

## White Paper



### Performance Review of COTS-based System Architecture

---

Ashutosh Shinde

#### Abstract

Commercial, off-the-shelf (COTS) software has gained considerable popularity as an approach that quickly and economically creates software systems that address business requirements. The saved development time and money will be wasted if the performance evaluation of the system is deferred, potentially leading to performance issues in the system. To save on the costs of performance problem detection it is proposed to introduce a separate review of the architecture from the performance perspective, to identify and resolve any performance issues early on in the life cycle. A few areas that can be addressed as a part of the review activity are covered in this paper.

## Introduction

Software systems increasingly depend on integration of COTS components for quick and effective realization of business requirements. In this context, COTS refers to commercially available independent software that provide core business functionality. These can either be used independently or integrated with other COTS to provide a complete solution for a large system. For example, a web content management system and a search engine COTS, from different vendors, can be integrated into an e-commerce site to achieve the desired functionalities of content management and information search, while the stake holders remain abstracted from the details of the implementation. While such an approach helps reduce development costs, performance analysis often becomes difficult due to inaccessibility of the COTS internals, viz. software artifacts like architecture documents, design documents, source code etc. Inter-play between multiple COTS components also makes the performance analysis process difficult. Component-based performance modeling techniques [PUTRO2] have been proposed to manage the performance of such systems. It is proposed that performance engineering activities like modeling be augmented with a separate performance review of the architecture to identify and resolve any performance issues early on in the life cycle.

The paper details certain architecture and design aspects that can be reviewed from a performance perspective, to gain qualitative insight.

## Elements of the review exercise

Software engineering employs reviews at all lifecycle stages for timely verification of systems against requirements. System architecture reviews are more advantageous as they help identify problems early in the lifecycle. This facilitates informed design decision making [MARA05]. A performance expert can add a performance perspective to the architecture review exercise to address potential issues impacting the performance of the system.

As part of the performance review process, some of the architectural aspects that need to be considered are detailed below –

**Integration** – The process of integrating COTS components into a system imposes additional constraints on the architecture. The impedance mismatch introduced because of the differences in the messaging semantics, the communication protocols and the technology platforms has an impact on system performance. Likewise, the adapters and wrappers, created to abstract the integration process, induce a level of indirection in the communication layer and are likely to impact performance.

The focus areas during the review exercise should include:

- Integrating APIs
- Connection pooling
- Data synchronization
- Integration using web services
- Disruptions
- Secure communication

**Other areas** – Certain areas that have a relevance to the architecture definition stage and have a significant impact on the performance of the system are addressed in this section. These areas do not necessarily come under the purview of architectural reviews.

- Customizations
- Prototypes

Subsequent sections detail each of the areas mentioned above.

## Integration

### *Compatibility of integration APIs*

Many COTS components are shipped with APIs to facilitate smooth integration into the application eco-system. While the APIs facilitate the integration process, the usability of the APIs is dependent largely on the environment in which the components are expected to be hosted and the nature of the components that need to be integrated. For example, consider a COTS component that is shipped with C APIs, but is intended to be used by a Java component. Similarly, a COTS component that is shipped with Perl APIs, but is intended to be used by a Java component. It is evident that these components cannot be used “as-is” because of the inherent incompatibilities in the technologies. But there are technologies like Java Native Interface (for bridging Java with C and C++) and Active Perl modules (for bridging Java-Perl) that can be used to bridge incompatibilities. Though these technologies render the incompatible APIs useful, it is necessary to understand that these technologies add an extra layer to negotiate the call across incompatible components. This has a definite impact on the performance.

Such technologies have been enhanced to reduce the impact on performance; for example, calling simple native methods from Java code improved by 74% between JDK 1.3 and JDK 1.4 [SUN]. Prototypes must be created to identify the performance impact before accepting/ suggesting the suitability of these technologies in the solution.

Areas, within the architecture, that rely on such technologies must be marked for performance review.

