



WRITTEN BY TOM YOHE

# Caching for XML Performance

## Delegating optimization throughout your Web services infrastructure

**H**ighly scalable implementations of service-oriented architectures (SOAs) always include heavy doses of caching. A guided tour through the SOA tiers, describing the caching and XML acceleration techniques employed along the way, provides the SOA enterprise architect with an awareness of optimization possibilities applicable to a Web service infrastructure. Consolidating the acceleration functions in an integrated appliance and controlling them via policies specified by WSDL annotations simplifies the implementation.

### AUTHOR BIO

Tom Yohe is the VP of Engineering and co-founder of Stampede Technologies, and currently leads one of the world's most elite enterprise optimization engineering teams, developing Web Acceleration Appliances for a broad range of Fortune 500 companies. Tom has been delivering award-winning enterprise products for over 25 years and has been granted numerous patents for unique data communications optimization techniques. Tom has a computer science degree from Pennsylvania State University.

Caching is a fundamental optimization technique found in all computer systems and networks. Modern CPU chips maintain instruction and data caches along with clever methods of ensuring coherency with the RAM that may be simultaneously accessed by other CPUs and I/O processors. In many respects, RAM can be thought of as a cache of your virtual memory swap file. Our networked world is full of caching subsystems, each designed to minimize the latency required to access correct content. As the computing paradigm has evolved to multitiered SOAs, caching continues to play a crucial role.

A typical SOA solution is implemented in a multitiered fashion for many reasons, not the least of which is performance. Figure 1 illustrates the

“nearest tier” through the “farthest” tier – the nearest tier is closest to the Web service consumer that issued the request; the farthest tier is the central database engine.

The computing devices and software that drive the farther tiers are generally more expensive compared with the subsystems in the nearer tiers. The processing that takes place in the farther tier subsystems tends to be “business-logic,” is often executed in higher-level interpretive languages, and generally takes more time and resources to execute. The logic of subsystems at the nearer tiers is usually very system software-oriented, is executed in the native machine language if not in ASICs, and tends to be highly efficient. The subsystems in all tiers are usually designed for high scalability, where simply adding more like subsystems can expand capacity. In front of each subsystem, tier load balancers are employed to intelligently deliver object requests and route responses between the tiers.

The message communication between each tier requires TCP/IP routing, load balancing, parsing, and other processing, all of which adds further latency to the transaction response time. Therefore, the more work that can be completed in the nearer tiers, the more efficient and responsive the overall SOA implementation will be.

### A Caching Tour of the Tiers

Caching begins at the farthest

Enterprise Information System (EIS) tier and proceeds through the nearest tier. XML caching takes place at the nearer tiers.

### Caching at the Core

In the farthest EIS tier, most database engines such as DB2, MS SQL Server, and Oracle have a myriad of caching options designed to minimize disk I/O operations as much as possible while ensuring that transactions are always physically recorded. The guidelines for these products usually suggest that the database files should be stored on a file system where the caching services supplied by the native OS's file system are disabled. The rationale for this is that the caching services of the native OS can't possibly be as intelligent as what is performed by the database, so therefore any memory resources devoted to the native OS caching engine would be wasted. Furthermore, the database engines want to guarantee that updates are physically written to storage. Database engines also make extensive use of “SQL Statement Caching,” which reduces the costly parse processing on the database server. While these database-caching techniques may be very exciting for system and database administrators, they are likely only of passing interest to the SOA enterprise architect.

### Moving Nearer Outward

As we move inward to the applica-

tion servers or business tier, the application designer often carries the burden of integrating caching optimization. In this tier we find two basic flavors of server platforms, J2EE and Microsoft ASP.NET. Each of the two environments offers the application developer a means for injecting caching to optimize performance and increase scalability. Some of the caching techniques employed in the business tier directly relate to Web services, while others optimize the business logic implementation of the Web service. The SOA enterprise architect should have a full understanding of all caching techniques utilized in this tier.

In the business tier, there are two important caching objectives: storing business data in memory to be readily accessed by application servers, and controlling caching strategies by setting related HTTP headers through platform APIs.

