RAX

Random Access XML

Fundamentally Changing How XML is Used and Processed

A Tarari White Paper

This page intentionally left blank

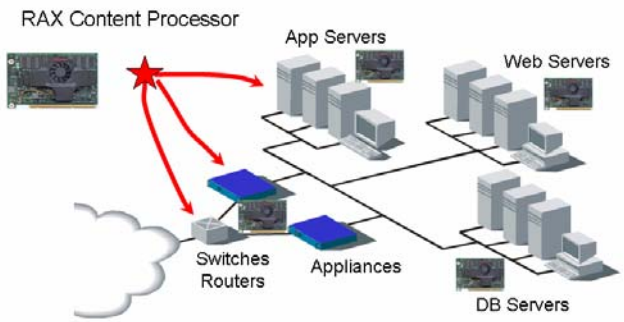
Table of Contents

This page intentionally left blank

Executive Summary

Tarari has developed the first successful implementation of **Random Access XML** (**RAX**): a breakthrough for XML processing which has been benchmarked at 40 to as much as 200 times the performance of conventional XML technologies.

Tarari's implementation of RAX is "industry changing" for two key reasons. First, it allows the information within XML formatted messages and files to be ***directly accessed*** without parsing and without the need for tree or streaming traversal. Second, it can be implemented ***natively in silicon.*** Tarari is the first company to create specialized silicon, designed and developed to power a RAX-based solution.



Deploying RAX to Accelerate XML

As this paper describes, XPath processing, which is the industry accepted way to best access XML information, but the performance penalty involved presents a true crisis. Tarari's addresses this crisis with its RAX Content Processor.

Random Access XML (RAX) represents a breakthrough in accelerating and simplifying XML processing. RAX, using XPaths as indices, gives applications direct access to any data within an XML message without parsing and without tree or streaming traversal. With all processing offloaded onto Tarari's RAX Content Processor, RAX is a near-zero-CPU-time solution. RAX can be used for any XML application that would traditionally be handled by DOM, SAX, JAXB, BEA's XMLBeans or any other approach. Simultaneous XPath also provides particularly fast, elegant solutions to XML applications using rule-based selection, look-up, classification, threat detection, Publish-Subscribe (PubSub), SOAP processing, XML security processing, and more, with up to 200 times the throughput of traditional XML processing. RAX is a comprehensive approach to XML message processing which also offers the possibility of combining direct access with streaming traversal of XML content, streaming transformation, and XML fragment interfaces to DOM- and SAX-based parsers with namespace support. The complete Tarari technology solution includes acceleration of XML security, XML compression, and Unicode character conversion.

RAX enables network switch, server, blade, and appliance vendors to create a variety of new applications that were simply not possible previously.

The most efficient way, up until the present time, to process XML has been to use one of two programming interfaces: DOM or SAX, but these software-based approaches simply don't scale. RAX on the other hand

utilizes W3C standards-based XPath queries which are fast becoming the most popular way to decode and route XML documents. Today's method of parsing using XPaths involves sequentially processing each XPath query—but this is also way too slow and compute-intensive to be practical in large-scale solutions.

The Tarari RAX Content Processor eliminates this barrier by enabling the simultaneous processing of very large groups of XPath queries and offloading compute-intensive tasks from the main processor. RAX enables gigabit-level XPath processing – something unattainable using software alone. The core technology enabling RAX is Tarari's Simultaneous XPath engine which offers remarkable performance characteristics that place it in a league of its own:

- Simultaneous XPath produces results directly from the input XML document, whereas DOM or SAX-based systems need to create an in-memory representation of the document
- Simultaneous XPath is vastly faster than any software-based XPath engines (e.g., Saxon, Xalan, libxml)
- Simultaneous XPath's performance is insensitive to the number of XPaths in an evaluation group and the complexity of the XPath expressions
- Simultaneous XPath handles XML namespaces and namespace prefixing on the fly without pre-scanning and declaration of prefixes
- Simultaneous XPath execution time increases linearly with the file size, without any performance degradation and without memory thrashing

Background and Motivation – XML Performance Crisis

This Tarari White Paper presents an analysis of a new approach to XML processing and assumes much more than a passing acquaintance with XML technologies. For the very intrepid reader without such a background we begin with an introduction to the topic.