### *Connection pooling capabilities*

Setting up connections between COTS components for remote collaboration can be a resource consuming activity. The process involves allocation of network and memory resources and occasionally authentication activities for setting the security context to the connection. Establishing a connection once and then re-using the same across multiple requests is an ideal mechanism to enhance performance of the system and reduce resource consumption. For example, Java Database Connectivity API (JDBC) is supported by major databases as a mechanism for establishing connections with the database. The connection instances that represent the physical connection between the database and the client can be pooled and re-used across different requests.

There is a tendency to associate connection pooling concept with middleware products, however, other products that require connections to be established can also use connection pooling capabilities. While connection pooling is a standard technique to optimize the use of resources and enhance performance, its proposed usage must be reviewed to identify performance issues. Some of the areas that need to be covered as a part of the review are –

- Some systems prevent client components from pooling connections in order to eliminate security threats associated with connections that are open for a long time. A very small ‘keep-alive’ time is associated with such connections which prevent connections from being pooled. Often, use of such pools is limited by business considerations rather than technical limitations. In such systems connections need to be opened and closed with the component for each interaction. Hence, it is necessary to find out if connections to the COTS components in the system under review can be pooled and re-used.
- At times COTS vendors provide APIs to facilitate smooth integration. In some cases, the APIs manage the connection opening and closing as well as the actual execution of the call to the component. If an API is provided by the COTS vendor, the review should investigate if connection pooling has been implemented as a part of the API. If not, it is worth analyzing if the API can be augmented externally to support a connection pool. In case a decision is made to augment the pool externally, the design should be flexible in accommodating the changes to the API that may happen in future COTS releases.
- The licensing strategy of certain products limits the number of connections that can be opened concurrently. Additional connections are allowed to be opened only with license upgrades. It is necessary to understand such limitations before deciding on COTS. As a general practice, the number of connections in the pool is set equal to the number of concurrent requests expected on the system so that wait time is minimized. Before deciding on the impact of the limitation the number of connections that can be opened should be judged against the expected concurrency on the system (existing as well as forecast).

## Data synchronization

COTS components typically work like isolated units, not sharing data with other elements in the system. However, there are situations when it is necessary to synchronize the data maintained by COTS with the rest of the system. Data synchronization is a technical challenge since COTS components are generally not designed to dissipate information to other elements in the system [EGYE04].

Data synchronization activities can be classified into real-time synchronization and deferred synchronization.

In real-time synchronization, data is immediately synchronized with other systems after changes have been effected in the primary COTS component. Techniques have been suggested [EGYE06] to augment COTS components externally to receive information about real-time changes that occur within components. However, the level of intrusiveness of these techniques and their impact on performance must be evaluated before adopting them. In real-time synchronization, the synchronization process shares resources with other processes that service business requirements. Hence, synchronization techniques must add minimum possible load on the processing bandwidth of the system so that business requests are processed optimally. The following areas must be covered while reviewing synchronization processes –

- **Change notification** - Event notification mechanisms to identify changes can be incorporated into custom built components. The same is not feasible in COTS components because of the absence of source code. The system has to rely on the pre-built capabilities of COTS components to disseminate change related information to other elements. However, such capabilities are very rare since COTS components are not designed to proactively disseminate information. Additional processing may be required to get necessary information.
- **Change identification** - The interfaces exposed by COTS components provide a convenient mechanism to extract information from the components, but the information that can be extracted is limited by the comprehensiveness of the interfaces. The techniques used to extract changed data must be reviewed; particularly when the interfaces expose limited information. Similarly, the technique used to segregate changed data from unchanged data also needs to be reviewed, since such a process can be computationally intensive depending on the amount of data to be matched.

In deferred synchronization, data is synchronized with other systems in an “off-line” mode. This technique is typically used in cases where data is synchronized for maintaining consistency between different elements in the system and/or for reconciliation between multiple elements in the system. In some cases business processes, in other elements of the system, are able to function with data delivered in a deferred mode. For example, refund processes in many systems are initiated as bulk jobs at the end of the day.