## J2EE Caching Techniques

In the J2EE environment, the caching of business data is achieved by various Object-Relational mapping technologies. The dominant choice offered by J2EE vendors is the Enterprise Java Bean (EJB). The Entity Bean developer often implements transactional caching of dynamic data by adhering to the Container Managed Persistence (CMP) model that provides caching as an inherent attribute. Some J2EE platforms such as IBM WebSphere offer extensions to CMP that allow data caching of static items across transactions. This extended functionality can be easily integrated into an application by setting attributes in property sheets. EJB can be a bit heavy-handed targeting large, complex systems. Some “lighter” technologies include both standards-based model abstraction such as Java Data Objects (JDO) and open-source offerings such as Hibernate.

The “JCache” specification defines a powerful set of Java-caching APIs that support implementations of caching frameworks. Both open-source and commercial offerings of caching frameworks allow cached objects to be shared among multiple applications. These caching frameworks are often an important component in enterprises where “single sign-on” to multiple applications is required. JCache provides methods to control temporary storage and expiration of Java objects in common memory in a manner that ensures

that the cached objects are coherent with the data stored in the farther tiers.

For controlling caching-employed HTTP servers and proxies, the J2EE specification defines the *HttpServletResponse* class, a component of the servlet container, as a fundamental J2EE building block. This class provides control of HTTP-specific functionality by furnishing methods to access and generate HTTP headers and cookies. Specifically, the *addHeader* method allows the HTTP 1.1 “cache-control” and “expiration” directives to be emitted to facilitate caching in the nearer tiers.

## The Microsoft Way

Instead of mapping business data into a constellation of objects, Microsoft employs a different strategy of caching transactional data. Data elements are presented as relational tables in memory, which is very similar to how they are persisted on disk. Those in-memory tables can be accessed via the *rowset* API, and maintained via either *DataReader* or *DataSet* APIs by ADO.NET.

ASP.NET developers have at least three types of caching capabilities available to them. “Output caching” informs ASP.NET that responses built for a page can be returned on subsequent requests for the same page without re-executing the ASPX script. Either an entire page can be added to the output cache or to a specified ASP.NET user control. Although the ASP.NET designer can embed an *HttpCachePolicy* class into their objects

so that standard HTTP caching proxies can interpret HTTP headers, this feature is of limited value because most standard HTTP proxy servers are not designed to deal with responses associated with HTTP “POST” commands. Another powerful caching method of ASP.NET is “application caching,” which allows the application to use the cache to store any data by leveraging the *System.Web.Caching.Cache* class.

## Web Servers and HTTP Caching Proxies

Much of the Web services infrastructure is dependent on a highly scalable Web server tier. Microsoft’s .NET framework for Web services depends largely upon the Web services functionality that is built into “Internet Information Server” (IIS) and its accompanying Microsoft Internet Security and Acceleration Server (ISA). J2EE-oriented infrastructures are more likely to leverage the J2EE-based Web/application servers, such as Apache Tomcat, embedded with the Axis SOAP handlers, as a common critical component. In an SOA implementation, caching proxy servers perform front-end processing offload and can sometimes provide layer-7 load balancing to the Web servers.

Traditional Web-caching proxy servers were designed to optimize the viewing of HTML Web pages by a browser. With HTML, even individualized page views share a similar layout, thus a large number of common components and many page views will be

“Although the ASP.NET designer can embed an *HttpCachePolicy* class into their objects so that standard HTTP caching proxies can interpret HTTP headers, this feature is of limited value because most standard HTTP proxy servers are not designed to deal with responses associated with HTTP ‘POST’ commands”

absolutely identical. This led to the establishment of a well-defined set of caching rules spelled out in the HTTP 1.1 specification for caching frequently requested pages and page components.

In an SOA, though the need for caching might be even greater, the means for caching in the Web tier is less defined. Unlike HTML, XML supports many different types of messages determined by the set of applications supported by the enterprise network. XML messages are most often transactional in nature – reflecting the dynamic nature of a conversation, negotiation, or exchange between parties. With XML traffic, there is less use of the common components that can be effectively cached. One of the caching rules set forth in the aforementioned HTTP 1.1 specification is that responses to HTTP POST messages are not cacheable, unless the response includes appropriate “Cache-Control” or “Expires” header fields. Unfortunately, most SOAP toolkits are designed to embed SOAP requests inside an HTTP POST message. Therefore, since traditional HTTP proxy servers make the assumption that responses to an HTTP POST are not cacheable, they provide no benefit for an SOA implementation.