XML is obviously an important topic. "Another conservative assumption, in my opinion, is that XML processing will become a primary activity of most software," wrote Software Development Times columnist Andrew Binstock[1]. While everyone knows that XML is rapidly becoming ubiquitous, some people still do not know that between all the new and very complicated things people would like to do with it (such as encoding and securing every business transaction) and its intrinsic nature (verbose and packed with complex relationships among data items), its performance is a **major issue**. Some are calling this major issue a "threat," others the "XML Performance Crisis."

The most efficient way, up until the present time, to process XML has been to use one of two programming interfaces: DOM or SAX. We'll explain more about these two choices later. There are a few other choices including JAXB, BEA's XMLBeans and XSLT, and although these choices offer convenience to the programmer, they provide less performance than DOM and SAX. Basically, none of the approaches to processing XML that people are

[1] Binstock, Andrew, "Helping Java Advance", SD Times, April 2004, Issue 99.

using today has any hope of solving the XML Performance Crisis.

We are going to talk a lot about XPath, a language for finding any part of an XML message or file and for testing the contents of any part of an XML message/file. XPath is useful for many things. It is used in XSLT to identify parts of an existing XML document, and as the source for creating a new XML document (transformation). It is used in the query languages supported by many databases to extract XML data. In fact, XPath is a general tool which can be used for almost any kind of XML processing. But it would not be evident to most people why we want to use XPath for most kinds of XML processing since XPath implementations are usually applications of DOM and cannot perform better than a custom-crafted DOM program. Tarari, however, has invented a way to execute XPaths faster than any existing XML tool could do the equivalent work. Much, much faster. Tarari uses this XPath as the heart of the solution to random access XML (RAX) processing. And RAX is The Solution to the XML Performance Crisis.

What kind of performance are we talking about? RAX may be 200 times faster than the XML software you are using today; we've clocked it that fast against one of most popular XML tools in use today.

The bottom line is that Tarari's implementation of RAX has reduced XML processing time to the level of noise. With RAX, the XML Performance Crisis is solved. Now, we

are able to create a new information infrastructure which takes full advantage of all that XML can offer.

We show how RAX enables us to create XML applications which run at network speeds and which seemed impossible until now: intelligent routing and inspection of XML messages, full XML security processing, Publish/Subscribe and real-time communications billing, to name just a few of the possibilities.

Accelerated XML– A New Approach from Tarari

Tarari has developed the first successful implementation of **Random Access XML** (**RAX**): a breakthrough for XML processing which has been benchmarked at 40 to as much as 200 times the performance of conventional XML technologies.

RAX in a Nutshell

RAX provides direct access to the data your application needs with near-zero parsing and other processing overhead. You identify the data you need through a set of XPaths and RAX indices into the source document for each matched XPath node. The processing that accomplishes this is done by Tarari's RAX Content Processor, a device which snaps into the server or appliance's standard PCI slot. Not only is the Tarari XML hardware much faster than software, but it also leaves the CPU free for other tasks. Your XML application can use the XPath node results directly or further traverse the document using the XPath indices as short-cut access points.

The Heart of RAX – Simultaneous XPath

At the heart of RAX is Tarari's **Simultaneous XPath.** Execution time with Simultaneous XPath is relatively unaltered by either the number of XPaths evaluated in a single pass or their complexity. While software-based XPath processors (e.g., Saxon, Xalan, libxml) suffer severe performance degradation at some file size threshold, the execution time for Simultaneous XPath is linear as the file size grows, and the throughput rate actually increases.

XML Breakdown

There have been two dominant XML programming models used by applications developers up to now: the Document Object Model (DOM) and Simple API for XML (SAX). DOM offers the most flexibility but is compute-intensive and requires large amounts of memory. Event-driven SAX was designed to be more lightweight and simple than DOM, with better performance on serially oriented XML processing and much lower memory consumption. To a large extent, SAX does achieve those goals on a relatively limited set of XML processing problems, but even on problems to which it is well suited, SAX still does not bring performance up to a level demanded by ubiquitous XML in Web Services and the IT and Networking infrastructure.

