typical XSLT stylesheet consists of a collection of template rules. Each rule can be applied by pattern matching the condition of the rule against the input XML document and then possibly generating some output. These template rules may be applied in any order. XSLT templates allow the designer to specify how the transformation should be carried out. Execution of an XSLT stylesheet essentially corresponds to 'walking' through the tree, applying the appropriate templates. Furthermore, XSLT stylesheets disallow variable assignment, thus eliminating side-effects. Characteristics such as lazy evaluation and purity make XSLT similar to functional languages.

Although in this paper we will focus on XSLT, we also note that other languages exist which can be used to transform XML, an important one being XQuery (World Wide Web Consortium 2004*b*). At the time of writing though, this is still in process of being standardised and software implementations are only just beginning to emerge. Another interesting language is Xcerpt (Schaffert & Bry 2004), based on on logic programming. It places an emphasis on the separation between query patterns and output answer patterns (construction patterns). The query part is limited to retrieving data and constraining results, while the construction part is limited to reassembling the data and creating new data.

3 Analysis of XSLT Transformations

We now discuss some of the issues regarding analysis of XSLT stylesheets. Analysis techniques allow the programmer to gain better knowledge about the behaviour of the transformations they have written. The presentation is based on work in (Dong & Bailey 2004*b*).

An XSLT stylesheet consists of a set of templates and execution of the stylesheet is by recursive application of individual templates to the source XML document. This recursive application of templates is an essential aspect of XSLT. However, important problems can arise when designing templates. Firstly, some templates within an XSLT stylesheet may never be applied during execution, regardless of the XML source being input. We call such templates *unreachable templates.* Secondly, there may be templates which issue calls to templates which don"t exist (rather like a function call to a non-existent function). We term this situation as a *missing template* and it is a likely indication of an error in the stylesheet. Thirdly, there may exist pairs of templates, which appear to call each other, based on the syntactic structure of the stylesheet, but in fact cannot, due to underlying constraints which exist within an accompanying DTD. We call these *invalid template calling relationships.* Fourthly, the XSLT stylesheet itself may loop forever on some XML inputs. An infinite template calling loop can have catastrophic consequences and result in failure of execution of the transformation. Current XSLT processors offer no support for detecting or handling such infinite behaviour. Instead, outputs are often cryptic stack overflow errors, or a blank output window in the browser. This is the problem of XSLT *termination.* Work in (Dong & Bailey 2004*b*) examines these four questions and proposes an analysis method based on the construction of a template association graph, which conservatively models the control flow of the stylesheet. A large body of work on termination in other contexts such as those for active databases also exists, e.g.(Bailey & Poulovassilis 1999, Aiken, Widom & Hellerstein 1995). Connections between active rules and XSLT have been observed in (Bonifati, Ceri & Paraboschi 2001*a*, Bonifati, Ceri & Paraboschi 2001*b*) and followed up by work in (Bailey, Pouolvassilis & Wood 2002*a*, Bailey, Pouolvassilis & Wood 2002*b*).

There is also an important line of theoretical research with regard to analysis of the behaviour of XSLT. Work in (Milo, Suciu & Vianu 2000) presents a theoretical model of XSLT and examines a number of decision questions for fragments of this model. Work in (Martens & Neven 2004) examines the question of whether the output of an XML transformation conforms to a particular document type. This is more generally related to the problem of detecting constraint violations due to transformations. Given a source XML and an XSLT stylesheet, determine statically whether applying the stylesheet can violate any constraints that have been specified on the output document. e.g. Ensure that a table in the output document contains every element in the input document.

Some important research issues in analysis are:

- Methods to analyse larger fragments of the XSLT language syntax are needed. Given the computational completeness of XSLT, obviously such methods are likely to be conservative. Constructs such as complex XPath expressions, functions and data joins all present difficulties for existing techniques. Similar to syntax, methods that take into account semantic complexities of the language are needed. e.g. Handling of template priorities.
- The use of analysis results is another important theme. Similar to the use of techniques such as the chase in the optimisation of conjunctive queries, investigation into the use of results of analysis in the optimisation of XSLT transformations is needed. Relationships of analysis information to debugging is likely to be another fruitful area (partially discussed in Section 6).