An intelligent, “SOAP-aware” Web-caching tier can provide many important optimizations to an SOA implementation. Advanced caching engines *can* observe the “Cache-Control” and “Expiration” directives that *can* be set in the farther tiers. They also provide the ability to observe

“WSDL annotations provide a powerful and extensible means for an SOA enterprise architect to centrally specify the internal policies needed to build a highly scalable, secure SOA infrastructure”

caching policy directives defined by the SOA enterprise architect.

**Near Tier XML Caching and Optimization**

Traditionally, XML processing was entirely handled at the business tier. The now mature XML-related standards and the desire for higher performance have resulted in the capability to perform many of these functions in the nearer tiers.

The nearest tier is what I refer to as the XML “front gate” – the subsystem that first receives and parses the SOAP message.

Usually, the first task to be performed in a highly scalable SOA implementation is content-based routing of incoming SOAP messages. This technique uses knowledge of traffic and caching for message classification in which a series of rules or tests are cached by a server component that provides useful information about each message as it is processed. These tests are used to determine whether the message type is accepted by the receiving system and has an associated cached XML schema. Dropping unsupported message types saves processing time. This XML content-aware classification is the basis for routing and load balancing SOAP messages to different servers based on various characteristics of the messages. Classification is an important ingredient in efficient protection against malicious content, including denial-of-service attacks. XML rules or tests are most frequently expressed as a series of XPath statements, a W3C-standardized language for finding any XML content in a message and making assertions against that content. Similar in many ways to the SQL statement caching performed at the EIS tier, a sophisticated SOA near-tier component will pre-compile and cache these XPath statements for the most rapid possible evaluation against incoming XML messages.

A properly classified SOAP message should then have its schema validated. While Web browsers can deal with malformed HTML content by simply ignoring the unrecognizable content, the same is seldom true of XML applications. It is critical for network secu-

SOA Function	Standard Library Throughput	RAX-CP Throughput	Improvement Factor	Comment
Schema Validation – Large Document Set	52 Megabits per second	1,340 Megabits per second	25x	Xerces was used as the standard library. The test schema validation was run with 45 different large SOAP messages, with an average size of about 280K.
Schema Validation – Small Document Set	3,760 validations per second	34,892 validations per second	near 10x	Xerces was used as the standard library. The test schema validation was run on 131 different small SOAP messages, with an average size of about 2K.
Content-Based Routing	132 message routes per second	10,012 message routes per second	76x	Saxon was used as the standard library. Router observed 63 rules and 42 destinations.

Table 1 • Summary of the throughput improvement that can be achieved by effectively applying the RAX-CP to SOA functions

Attribute	Default Value	Description
<b>Tcs.cacheable</b>	No	Response is cacheable by a trusted caching server
<b>Tcs.lifetime</b>	30	Specifies how long in seconds the trusted caching server should be considered fresh after storing the object in its cache
<b>Tcs.private</b>	No	Although the response is cacheable the trusted caching server must be stored in a consumer context cache
<b>Tcs.invalidationOperations</b>	None	A list of SOAP operations that would immediately cause the trusted caching server to invalidate the cached response, regardless of lifetime
<b>Tcs.loadSchema</b>	Yes	The trusted caching server should load and cache this schema
<b>Tcs.allowSchemaImport</b>	No	The trusted caching server should load and cache schemas from this location
<b>Tcs.validateRequest</b>	Yes	The trusted caching server should validate requests against a cached schema
<b>Tcs.validateResponse</b>	Yes	The trusted caching server should validate responses against a cached schema

**Table 2** • Examples of caching and XML acceleration attributes

ity and performance to ensure that each message is correctly composed and can be safely handled by the receiving applications in the farther tiers. XML message types are described in a formal grammar known as an XML schema. This grammar, processed against the individual message, is used by an XML validator to ensure that the message can be correctly and safely handled. XML schema validation is a CPU-intensive operation, consuming as much as 30 percent of the overall time spent processing each XML message. When XML Schema validation is performed on a server in the nearer tier, the computing resources of the farther tiers are significantly off-loaded. XML schema validation should also be performed on outbound responses. This ensures that dynamically composed XML messages do not compromise unprotected Web service intermediaries and that sent messages conform to content-level agreements between partners.