How severe is the problem? In a report published in November of 2003[2], IBM researchers documented many examples of the breakdown of IT infrastructure when XML processing is added to its workload:

- A large provider of IT Systems for processing secure brokerage transactions and securities data needs to support as many as 50,000 transactions per minute. The current generation of XML technology falls far short of this goal.
- A large bank could not maintain acceptable service levels after introducing XML to its

[2] Nicola, M., John, J.: "XML Parsing: A Threat to Database Performance", ACM, November 2003.

workflow. Average response time per XML transaction skyrocketed by a factor of 10.

- A life science company receives and produces XML documents between 10MB and 500MB in size. The demands on their server caused by XML processing left the company unable to include these documents in their system.

In a benchmark of a high-speed bulk loader for data warehousing, XML files took 26 times longer to load than flat files containing the same information.

Having sounding the tocsin, the authors have no solution to this fundamental threat. “Given the severe conflict between XML parsing/validation cost and transaction processing performance requirements, significant progress in research and development is needed,” they conclude.

The RAX Revolution

Underlying the pessimistic assessment of the IBM researchers is the assumption that there is no alternative to the parsing technology which today constitutes a performance roadblock. “There are two models of XML parsing: DOM and SAX,” they assert. With Tarari’s successful implementation of RAX, this assertion is no longer true and a way around the parsing roadblock now exists. Underpinning RAX is, effectively, a new parsing model that is neither DOM nor SAX, but a machine-based, micro-parallel processing strategy implemented in purpose-built XML hardware. Of course, the application programmer is able to access the machine parsing model transparently; he or she will typically employ RAX

methodology with its W3C standards-based, XPath-based indexing to directly access XML data. RAX-selected XML fragments can also be input into a SAX or DOM parser where needed when integrating with existing libraries.

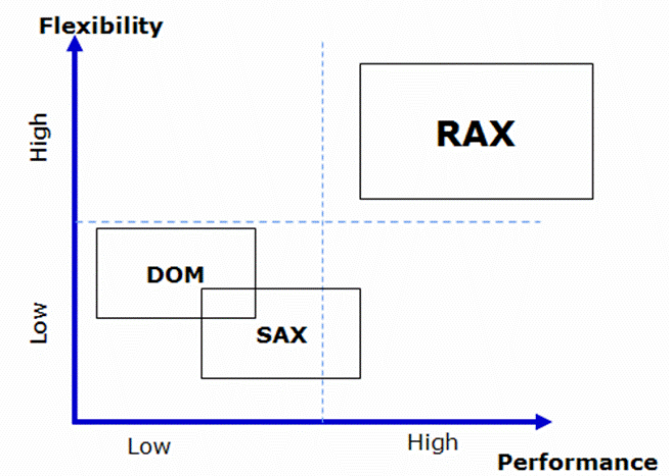


Figure 1—Comparison of DOM, SAX and RAX

SAX offers higher performance than DOM but is more difficult to use in most applications. RAX offers both the highest performance and the simplest approach to creating XML applications.

How can RAX change the negative prognosis for transactional XML reached by IBM researchers?

- Applications using RAX today have achieved the scalability and high XML message throughput required by Web Services and other transactional XML applications. Throughput rates exceeding 300,000 transactions per minute are attainable for many complex, XML-based operations.
- Latency for all types of XML operations is greatly reduced with RAX, with times as low as 100 microseconds on some operations.
- In contrast to most software, RAX maintains high performance on larger Web Services business

documents and remains level on outsized documents such as catalogs. Unlike software-based approaches, RAX does not suffer from resource degradation or memory thrashing on large documents.

- Processing time under RAX is largely unaffected by the complexity of XML documents. Tarari's special hardware rips through dense, complex, and namespace-heavy documents.