4 Optimisation of XSLT

XSLT can be viewed as a high-level, declarative programming language. This has the benefit that stylesheets written in this language are often amenable to a wide range of optimisation techniques.

Server-side XSLT processing is an example of where optimisation is important. Here, a Web server receives form based user queries (from a client PC or hand-held browsing device), applies them to a relational database, exports the result as XML and then applies an XSLT stylesheet to deliver the result as HTML back to the user. If the system is to be scalable with respect to the number of users, then it is crucial that application of the XSLT stylesheet be efficient. Optimisation techniques are particularly important for applications which need to perform a high volume of transformations.

Work in (Dong & Bailey 2004*a*) looks at the use of template specialisation as a technique for improving the performance of XSLT stylesheets. This is a kind of partial evaluation, for which there already exists a rich body of work, especially for functional languages (Jones 1996) (the difference between XSLT and say Haskell, is that XSLT is data intensive and there is also often an accompanying DTD for the input, providing more explicit knowledge about the underlying input constraints). The main idea is that server-side XSLT stylesheets are often written to be generic and may contain a lot of logic that is not needed for execution of the transformation with reference to given user query inputs. Such inputs are passed as parameters to the XSLT stylesheets at run-time, often using a mechanism such as forms in a Web browser. For example, a user reading a book represented in XML might pass a parameter to an XSLT stylesheet referring to the number of the chapter they wish to see presented. (i.e. transformed from XML to HTML by the XSLT stylesheet). The XSLT stylesheet may obviously contain logic which is designed for presenting the contents of other chapters, but it will not be needed for this user query. Given knowledge of the user input space, it is possible to instead automatically (statically) create different specialised versions of the original XSLT stylesheet, that can be invoked in preference to the more generic, and possibly larger version at run-time. In our book example, specialised XSLT stylesheets could be created for each chapter. Since the specialised versions can be much smaller than the original stylesheet, important savings in execution time and consequently response time are possible. This work presupposes the use of a cost model for estimating the run-time efficiency of a given stylesheet, which in itself is a challenging task.

Looking at other work in the area of XSLT stylesheet optimisation, work in (Villard & Layaida 2002) discusses incremental methods for evaluating multiple transformations, while work in (Jain, Mahajan & Suciu 2002) takes the approach of mapping XSLT stylesheets into SQL for higher performance. This latter work suffers from the drawback that not all XSLT stylesheets can be mapped into SQL, however. Optimisation of XSLT is also discussed in (Kay 2004), which outlines the internals of the popular XSLT processor Saxon. XSLT optimisation has been considered by Z. Guo, M.Li et al in (Guo, Li, Wang & Zhou 2002). They use a streaming model (SPM) to evaluate a subset of XSLT. Using SPM, an XSLT processor can transform an XML document to other formats without using extra buffer space. XPath (Clark & DeRose 1999) query optimisation has also been considered in a large number of papers, e.g. (Gottlob, Koch & Pichler 2003*a*, Buneman, Fan & Weinstein 2000, Gottlob, Koch & Pichler 2003*b*, McHugh & Widom 1999).

Some research issues in the area of optimisation are:

- Constructing accurate cost models for estimating the output size and/or runtime of an XSLT stylesheet. Results in this direction have been published for estimating the selectivity of (much simpler) XPath expressions, e.g. (Aboulnaga, Alameldeen & Naughton 2001).
- Developing a standard set of benchmarks for XSLT. A recent step in this direction is the XSLTMark suite (Datapower 2001).
- Similar to the direction mentioned for analysis in Section 3, using static analysis information, such as likely template calling patterns, to optimise the stylesheet.

