Yugoslav Journal of Operations Research
17 (2007), Number 2, 209-222
DOI: 10.2298/YUJOR0702209R

AN INTEGRATION STRATEGY FOR LARGE ENTERPRISES

Dejan RISIMIC
HSBC Bank
dejan_risimic@hsbc.ca

Received: November 2006 / Accepted: September 2007

Abstract: Integration is the process of enabling a communication between disparate software components. Integration has been the burning issue for large enterprises in the last twenty years, due to the fact that 70% of the development and deployment budget is spent on integrating complex and heterogeneous back-end and front-end IT systems. The need to integrate existing applications is to support newer, faster, more accurate business processes and to provide meaningful, consistent management information. Historically, integration started with the introduction of point-to-point approaches evolving into simpler hub-and-spoke topologies. These topologies were combined with custom remote procedure calls, distributed object technologies and message-oriented middleware (MOM), continued with enterprise application integration (EAI) and used an application server as a primary vehicle for integration. The current phase of the evolution is service-oriented architecture (SOA) combined with an enterprise service bus (ESB). Technical aspects of the comparison between the aforementioned technologies are analyzed and presented. The result of the study is the recommended integration strategy for large enterprises.

Keywords: Hub-and-Spoke, point-to-point integration, service-oriented architecture (SOA), enterprise service bus (ESB).

1. INTRODUCTION

Nowadays large enterprises are confronting new and significant challenges including: customer expectations for more comprehensive and timely access to markets and information, growth in business volumes, and changes to service delivery models [7]. The approach to the management and use of Information Technologies is critical to successfully

meet IT's expanding responsibilities, and to satisfy its customers increasing needs. The industry is moving from a product model to a customer-focused business model. Information technology is seen as an enabler for the organization to restructure operations and re-think service delivery mechanisms. These changes are necessary so the company can work better, reduce costs, and become more customer focused. However, the present technical infrastructure imposes a variety of constraints on a company's ability to improve the way it does business.

The existing IT architecture is the result of years of evolution as the business has changed and grown and as new systems have been deployed. Historically, architectural decisions have been made largely on a per application basis, with limited consideration for enterprise application and data integration issues. The present IT architecture largely reflects the capabilities and application system designs of key business systems developed over the past ten to twenty years. Although these systems served the organization well in the past, they have tended to be difficult to modify as business requirements change. It is also extremely expensive to expand them to provide additional required features and functions. There are many complex interdependencies between systems. The technical environment is "brittle"; often there is fear that making a change will cause something somewhere else to break. The fact that multiple processes exist for similar functions makes environments even more complex.

Integration appears in several different ways:

1.1. People

These are the most common means of integrating interactions between humans and those from machines to humans. Employees integrate their work typically through interaction, while business partners integrate through participation in horizontal processes. Employees and business partners are important parts of integration, but very often humans represent the bottleneck in business process interactions. Common practice for solving these types of problems is through elimination of human interactions, by introducing a process called Straight-Through-Processing—STP.

1.2. Process

Repeatable elements on the business and IT side make good candidates for sharing across the organization. Business processes such as Account Open and Customer Info are examples of horizontal services that can be decoupled from the particular application. Security and Monitoring are further examples of the services appropriate for decoupling and reuse. Precisely decoupling the processes and services enables business to respond rapidly to market changes.

1.3. Applications

One very common means of achieving reuse is through integration of already existing functionality with a new function, one either custom built or vendor supplied. The application integration goal is to reuse, rather than to replace. Therefore providing access to

the existing assets, routing and transforming the data, becomes the primary target for the application integration.

1.4. Systems

The primary role of systems is to manage process, and route and transform the data across applications.

1.5. Data

The data are the most critical assets for every organization and primary sources of information. As such they are highly required to be secured and shared amongst applications.

At present marketplace there is increasing attention to cost effective solutions. To continue with current architectural practices will continue to increase the costs of new offerings and functionality, and will drive up the costs of general support and maintenance.

2. TRADITIONAL APPROACH

At the conceptual level, the current view of large enterprise system environments consists largely of two tier applications that rely on “point-to-point” integration between application front-ends and back-ends and, in some cases, back-ends to back-ends.

2.1. Point-to-point integration

The existing IT architecture is the result of years of evolution as business has changed and grown and as new systems have been deployed. Historically, application interfaces were developed largely on a per application basis, with limited consideration for enterprise application and data integration issues. While interfaces of this type can be built and deployed relatively cheaply and rapidly, it is often extremely expensive to expand them to provide connectivity to additional systems. This predominance of tightly-coupled point-to-point interfaces between applications is largely to blame for many of the cost inefficiencies in today’s enterprises.