At its core, XML offers two features: 1) a simple generative grammar capable of creating expressive languages for a wide variety of information processing applications; and 2) a core syntax to which all language instances adhere. The first feature has made XML successful to the point of serving as the basis of thousands of languages and information applications. The core syntax gave us the ability to build a standard suite of interchangeable XML processing tools. RAX is the latest addition to this suite and, in all likelihood, the fulfillment of the promise of this approach – that, in time, tools to process the core syntax would be so highly perfected that the actual parsing time would become inconsequential.

RAX also greatly simplifies, and sometimes eliminates, the ugly details of the interface between the XML parsing machinery and application logic. RAX is a revolution not only for XML, but also for information technology, which is finally free to use the full expressive power of XML without concern for the overhead and programmatic details of its physical handling.

New Applications with Simultaneous XPath

The core technology enabling RAX is Tarari's Simultaneous XPath engine. XPath is a language for addressing elements and other parts of an XML document. It may also be thought of as an XML document query language and as a type of rule language for making assertions about the content in XML documents.

Tarari's Simultaneous XPath offers remarkable performance characteristics that place it in a league of its own:

- It is not necessary to create a DOM- or SAX-based, in-memory representation of the document. Simultaneous XPath produces results directly from the input XML document.
- Simultaneous XPath is vastly faster than any software-based XPath engine.
- Simultaneous XPath's performance is insensitive to the number of XPaths in an evaluation group and the complexity of the XPath expressions. Very large sets of XPaths can be evaluated in a single pass.
- Unlike software-based XPath engines, Simultaneous XPath handles XML namespaces and namespace prefixing on the fly without pre-scanning and declaration of prefixes.
- Simultaneous XPath execution time increases in a linear fashion with the file size, without any performance degradation and without memory thrashing. In fact, throughput can increase significantly as file sizes increase.

Simultaneous XPath is used to enable RAX programming but it is actually very useful for much more. XPath is a way to express *assertions* against XML documents. It is easy to see how the ability to process a set of XPath-based assertions in near-zero CPU time can form the basis of applications such as an XML firewall, where positive assertions prove that the content is valid and negative assertions prove the absence of malicious content.

Example - Publish-Subscribe (PubSub) Applications

Yet the family of applications that may be cast as "assertion problems" and solved in near-zero CPU time is even larger. For example, many problems that involve look-up, implemented typically with relational databases, can be mapped into assertion problems. One common problem that requires a combination of look-up and content-based routing is Publish-Subscribe (PubSub) messaging, used increasingly in content syndication.

Example - Rate Table Billing Applications

Another common look-up problem exists in real-time billing. Let's examine this problem in some detail.

Rate Table Billing Example in Traditional XML

First, let's look at a system, like thousands of XML-based applications today, based on the "classic" architecture.