5 Automatic Generation of XSLT

XSLT is a language that is widely considered difficult to learn (Leventhal 1999). Rendering from XML to HTML using XSLT requires skills and knowledge of both XSLT programming and also Web page styling. This motivates the consideration of methods for automatically generating XSLT. Given a source XML document and a desired output XML/HTML document, an XSLT stylesheet is automatically generated to transform the source into the output. The generated stylesheet contains rules needed to transform the source document to the output document and can also be applied to other source documents having the same structure. The important feature here is that users can generate an XSLT stylesheet based solely on their knowledge of XML and HTML. They need not know anything about the syntax or programming of XSLT.

Such automatic generation is a useful facility for students and Web developers learning the XSLT language. With an automatic generation tool, they are able to see and understand how the XSLT stylesheet should look, in order to transform a particular XML document into a desired output. In addition, this tool can also be useful for aiding the XSLT development process. Programmers may use the automatically generated stylesheet as the starting point for something more complex.

A naive solution to the problem of automatic generation is to create an XSLT stylesheet consisting of only one template rule, whose pattern matches the XML root element and whose template contains the HTML document markup (in other words create a stylesheet which is very specific to the desired output). This naive approach has a major drawback in terms of reusability. Such a stylesheet is specific for transforming the given XML document only and could not be used to transform other XML documents having similar structure. Instead, we wish to generate stylesheets that can be applied to documents that are instances of the same document class as the given XML document. There are some interesting connections to genericity here. Work in (Erwig 2003) looks at information preserving transformations for XSLT. Work in (Waworuntu & Bailey 2004) presents the XSLTGen system, which shows how automatic generation of XSLT is possible via the discovery and use of semantic mappings between the input and output.

There is also other related work on the automatic generation of XSLT stylesheets. For visualising the transformation process, there is the XSLbyDemo system (Ono, Koyanagi, Abe & Hori 2002). Tree matching is another related area and a number of well established algorithms exist. Work done in (Tai 1979, Selkow 1977), on the tree distance problem or tree-to-tree correction problem and work done in (Chawathe, Rajaraman, Garcia-Molina & Widom 1996, Lim & Ng 2001) on change detection, compare and discover the sequence of edit operations needed to transform the source tree into the result tree. These algorithms principally examine the tree structures (as opposed to text/data values). Conversely, systems such as XSLTGen (Waworuntu & Bailey 2004) use both text values and document structure to compare documents.

In the field of semantic mapping, there is a significant amount of work that has been done on schema matching (Rahm & Bernstein 2001). The main difference between schema matching and automatic transformation generation is that mappings are believed to exist between elements in the source and destination documents in automatic generation, since the output document is derived by the user, as opposed to schema matching where entirely different vocabularies could be used. Moreover the mappings generated by the matching process are used to generate code. The CLIO system (Popa, Velegrakis, Miller, Hernandez & Fagin 2002) examines the mapping problem for mapping between combinations of XML and relational schemas and methods for generating queries to convert between the two.

Some research issues for automatic generation of XSLT are:

- Gaining a deeper understanding of what it means for an automatically generated XSLT stylesheet to be generic.
- Developing more powerful methods to improve the kinds of XSLT stylesheets generated, possibly relying on complex information such as $m - m$ mappings.
- Developing standard test suites for evaluating the quality of generated stylesheets, in a similar spirit to the well known test datasets that are important in the area of machine learning.