Deferred synchronization is typically achieved by extracting relevant information from the persistent store of the application (data files, database etc) and then synchronizing the data with other systems using scheduled batch jobs and programs.

The review process must cover the design of batch jobs and other programs to ensure they do not lock the data sources for unacceptably long periods of time and affect the performance of other processes.

## Web Services

Business owners and architects have accepted the simplicity of XML-based web services for the integration of COTS components that reside on different platforms to deliver a flexible system.

Web services are web-based applications that use XML-based open standard and transport protocols to exchange data with collaborating systems. Web services allow easy integration between a Microsoft .NET component and a J2EE component over the web. However, the flexibility and interoperability associated with web services come with a performance price, which needs to be evaluated in the performance review exercise.

Some of the questions that need to be answered as a part of the review are listed below –

- How coarse-grained is the web service API? A coarse-grained API reduces the number of requests a client has to make to get the desired information. This in turn reduces network traffic and enhances performance.
- How large is the SOAP message? A larger payload increases parsing time and the time consumed in delivering the message. The messages should be evaluated to eliminate redundant information to reduce the size of the message.
- In case the SOAP message is very large, can the message be compressed to reduce its size? While compression and decompression consume additional CPU resources, the technique is helpful in cases where bandwidth is a constraint

in meeting performance requirements. Typically SOAP requests are smaller in size than SOAP responses. Hence, only SOAP responses can be compressed with the help of SOAP extensions.

- How complex is the SOAP message? Intricate hierarchies in the XML structure can have a negative impact on the time spent in parsing the message.
- What techniques are used for parsing and creating a message? Which parsing techniques (SAX/DOM) are used? SAX parsers execute faster than DOM parsers.
- Is caching implemented for static message responses? Pre-defined (static) responses need not be created every time a call is initiated. The messages can be cached to enhance performance.
- Is security implemented as a part of the web service? Do all messages need to be secure? Secure web services introduce the overhead of establishing the credentials of the caller before responding to the request. Web service calls that are not intended to carry sensitive information need not be secured.
- Can an asynchronous messaging model be used in the situation? Asynchronous messages can increase throughput at the cost of introducing latency.
- Are transactional services warranted in the solution? Transaction services add the overhead of maintaining transaction context across calls and thus impact performance.

This list is indicative of the kind of questions that one needs to ask in the performance review exercise. More questions can be added depending on the application context.

## Disruptions

Businesses require systems to handle both planned and unplanned outages. Some systems continue to operate during partial outages, at times in a limited capacity. To maintain business continuity, it is necessary that the procedures that diagnose failures and attempt to adapt to conditions are reviewed properly. While these procedures may not be executed as part of the normal work flow, they are critical in case of a breakdown and can have a major role in reacting to the situation. Hence, such procedures need to be identified and reviewed for performance. Some examples of such procedures are –

- Re-connection modules that attempt to reconnect with failed COTS components
- Router algorithms that identify failures in components and route calls to other healthy peer components
- Alerting mechanisms that dissipate failure information to configured entities

Consider the following scenario where the system was designed to operate in breakdown conditions and maintain business continuity. However, it faced constraints in maintaining continuity because the part of the system that handled disruption was not reviewed.

An e-commerce site used the services of a commercial payment gateway to authorize credit card transactions. The site used the payment gateway to conduct credit checks on customers to reduce the risks of bad transactions. The payment gateway in-turn integrated with other COTS products which conducted the credit checks. The payment gateway distributed requests between these products based on a proprietary algorithm. The redundancy ensured that the payment gateway could guarantee business continuity even if some of the credit history agencies were not operational. During one of the testing cycles it was observed that the payment gateway experienced a very high CPU usage and substantial performance degradation when any one of the credit agency's COTS components was not operational or disconnected from the TCP/IP network. The degradation had a considerable impact on the business continuity as even healthy credit agency components which were operational could not be utilized for credit history checking.