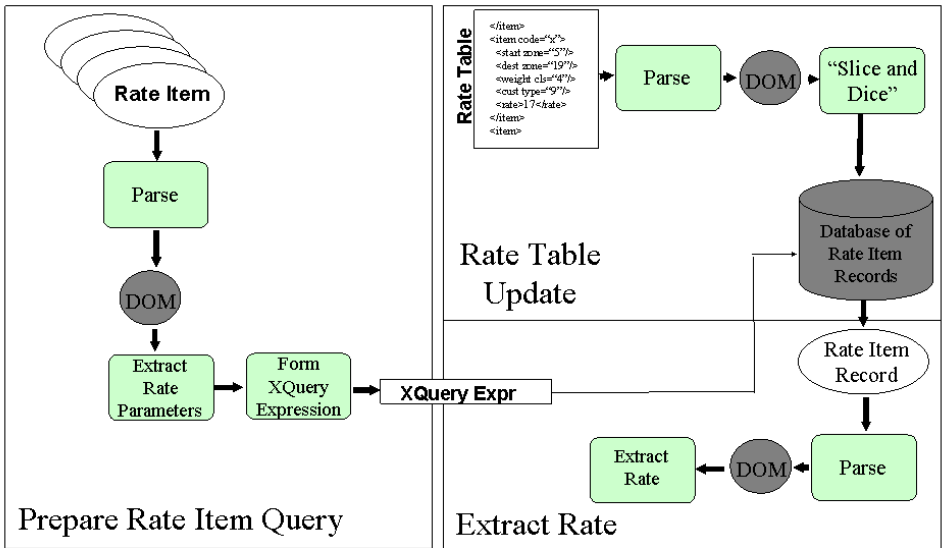


Figure 2—Traditional XML Application: Rate Table Billing

Rate information is published in the form of an XML Rate Table, containing a large number of records expressing a set of rate criteria and corresponding rates. In some instances (e.g., airfares) such tables may be updated as frequently as every day or hour; in others (e.g., payroll taxes), more infrequently, say, every quarter or so. Each time the Rate Table is updated, the XML document is parsed, resulting in a DOM or a SAX structure in memory which is then "sliced and diced" into a set of records in a database. The database could be a native XML database, which might require considerably less slicing and dicing,

or it could be a traditional relational database, albeit modernized with XML features such as the ability to store XML fragments in character fields or blobs, and the ability to execute XQuery or XML-aware SQL over XML content. A smart application may update only the deltas to the Rate Table, but it is often more straightforward and secure to do a wholesale creation of a second table and a swap once the new table is created. While updates to the Rate Table might be relatively infrequent, the time and resources needed for the database and the update mechanism may still be very significant.

Billing events are received continuously, with each event represented by a single, typically very small XML document, the Rate Item. In some sectors, such as telecommunications, peak loads of thousands of billing events per second do occur. The charge must be established in real time, which requires look-up of the rate associated with the conditions described in the billing event. Our traditional XML application accomplishes this look-up by parsing the Rate Item into a DOM or SAX structure, extracting the parameters related to the rate from that structure, and using those parameters to form a query executed against the Rate Table. This query will then select a record from the Rate Table. It may be necessary to again parse an XML string field returned from the database (depending on how the slice-and-dice was done) and from the resulting structure to extract the desired rate information.

Rate Table Billing Example in RAX

Now let's see how RAX turns this problem on its side and solves it with no database, virtually no programming, and virtually no CPU time. While such a claim and such results are naturally quite surprising, it is really very common in computer science that problems intuitively approached by "brute force" methods can often be solved with orders of magnitude better performance by less obvious approaches. And, such approaches are often far simpler than the initial brute force approach.

Let's walk through the application step by step.

As shown in Figure 3 below, XPath expressions are generated from the Rate Table document such that there is one XPath expression corresponding to each item in the Rate Table, and each XPath expression selects only the corresponding XPath item. This is similar to the database requirement that each record have a unique key[3]. The XPath expressions may be generated with a script. The process of generating an XPath expression corresponding to each Rate Table item is far simpler than doing a slice-and-dice to the database. The structure of the expression corresponds closely to the XPath expression string (isomorphism), in contrast to very different structures that we must translate between XML and the relational table.

[3] In situations where uniqueness is not enforced, an XPath expression can also select multiple records and processing can be modified accordingly.

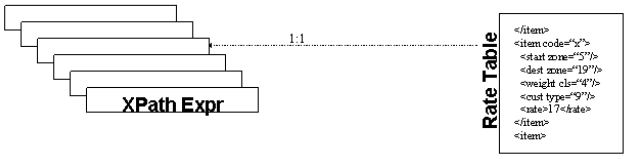


Figure 3—RAX Example: Generation of XPaths from Rate Table

The XPath expressions are then evaluated *simultaneously* to select the rate field in each record (with a very slight modification to select the rate text at the end of each XPath expression), generating the Rate List, as shown in Figure 4 below.