6 Debugging of XSLT

Similar to analysis of rules, as discussed in section 3, a related important issue is that of rule debugging. Being a young language, there are not yet many techniques and tools for writing error free XSLT stylesheets, compared to general purpose programming languages such as C and Java. Also, as discussed previously in section 5, it is a complex language and both seasoned programmers and beginners alike find it difficult to learn. Indeed little academic work has been undertaken on XSLT debugging, an exception being (Bae & Bailey December 2003). Instead, work in this area is mainly represented by commercial product implementations. There has been a proliferation of tools developed by major companies, with the XSLT debugging feature included within a much bigger tool in an Integrated Development Environment (IDE). These tools often have highly usable interfaces, but the main focus is on the development environment for XSLT, rather than innovative debugging techniques. This lag in academic research is perhaps explained by the pressure of market demands and the rapidly increasing uptake of XSLT. In (Bae & Bailey December 2003) it is shown that backwards debugging, template slicing and visualisation are all important techniques and these have been implemented in a prototype tooled called CodeX. Visualisation is a result oriented technique that moves programmer attention away from internal computations and allows them to focus directly on the output. It has achieved great success in the area of debugging. A particular challenge is the visualisation of recursive template applications, due to inherently recursive nature of the language. The functional nature of XSLT makes backwards debugging a particularly natural idea, due to its similarity to functional languages (Nilsson & Fritzson 1994). Slicing is the act of breaking large program code into many coherent pieces (Weiser 1982), called slices of a program. In XSLT, we consider a slice to consist of a single template rule. This allows the users to interact with a subset of the templates available and they have the option to debug only the necessary templates and check whether their transformations are performed correctly. This is particularly useful in the context of software maintenance, where the programmer does not want to step through the whole stylesheet, being presented with template rules that have been verified through earlier debugging sessions. A number of tools also exist for debugging XPath expressions, e.g. XPath Analyzer (ALTOVA 2005).

Some research issues in the area debugging are:

- Developing debugging strategies that deal with the full XSLT language and that are able to precisely isolate and explain errors. Integration of analysis information within debugging sessions is also important and was mentioned previously in Section 3.
- Integrating XPath debugging and XSLT debugging. Both languages operate at different levels of granularity and it is a challenge to combine debugging feedback to the user from both levels.
- Since debugging sessions can add a lot of extra runtime overhead, techniques for debugging efficiently are required. Incremental methods for XSLT optimisation (Villard & Layaida 2002) may prove useful here.

7 Reaction Rules for XML

With the increasing use of XML in applications such as data warehousing, e-commerce and e-learning (Abiteboul, Cluet, Ferran & Rousset 2002, Bonifati et al. 2001*a*, Bonifati et al. 2001*b*, Cluet, Veltri & Vodislav 2001, Ishikawa & Ohta 2001, Tatarinov, Ives, Halevy & Weld 2001), there is a rapidly growing need for reactive functionality on XML repositories. *Event-condition-action* (ECA) rules are a natural candidate for this. ECA rules automatically perform actions in response to events provided the stated conditions hold. They are used in conventional data warehouses for incremental maintenance of materialised views, validation and cleansing of data and maintaining audit trails. By analogy, ECA rules can also be used as an integrating technology for providing this kind of functionality on XML repositories. Further uses include checking key and other constraints on XML documents and performing automatic repairs when violations of constraints are detected. In a 'push' type environment, they can be used for automatically broadcasting information to subscribers as the contents of relevant documents change. They can also be employed as a flexible means of maintaining statistics about document and Web site usage and behaviour.

There are two main advantages in using ECA rules to support such functionality, as opposed to implementing it directly in a programming language such as Java. First, ECA rules allow an application's reactive functionality to be defined and managed within a single rule base, rather than being encoded in diverse programs. This enhances the modularity and maintainability of such applications. Second, ECA rules have a high level, declarative syntax and are thus amenable to powerful analysis and optimisation techniques, which cannot be applied if the same functionality is expressed directly in programming language code.

An alternative way to implement the functionality described above might be to use XSLT to transform source XML documents. However, XSLT would have to process an entire document after any update to it in order to produce a new document, whereas we are concerned with the detection and subsequent processing of updates of much finer granularity. Also, using ECA rules allows direct update of a document, whereas XSLT requires a new result tree to be generated by applying transformations to the source document. ECA rules have been used in many settings, including active databases (Paton 1999), workflow management, network management, personalisation and publish/subscribe technology (Adi, Botzer, Etzion & Yatzkar-Haham 2000, Bonifati et al. 2001*a*, Bonifati et al. 2001*b*, Ceri, Fraternali & Paraboschi 1999, Pereira, Fabret, Llirbat & Shasha 2000) and specifying and implementing business process (Abiteboul, Vianu, Fordham & Yesha 2000, Ceri & Fraternali 1997, Ishikawa & Ohta 2001).