On investigation, the problem was traced to a design issue in the connection module which attempted to re-establish the connection between the payment gateway and the failed credit agency COTS component. Blocked connection attempts by the connection module led to the high CPU usage, thus reducing the resources available for processing other transactions. The existing JDK 1.4 IO library was replaced with the NIO library. The NIO library offered non-blocking features and guaranteed time-outs on IO operations. This resulted in reduced CPU utilization in case of failures.

This problem should have been identified as a part of the performance review exercise.

Thus, special conditions that arise out of disruptions in COTS components and impact business continuity and business at large need to be reviewed for potential performance issues.

## Secure communication

An important aspect of system security is secure data communication between collaborating COTS components. This aspect of integration can have serious performance implications because of the resource intensiveness of most encryption and decryption algorithms. In some cases, like the Secure Socket Layer (SSL) which is commonly used for securing data over the TCP/IP network, the performance cost of a handshake is more than the cost associated with the encryption and decryption of data. The security aspects related to integration of COTS should hence be reviewed to identify any performance issues. Some of the questions that need to be addressed in the review are listed below -

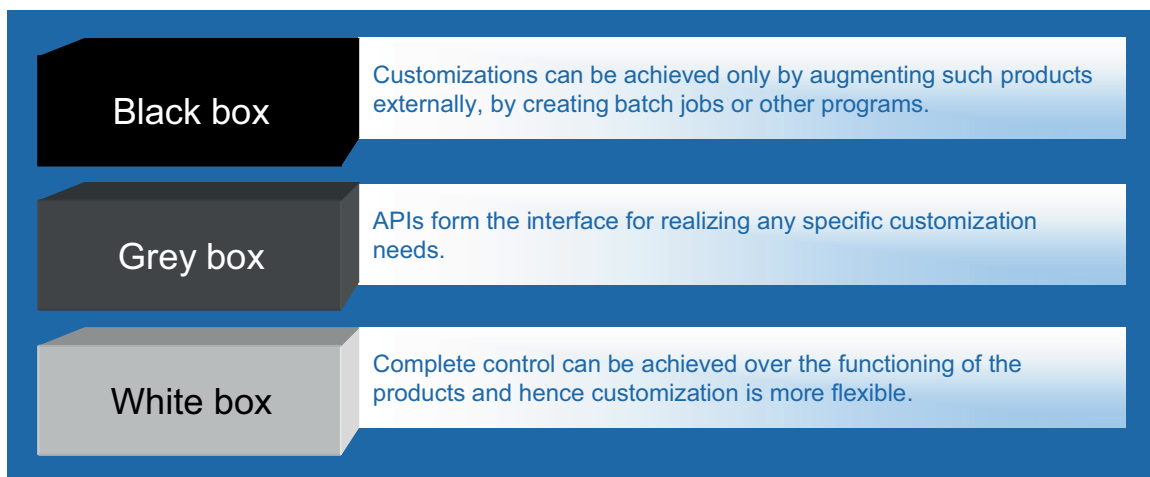
- Secure communication may be mandated only in some component interactions. Does the system switch to non-secure communication when data security over the network is not necessary?
- Can connections be kept alive once a communication link has been established to reduce the recurring cost of establishing security context?
- How many requests can be passed on to a live connection before the connection is dropped? Some systems limit the number of transactions that can be sent over a live connection to prevent denial-of-service attacks. What number of live connections is sufficient to handle the concurrency requirements of the system?
- Is the proposed cipher suite the best option in the given deployment environment and prevalent performance constraints? Are benchmarks available to take an informed decision? The performance of encryption and decryption is affected by the cipher suite used in the process.
- Can hardware accelerators be used in case the number of SSL handshakes is significant? Does the deployment environment support hardware accelerators?