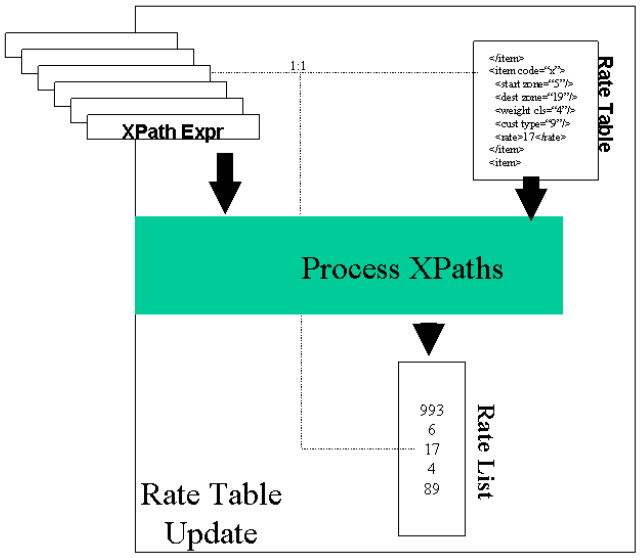


Figure 4—RAX Example: Generation of the Rate List from the Rate Table Using Generated XPaths

Individual Rate Items are isomorphic to items in the Rate Table. The same set of XPaths used to generate the Rate List may be evaluated simultaneously against each Rate Item as it is received. Each XPath produces a truth-value indicating whether it is matched or not. The uniqueness constraint guarantees that one and only one XPath will have a true truth-value. The position of the matching

XPath in the XPath set provides us with an index to the corresponding rate in the Rate List. We have successfully established the rate corresponding to rate event conditions. Continuous Rate Item processing is illustrated in Figure 5 below.

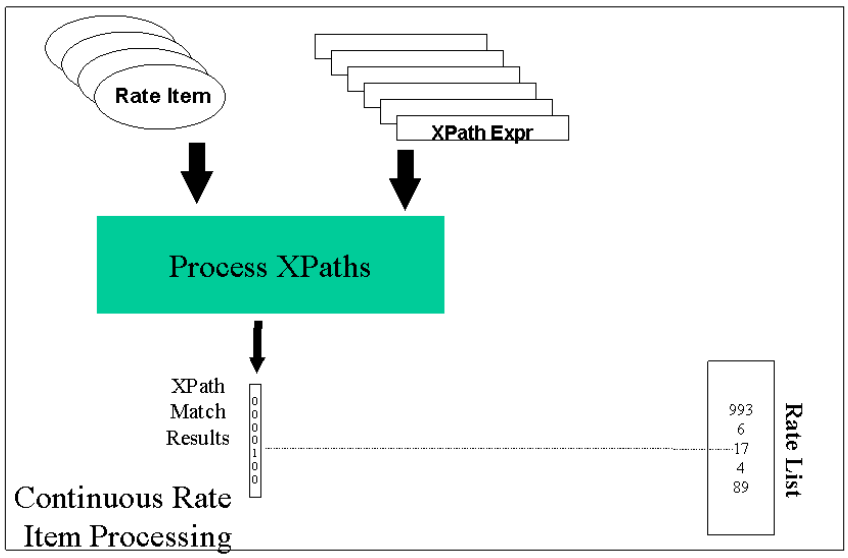


Figure 5—RAX Example: *Continuous Rate Item Processing*

The key enabling feature of the RAX approach is that, because the set of XPath expressions is equivalent to the entire set of XQuery expressions that could be generated, Tarari's XPath engine tests all records in the Rate Table simultaneously. The unique matching record that corresponds to the parameters in the current Rate Item is selected directly. Furthermore, whereas the traditional XML application needs to create a database and store all the selection criteria, RAX simply embeds these criteria in the XPath expressions that select the matching rate.

Our new architecture performs rate look-ups at line speed and allows us to eliminate:

- CPU bottlenecks
- network I/O
- need for a database server
- need for additional servers to load-balance Rate Item processing
- server overhead in Rate Table updates
- programming work
- points of potential failure

RAX-based Declarative Programming

The previous example provided an excellent illustration of how RAX can simplify XML software development by replacing complex procedural logic with XPath statements which *declare* what content needs to accessed or what assertions need to be tested. In Rate Table Billing we were able to show how in this particular case almost the entire application could be replaced by XPath expressions. RAX may be compared with XSLT, the most popular XML transformation technology. Transformations in XSLT are expressed in a primarily declarative, “stylesheet” language. An XSLT stylesheet declares how the final, transformed document is composed and how components from the input source are used in the output. XSLT uses XPath to identify these components selected from the input source.