An ECA rule has the general syntax:

on event *if* condition *do* action

The event part describes a situation of interest and dictates when the rule should be triggered. The condition part determines if the (XML) database is in a particular state. It is a query over the database and its environment. The action part of a rule describes the logic to be performed if the condition evaluates to true. It is usually a sequence of modifications applied to the database, using the same syntax as that used by updates within a transaction. More details on the foundations of ECA rules in active databases for relational and Object Oriented systems and descriptions of a range of prototypes can be found in (Paton 1999, Widom & Ceri 1995). The semistructured nature of XML data gives rise to new issues affecting the use of ECA rules. These issues are principally linked to choice of appropriate language syntax and execution model:

- *Event granularity:* In the relational model, the granularity of data manipulation events is straightforward, since insert, delete or update events occur when a relation is inserted into, deleted from or updated, respectively. With XML, this kind of strong typing of events no longer exists. Specifying the granularity of where data has been inserted or deleted within an XML document and path expressions that identify locations within the document now become necessary.

- *Action granularity:* Again in the relational model, the effect of data manipulation actions is straightforward, since an insert, delete or update action can only affect tuples in a single relation. With XML, actions now manipulate entire subdocuments and the insertion or deletion of subdocuments can trigger a different set of events. Thus, analysis of which events are triggered by an action can no longer be based on syntax alone. Also, the choice of an appropriate action language for XML is not obvious, since there is as yet no standard for an XML update language.

Compared to rules for relational databases, ECA rules for XML data are more difficult to analyse, due to the richer types of events and actions. However, rules for XML have arguably less analysis complexity than rules for object oriented data. This stems from the fact that object oriented databases may permit arbitrary method calls to trigger events, and determining triggering relationships between rules may therefore be as difficult as analysing a program written in a language such as C++ or Java. ECA rules for XML, in contrast, can be based on declarative languages such as XQuery and XPath and so are more amenable to analysis, particularly with a set of natural syntactic restrictions.

In recent work (Bailey et al. 2002*a*, Bailey et al. 2002*b*), we specified a language for defining ECA rules on XML data, based on the XPath and XQuery standards. We also developed techniques for analysing the triggering and activation relationships between such rules. A number of other ECA rule languages have also been proposed, although none of this work has focused on rule analysis.

Bonifati et al (Bonifati et al. 2001*a*) discuss extending XML repositories with ECA rules in order to support e-services. Active extensions to the XSLT and Lorel (Abiteboul, Quass, McHugh, Widom & Wiener 1997) languages are proposed that handle insertion, deletion and update events on XML documents. Bonifati et al (Bonifati et al. 2001*b*) discuss a more specific application of the approach to push technology, where rule actions are methods that cannot update the repository and hence cannot trigger other rules.

ARML (Cho, Park, Hyum & Kim 2002) provides an XML based rule description for rule sharing among different heterogeneous ECA rule processing systems. GRML (Wagner 2002) is a multipurpose rule markup language for defining integrity, derivation and ECA rules. It uses an abstract syntax based on RuleML, leaving the mapping to a real language for each underlying system implementation. GRML aims to provide semantics for defining access over distributed, heterogeneous data sources for rule evaluation and allows the user to declare most of the semantics necessary for processing a rule, and to evaluate events and conditions coming from heterogeneous data sources. Other related work is (Tatarinov et al. 2001), which proposes extensions to the XQuery language (World Wide Web Consortium 2004*b*) to incorporate update operations. Triggers are discussed in (Tatarinov et al. 2001) as an implementation mechanism for deletion operations on the underlying relational store of the XML, but provision of ECA rules at the 'logical' XML level is not considered.

Closely related to reactive functionality for XML, is the issue of dynamic XML documents, where some data within the document is provided by embedded calls to Web services. Work in (Abiteboul, Bonifati, Cobena, Manolescu & Milo 2003) examines issues in this area with regard to distribution and replication.

Recent work in (Bernauer, Kappel & Kramler 2004) and (Bry & Patranjan 2005) looks at the issue of complex events for rules in XML. (Bernauer et al. 2004) investigates the intricacies of cascading complex events when updates are performed on a document, while (Bry & Patranjan 2005) presents the language XChange, which is used for defining rules for XML data that can be triggered by complex events.