More questions can be added to the list based on the application context.

## Other areas

### Customization

Unique business requirements necessitate customization of COTS products. COTS products are usually shipped in one of three forms- black box, grey box and white box [HAIN97]. The customization technique for each of these types is different -



In all the above cases, customization can be achieved with the help of configuration changes to the product. However, there are times when mere configuration changes are not sufficient to achieve customization in the business process. In such cases, amendments to the product are mandated.

Generally, vendors endorse the performance of their products in its original form, but are quick to disclaim the performance if any customization is implemented. Hence, it is necessary to review the proposed customizations from the performance perspective and ensure that any pitfalls are highlighted. It is also advisable to share the proposed major customizations with the vendor to solicit their feedback on the perceived performance implications of the change.

Some of the areas that need to be included in the review are:

- Aspects of the COTS product that are impacted due to customizations have an effect on the complexity of the customization. Isolated customization of the presentation, data or the control (logic) of the product is relatively less complicated compared to customizations that cut across these areas. Complex customizations that include changes to presentation, data and control aspects of the product must be reviewed.
- Information about the effect of similar customizations of the product in other installations can provide qualitative insight about the effect of customization on the performance. This information can be sought from the vendor but should only be used as a guideline after considering the similarity between the customizations and the deployment scenarios.

## Prototypes

At the architecture definition stage, prototypes are created to obtain a definitive interpretation of the proposed solution. A well thought out and properly implemented prototype can be a good source for gathering information on the feasibility, risks and general characteristics of the system. It can also be an excellent information source to understand the performance characteristics of the system. The information gathered from prototypes can be fed back into the architecture exercise to fill gaps and enhance the architecture.

Prototypes are created to validate the proposed technical solution, but the intent should be to analyze the performance behavior of the solution. Necessary amendments can be made to the architecture to address performance issues exposed by the prototype.

## Conclusion

Augmenting the architecture review exercise with a performance perspective will yield qualitative insight into the performance characteristics of the system. The insight will provide directions for subsequent software and performance engineering activities and enhance their efficacy in achieving the desired quality of service. Performance reviews are good mitigation strategies for reducing risks to the business caused due to poor performance issues.

This paper covered critical areas of COTS based systems which must be reviewed in such performance review exercises.

### About the Author

Ashutosh Shinde has worked with large clients over the past nine years in the areas of architecture evaluation and definition, framework conceptualization, technology evaluation and performance enhancements. He has worked in the development of OLTP-based enterprise products for Banking and Telecom, primarily on the J2EE platform.

## References

- [PUTR02] Erik Putrycz, Murray Woodside and Xiuping Wu, Performance Techniques for COTS Systems, 2002.
- [MARA05] Maranzano, J. E., Sandra A. Rozsypal, Gus H. Zimmerman, Guy W. Warnken, Patricia E. Wirth, David M. Weiss, Architecture Reviews: Practice and Experience, IEEE Software, vol. 22, no. 2, pp. 34-43, Mar/Apr, 2005.
- [EGYE04] Alexander Egyed, Sven Johann, and Robert Balzer, Data and State Synchronicity Problems While Integrating COTS Software into Systems
- [HAIN97] Gary Haines, David Carney, Component Based Software Development/ COTS Integration, Jan 1997
- [SUN] <http://java.sun.com/j2se/1.4.2/performance.guide.html>
- [EGYE06] Alexander Egyed and Robert Balzer, Integrating COTS Software into Systems through Instrumentation and Reasoning



For more information, contact [askus@infosys.com](mailto:askus@infosys.com)

#### About Infosys

Many of the world's most successful organizations rely on Infosys to deliver measurable business value. Infosys provides business consulting, technology, engineering and outsourcing services to help clients in over 30 countries build tomorrow's enterprise.

For more information about Infosys (NASDAQ:INFY), visit [www.infosys.com](http://www.infosys.com).