RAX expands the applicability of the declarative programming approach beyond transformation to any application which uses XML in its business logic. XSLT is a great idea from the point of view of the convenience to the application developer, but its performance is poor. RAX, on the other hand, offers the advantages of

declarative programming and also far better performance than any other approach to processing XML.

Example - Content-Based-Routing (CBR) Applications

Content-based-routing (CBR) is done in a similar way: Messages are sorted in classification categories based on a series of content tests couched as assertions. In fact, the family of assertion-based XML applications is much larger than this. Jeff Ryan of Hartford Financial Services, in the developer.com article "XPath Rules!"[4] describes an XPath-based, general-purpose rule engine which can be used in a wide variety of XML application scenarios.

Content-Based Routing (CBR) in RAX vs. Traditional Software

Let's look now at an application which is traditionally XPath-based even when implemented in traditional software: content-based routing (CBR) and message classification. Because both the RAX approach and the traditional software tools use XPath to solve this problem, we can do a direct, apples-to-apples comparison of their performance.

[4] Ryan, J., "XPath Rules!", developer.com, www.developer.com/xml/article.php/3111191

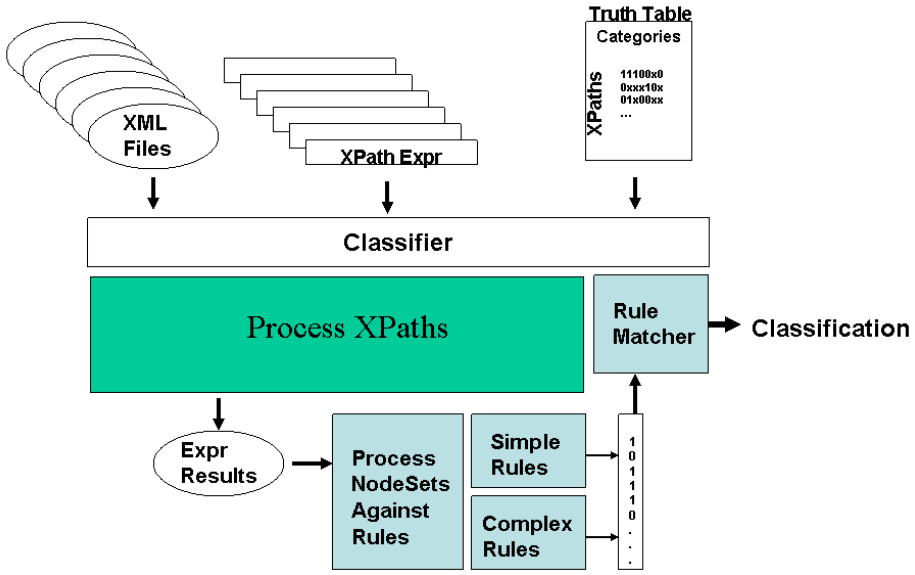


Figure 6—Content-Based Routing

A set of XPaths is written that will provide enough information to classify or route each message that comes into the system. Each XPath expression may be viewed as an assertion against the document; the assertion must ultimately resolve to a Boolean truth-value of either true or false. Simple evaluation of the XPath may be enough to establish its truth-value; for example, the truth-value could be based on whether the XPath returns 0 matched nodes (false) or more than 0 matched nodes (true). Other XPaths may require more complex evaluation, with XPath being further tested against business rules before its truth-value is established.

A classification for a given input document is obtained from the truth-values of all XPaths run against it. The bit value (0=false, 1=true, x=don't care) for the XPath result for each classification category is put into the Truth Table. As shown in Figure 4, above, there is one row for each

XPath in the classification set Truth Table. The bit pattern corresponding to a category or classification is read down each column.