Some research issues for active rules on XML are:

- Successfully combining the eventual standard XML update language within ECA rules. Most proposals in the area have so far assumed a action syntax based on XQuery.

- Similar to XML transformations, it is again important to be able to ensure that a document remains valid with respect to its DTD or XML schema after rule execution. Analysis methods are thus needed to predict the effect of rules on documents. Work in a relational context which looks at relational transducers (Abiteboul et al. 2000) may be useful here.

- There are a number of emerging applications closely related to XML and the Web, such as the Resource Description Framework ((World Wide Web Consortium 2004*a*)) and sensor networks. Active rules are likely to be important in both contexts (Papamarkos, Poulovassilis & Wood 2003, Zoumboulakis, Roussos & Poulovassilis 2004).

8 Summary

In conclusion, we see that many new and interesting challenges exist for XML transformations and reactivity. In this paper we have focused on the XSLT and active rule approaches to manipulation of, and reaction for, XML data. It will be interesting to see the impact on both these areas, once XQuery becomes a W3C recommendation and further tools begin to be developed.

Acknowledgements: I would like to thank the following people, for the discussions and collaborations I've had in the areas of XML transformations and reactive rules: Eric Bae, Francois Bry, Ce Dong, Paula Patranjan, Alex Poulovassilis, Stella Waworuntu and Peter Wood.

References

Abiteboul, S., Bonifati, A., Cobena, G., Manolescu, I. & Milo, T. (2003), Dynamic XML documents with distribution and replication, *in* 'Proceedings of the ACM Conference on the Managament of Data (SIGMOD)', pp. 527–538.

Abiteboul, S., Cluet, S., Ferran, G. & Rousset, M. (2002), 'The Xyleme project', *Computer Networks* **39**, 225–238.

Abiteboul, S., Quass, D., McHugh, J., Widom, J. & Wiener, J. (1997), 'The Lorel query language for semistructured data', *VLDB Journal* **1**(1), 68–88.

Abiteboul, S., Vianu, V., Fordham, B. & Yesha, Y. (2000), 'Relational transducers for electronic commerce', *Journal of Computer and System Sciences* **61(2)**, 236–269.

Aboulnaga, A., Alameldeen, A. R. & Naughton, J. F. (2001), Estimating the selectivity of XML path expressions for internet scale applications, *in* 'VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy', pp. 591–600.

Adi, A., Botzer, D., Etzion, O. & Yatzkar-Haham, T. (2000), Push technology personalization through event correlation, *in* 'Proc 26th International Conference on Very Large Databases (VLDB)', pp. 643–645.

Aiken, A., Widom, J. & Hellerstein, J. M. (1995), 'Static analysis techniques for predicting the behavior of active database rules', *ACM Transactions on Database Systems* **20**(1), 3–41.

ALTOVA (2005), 'XPath Analyzer', http://www.altova.com/features_xpath.html.

Bae, E. & Bailey, J. (December 2003), CodeX: An approach for debugging XSLT transformations, *in* 'Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE)', IEEE Computer Society, pp. 309–312.

Bailey, J. & Poulovassilis, A. (1999), An abstract interpretation framework for termination analysis of active rules, *in* 'Proc. 7th International Workshop on Database Programming Languages, LNCS 1949', Kinloch Rannoch, Scotland, pp. 249–266.

Bailey, J., Pouolvassilis, A. & Wood, P. T. (2002*a*), An Event-Condition-Action Language for XML, *in* 'Proceedings of the World Wide Web Conference (WWW)'.

Bailey, J., Pouolvassilis, A. & Wood, P. T. (2002*b*), 'Analysis and optimisation for event-condition-action language on XML', *Computer Networks* **39**, 239–259.

Bernauer, M., Kappel, G. & Kramler, G. (2004), Composite events for XML, *in* 'Proceedings of the 13th International Wide Web Conference', ACM Press, pp. 175–183.

Bonifati, A., Ceri, S. & Paraboschi, S. (2001*a*), 'Active rules for XML: A new paradigm for E-services', *VLDB Journal* **10**(1), 39–47.