Caching is important to smooth schema validation processing. While XML may describe an infinite number of message types, within any given application environment there is a limited number of types of messages that can be handled. Therefore, an intelligent SOA-processing component will cache the XML schemas for those messages that an SOA implementation will handle. The cached schemas are precompiled into a binary data structure to increase the execution speed of validation. In most validation systems, the XML message must be fully parsed before it is

validated, but a more sophisticated validator may be able to scan the message to determine whether the message type is supported before invoking full validation. Messages that are not supported by the SOA implementation can be discarded at the least possible cost.

In order to meet the heavy computational requirements of schema validation and content-based routing, using a device that can perform these computations in ASIC hardware is a recommended practice. One such device is the Tarari RAX Content Processor (RAX-CP). Table 1 summarizes the tremendous throughput improvement that can be achieved by effectively applying the RAX-CP to these important SOA functions.

### WSDL Annotations – The New Sheriff in Town

In an SOA, caching optimizations can be provided by many SOAP nodes, including service providers, clients, and intermediaries. Caching is not even limited to HTTP traffic since SOAP is not bound to any specific transport. A generic, transport-independent caching control mechanism is needed.

The common denominator for setting policies that can be observed throughout an SOA implementation is the Web Service Descriptor Language (WSDL). The primary objective of WSDL is to specify the operations and interfaces so that a consumer can employ a Web service without any knowledge of its underlying implementation. The WSDL

is like the contract between the Web services consumer and provider. The designers of the WSDL message format had the foresight to allow seemingly innocuous annotations to be added to a SOAP interface. To external consumers, these annotations are meaningless and therefore ignored, but inside a Web services-based infrastructure, these annotations can provide instructions to trusted components running at different tiers. These annotations can advertise reliability capabilities and transactional properties, as well as provide caching and other optimization directives.

WSDL annotations provide a powerful and extensible means for an SOA enterprise architect to centrally specify the internal policies