Each time a file is run through the system, each XPath in the classification set is run against it, producing Expression Results. The Expression Results are then processed, with Simple Rule processing deriving the truth-value directly from the node count for the XPath's result set and Complex Rule processing deriving the truth-value from business rules which have the node set as input. A truth-value for each XPath is formed from the rule processing, and the set of truth-values is compared to the Truth Table by the Rule Matcher to obtain the classification of the original XML document.

Performance Benchmarks

Tarari created the classification application described above, implementing it using both RAX Simultaneous XPath and the most widely available, open-source software XPath APIs.

The setup of the RAX version and the software versions was functionally identical on all but one point: all traditional XPath processors require the association between namespace prefixes and namespace URIs to be provided to the processor in order to correctly handle documents that use standard XML namespacing. In effect, this limitation of traditional XPath processing requires that the entire document be scanned and a table of associations be constructed between prefixes and namespace URIs before XPath processing can take place.

Tarari's implementation of RAX is not encumbered by this limitation because namespaces are handled correctly as the document is processed.

Our tests were also run on a number of different hardware platforms and on single and multiple CPU configurations. While the classification application was set up as a self-contained application in order to isolate the performance of XPath processing, our results have been shown to be consistent in network applications. The setup of the benchmark produced very high throughput rates for both Tarari and for the software XPaths; however, both approaches were equally optimized, and we believe that they represent conditions which favor the software approach due to the namespace prefix problem described earlier.

The results in Table 1 are for a set of 63 classification XPaths run against thousands of XML messages ranging in size from 3Kb to 13Kb, with an average size of 8.8Kb. Messages were sorted into 34 different categories. The performance of Saxon is shown for the software comparison. In our tests, Saxon outclasses most other software XPath processors available and is arguably one of the fastest software XPath implementations outright.

	IBM SMP Pentium Msgs/sec	**Xeon P4 Single CPU, 2.4 GHz Msgs/sec**
RAX (Tarari)	5118	2652
Traditional XPath Software (Saxon)	48	67

Table 1--Test Results, RAX vs. Software

There's More to RAX

RAX is a comprehensive approach to ultra-fast XML message processing. In addition to the Simultaneous XPath engine, at the core of RAX are APIs which support:

- Access to the XML content either through XPath-generated indices or by ordinary streaming traversal
- Streaming transformation of the XML document
- Interface to DOM and SAX parsing models for XML fragments (with namespacing)

In addition, other Tarari technology is optionally available which may also be incorporated in high-speed XML processing applications:

- Acceleration of cryptographic libraries including JCE, used in most Java-based XML Security toolkits, and OpenSSL, commonly used in SSL for Web Services and other XML Internet applications
- Compression and decompression, to reduce network transmission time of verbose XML messages
- Character conversion, between UTF-16 and UTF-8 encodings

For More Information

To begin exploring how Tarari's accelerated XML processing can address your organization's business needs, please visit our Web site: www.tarari.com or contact us at info@tarari.com.

Legal Information

Tarari is a trademark or registered trademark of Tarari, Inc. or its subsidiaries in the United States and other countries.

Information in this document is provided in connection with Tarari products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Tarari's Terms and Conditions of Sale for such products, Tarari assumes no liability whatsoever, and Tarari disclaims any express or implied warranty, relating to sale and/or use of Tarari products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright, or other intellectual property right. Tarari products are not intended for use in medical, life-saving, or life sustaining applications. Tarari may make changes to specifications and product descriptions at any time, without notice.

* Other names and brands may be claimed as the property of others.

** Performance tests and ratings are measured using specific computer systems and/or components, and reflect the approximate performance of Tarari products as measured by those tests. Any difference in system hardware or software design or configuration can affect actual performance. Buyers should consult other sources of information to evaluate the performance of components they are considering purchasing. For more information on performance tests, and on the performance of Tarari products, contact us as indicated below.

RAX

Random Access XML

Fundamentally Changing How XML is Used and Processed

A Tarari White Paper

Additional information: info@tarari.com

Internet: www.tarari.com

Telephone: (858) 385-5131

Fax: (858) 385-5129

Tarari, Inc.

10908 Technology Place

San Diego, CA 92127-1874

USA

Tarari™

The Acceleration Company