Bonifati, A., Ceri, S. & Paraboschi, S. (2001*b*), Pushing reactive services to XML repositories using active rules, *in* 'Proceedings of the 10th World-Wide-Web Conference (WWW)'.

Bry, F. & Patranjan, P. (2005), Reactivity on the Web: Paradigms and Applications of the Language XChange, *in* 'To appear in The 20th Annual ACM Symposium on Applied Computing'.

Buneman, P., Fan, W. & Weinstein, S. (2000), 'Query optimization for semistructured data using path constraints in a deterministic data model', *Lecture Notes in Computer Science* **1949**.

Ceri, S. & Fraternali, P. (1997), *Designing Database Applications with Objects and Rules:The IDEA Methodology*, Addison-Wesley.

Ceri, S., Fraternali, P. & Paraboschi, S. (1999), Data-driven one-to-one web site generation for data-intensive applications, *in* 'Proc. 25th International Conference on Very Large Databases (VLDB)', pp. 615–626.

Chawathe, S., Rajaraman, A., Garcia-Molina, H. & Widom, J. (1996), Change Detection in Hierarchically Structured Information, *in* 'Proceedings of the 1996 International Conference on Management of Data', Montreal, Canada, pp. 493–504.

Cho, E., Park, I., Hyum, S. J. & Kim, M. (2002), Arml: An active rule markup language for heterogeneous active information systems, *in* 'Proceedings of RuleML 2002'.

Clark, J. & DeRose, S. (1999), *XML Path Language (XPath) Version 1.0*, W3C Recommendation. *http : //www.w3.org/TR/xpath.*

Cluet, S., Veltri, P. & Vodislav, D. (2001), Views in a large scale XML repository, *in* 'Proceedings of the 27th International Conference on Very Large Databases (VLDB)', pp. 271–280.

Datapower (2001), 'XSLTMark version 2.1.0', http://www.datapower.com/xmldev/xsltmark.html.

Dong, C. & Bailey, J. (2004*a*), Optimization of XML transformations using template specialization, *in* 'To appear in Proceedings of the 5th International Conference on Web Information Systems Engineering (WISE 2004)'.

Dong, C. & Bailey, J. (2004*b*), Static analysis of XSLT programs, *in* 'Proceedings of the 15th Australasian Database Conference (ADC)', Vol. Australian Computer Science Communications 26(2), pp. 151–160.

Erwig, M. (2003), Toward the automatic derivation of XML transformations, *in* '1st Int. Workshop on XML Schema and Data Management, LNCS 2814', pp. 342–354.

Gottlob, G., Koch, C. & Pichler, R. (2003*a*), 'XPath processing in a nutshell', *SIGMOD Rec.* **32**(2), 21–27.

Gottlob, G., Koch, C. & Pichler, R. (2003*b*), XPath query evaluation: Improving time and space efficiency, *in* 'Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)'.

Guo, Z., Li, M., Wang, X. & Zhou, A. (2002), Scalable XSLT evaluation, *in* 'Proceedings of the 2004 Asia Pacific Web Conference (APWEB)'.

Ishikawa, H. & Ohta, M. (2001), An active web-based distributed database system for E-Commerce, *in* 'Proceedings of the International Web Dynamics Workshop, London'.

Jain, S., Mahajan, R. & Suciu, D. (2002), Translating XSLT programs to efficient SQL queries, *in* 'Proceedings of the eleventh international conference on World Wide Web', ACM Press, pp. 616–626.

Jones, N. (1996), 'An introduction to partial evaluation', *ACM Computing Surveys* **28**(3), 480–503.

Kay, M. (2004), XSLT and XPath Optimization, *in* 'Proceedings of XML Europe 2004'.

Leventhal, M. (1999), 'XSL considered harmful', http://www.xml.com/pub/a/1999/05/xsl/xslconsidered_1.html.

Lim, S. & Ng, Y. (2001), An Automated Change-Detection Algorithm for HTML Documents Based on Semantic Hierarchies, *in* 'Proceedings of the 17th International Conference on Data Engineering', Heidelberg, Germany, pp. 303–312.