**“An intelligent, ‘SOAP-aware’ Web-caching tier can provide many important optimizations to an SOA implementation”**

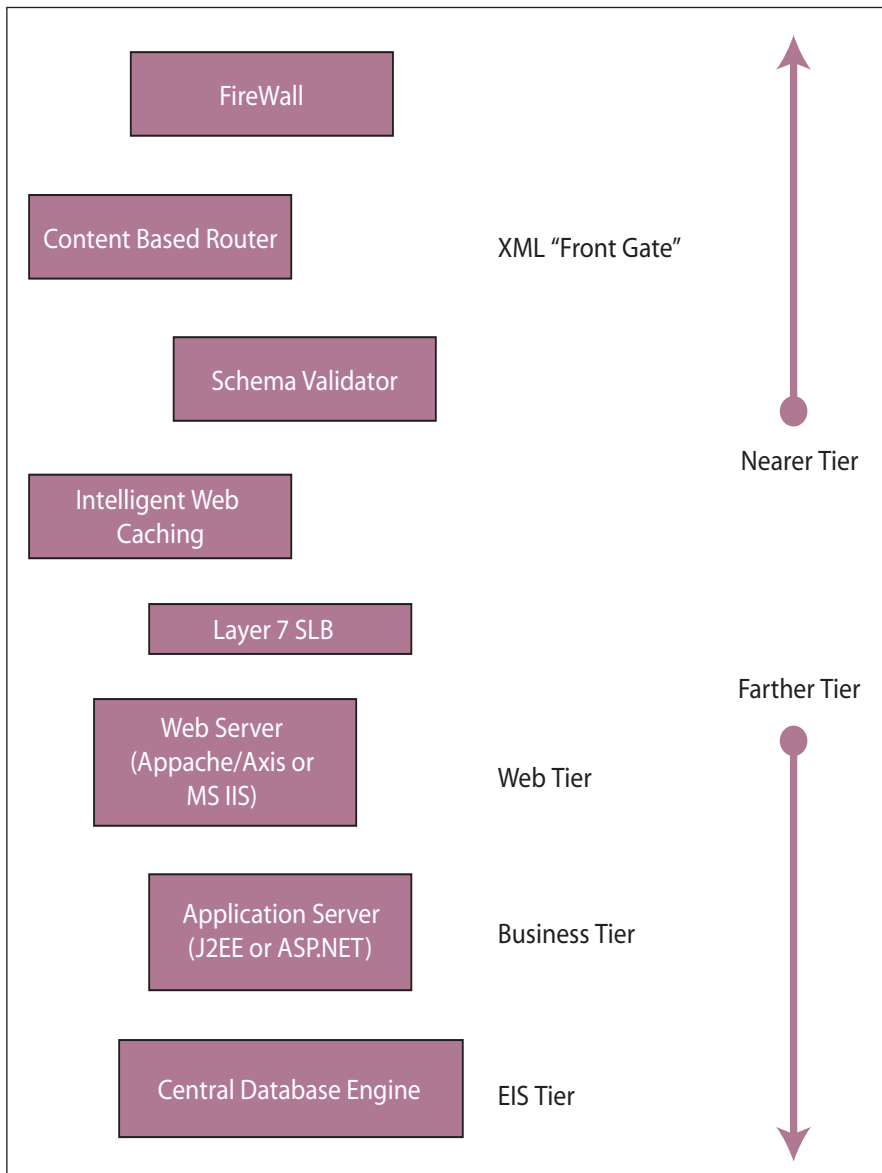


Figure 1 • Multitiered SOA

needed to build a highly scalable, secure SOA infrastructure.

### Consolidating-Near Tier Caching Functionality

Deployment of Web service-caching techniques is greatly simplified when multiple functions can be performed in a single appliance. An appliance that consolidates the XML "front gate" functionality of the XML content-based router and schema validation with the advanced HTTP Web-caching engine that is necessary for caching SOAP responses works best. An intuitive browser-based management interface to control the powerful XML schema validation and content-based routing features of an integrated

RAX/CP covers all bases. XML caching and optimization should be easy to configure by annotating enterprise WSDLs to define policies that are consistent with your SOA design objectives. Table 2 presents some examples of caching and XML acceleration attributes that could be associated with the operations exported by a Web service from a good XML caching appliance.

### Conclusion

A senior business manager once taught me that the key to running a successful organization is to always push work down to the lowest level possible. Of course for this to work the manager must be able to trust that the delegated tasks will be

carried out competently. This philosophy also holds true in an SOA implementation. If possible, the SOA enterprise architect should delegate work down to the cache processing of the nearer tiers. However the SOA enterprise architect must ensure that the processing that takes place in the nearer tiers is securely performed without compromising the integrity of the databases of the farthest tier. By specifying XML content processing and SOAP-aware caching using WSDL annotations, the SOA enterprise architect can delegate this work to the nearer tiers in a trusted fashion.

### References

- "Dynamic Statements Cache: Getting the Most Out of It" by Namik Hrlle. *The IDUG Solutions Journal*, Volume 8, Number 2 (September 2001): [www.idug.org/idug/member/journal/Sept01/articl08.cfm](http://www.idug.org/idug/member/journal/Sept01/articl08.cfm)
- Using JCache to Save Money – Nigel Thomas: <http://java.sys-con.com/read/37551.htm>
- XML Web Service Caching Strategies – Matt Powell, Microsoft: <http://msdn.microsoft.com/library/en-us/dnservice/html/service04172002.asp>
- Container-managed persistence features: [http://publib.boulder.ibm.com/infocenter/wsdoc400/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/rdat\\_cmppers.html](http://publib.boulder.ibm.com/infocenter/wsdoc400/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/rdat_cmppers.html)
- Package javax.util.jcache documentation: [www.jdocs.com/jcache/1.0/api/javax/util/jcache/package-summary.html](http://jdocs.com/jcache/1.0/api/javax/util/jcache/package-summary.html)
- J2EE 1.4 API Specification: <http://java.sun.com/j2ee/1.4/docs/api/index.html>
- Microsoft Research: Caching of XML Web Services for Disconnected Operation, Venugopalan Ramasubramanian and Douglas B. Terry (December 2004): <http://research.microsoft.com/research/pubs/view.aspx?type=Technical%20Report&id=844>
- Tarari Random Access XML Content Processor: [www.tarari.com/rax/](http://www.tarari.com/rax/)
- *Random Access XML Programming Assisted with XML Hardware* by Michael Leventhal, XML 2004, November 2004, Washington, D.C.: [www.idealliance.org/proceedings/xml04/papers/299/RandomAccessXML](http://www.idealliance.org/proceedings/xml04/papers/299/RandomAccessXML)

tyohe@stampede.com