Martens, W. & Neven, F. (2004), Frontiers of tractability for typechecking simple XML transformations, *in* 'Proceedings of the ACM Symposium on Principles of Database Systems (PODS)', pp. 23–34.

McHugh, J. & Widom, J. (1999), Query optimization for XML, *in* 'The VLDB Journal', pp. 315–326.

Milo, T., Suciu, D. & Vianu, V. (2000), Typechecking for XML transformers, *in* 'Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA', ACM, pp. 11–22.

Nilsson, H. & Fritzson, P. (1994), Algorithmic debugging for lazy functional languages, *in* 'Journal of Functional Programming', pp. 337–370.

Ono, K., Koyanagi, T., Abe, M. & Hori, M. (2002), XSLT Stylesheet Generation by Example with WYSIWYG Editing, *in* 'Proceedings of the 2002 International Symposium on Applications and the Internet', Nara, Japan.

Papamarkos, G., Poulovassilis, A. & Wood, P. T. (2003), Event-Condition-Action Rule Languages for the Semantic Web, *in* 'Proceedings of the First International Workshop on the Semantic Web and Databases', pp. 309–327.

Paton, N., ed. (1999), *Active Rules in Database Systems*, Springer-Verlag.

Pereira, J., Fabret, F., Llirbat, F. & Shasha, D. (2000), Efficient matching for web-based publish/subscribe systems, *in* 'Proc 7th Int. Conf. on Cooperative Information Systems (CoopIS'2000)', pp. 162–173.

Popa, L., Velegrakis, Y., Miller, R. R., Hernandez, M. A. & Fagin, R. (2002), Translating Web data, *in* 'Proceedings of the International Conference on Very Large Databases (VLDB)', pp. 598–609.

Rahm, E. & Bernstein, P. (2001), 'A Survey of Approaches to Automatic Schema Matching', *VLDB Journal* **10**(4), 334–350.

Schaffert, S. & Bry, F. (2004), Querying the web reconsidered: A practical introduction to Xcerpt, *in* 'Proceedings of Extreme Markup Languages 2004'.

Selkow, S. (1977), 'The Tree-to-Tree Editing Problem', *Information Processing Letters* **6**(6), 184–186.

Tai, K. (1979), 'The Tree-to-Tree Correction Problem', *Journal of the ACM* **26**(3), 422–433.

Tatarinov, I., Ives, Z., Halevy, A. & Weld, D. (2001), Updating XML, *in* 'Proc. ACM SIGMOD Int. Conf. on Management of Data', pp. 413–424.

Villard, L. & Layaida, N. (2002), An incremental XSLT processor for XML document manipulation, *in* 'Proceedings of the International World Wide Web Conference (WWW)'.

Wagner, G. (2002), How to design a general rule markup language, *in* 'Invited Talk at the Workshop XML Technoligien fur das Semantic Web (XSW 2002)'.

Waworuntu, S. & Bailey, J. (2004), XSLTGen: A system for automatically generating XML transformations via semantic mappings, *in* 'To appear in Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004)'.

Weiser, M. (1982), Programmers use slices when debugging, *in* 'Communications of ACM', Vol. 25, pp. 446–452.

Widom, J. & Ceri, S. (1995), *Active Database Systems*, Morgan-Kaufmann, San Mateo, California.

World Wide Web Consortium (1999), 'XSL Transformations (XSLT), Version 1.0', See `http://www.w3.org/TR/xslt`. W3C Recommendation.

World Wide Web Consortium (2004*a*), 'RDF Primer ', See `http://www.w3.org/TR/rdf-primer`. W3C Recommendation.

World Wide Web Consortium (2004*b*), 'XQuery 1.0: An XML Query Language', See `http://www.w3.org/TR/xquery`. W3C Working Draft.

Zoumboulakis, M., Roussos, G. & Poulovassilis, A. (2004), Active rules for sensor databases, *in* 'Proceedings of the International Workshop on Data Management for Sensor Networks, VLDB 2004'.