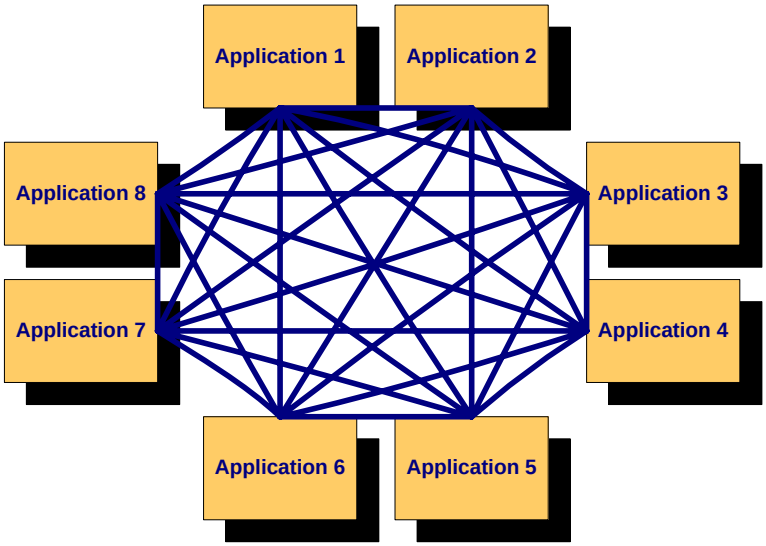

Figure I: Point-to-point integration topology

The topology from Figure I is also known as “islands of automation.” This occurs when the value of an individual system is not maximized due to partial or full isolation. If integration is applied without following a structured approach, point-to-point connections grow across an organization. Dependencies are added on an impromptu basis, resulting in a tangled mess that is difficult to maintain. This is commonly referred to as spaghetti, an allusion to the programming equivalent of spaghetti code. For example: The number of connections needed to have a fully-meshed point-to-point topology with *n* applications is given by: $n*(n-1)/2$. Thus for 8 applications to be fully integrated, 28 individual connections are required.

2.2. Traditional[1] Hub-and-Spoke

On the other hand, a traditional hub-and-spoke integration topology consists of an integration hub and many spoke systems connected as depicted in Figure II below.

[1] Some authorities consider ESB being a hub-and-spoke.

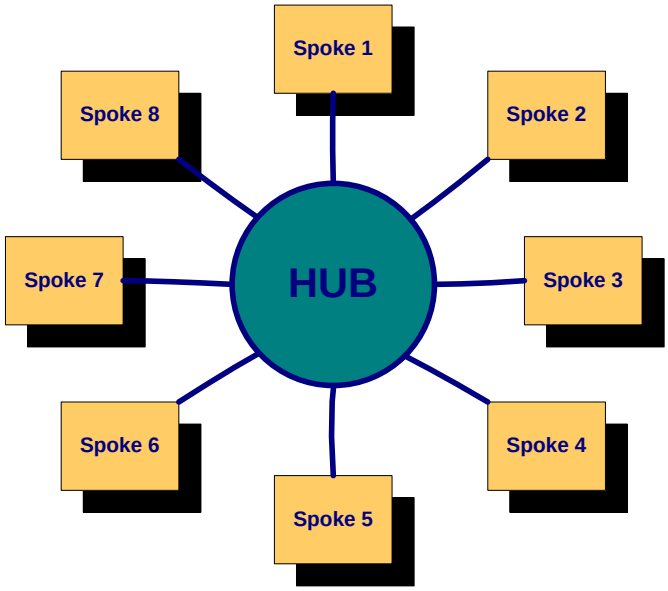

Figure II: Hub-and-spoke integration topology

An integration hub typically contains:

- a generic business object model, that deals with conflicts, ambiguities, and compatibility issues between the applications,
- a transformation engine that maps all application specific business objects to the generic form and vice versa during the integration process, and,
- a collaboration engine that executes any business process logic that is part of the integration process.

A typical hub-and-spoke integration architecture has as many connections as it has integrating applications. Thus for 8 applications to be fully integrated, 8 connections are required. Whenever a new application needs to be integrated only a single new connection needs to be added in such a topology.

2.3. Distributed Technologies

Once networked computers proliferated, and started being widely adopted, it became necessary to share resources and data amongst them in a cohesive manner [4]. The first attempt to achieve integration between physical systems of disparate technology was with the "Remote Procedure Call." This approach has become widespread with the rise of distributed computing. RPC was one of the first approaches designed to be as similar to making local procedure calls as possible.

The emergence of distributed computing required the support of new N-Tier system architectures, which in turn required more sophisticated ways to integrate different tiers within a system, in order to promote the concept of reuse. Therefore several different technologies, based upon distributed objects [15], appeared: CORBA, RMI and DCOM/COM+.

Both technologies, RPC and Distributed Objects, are based on a concept known as "tightly coupled interfaces." In tightly coupled environments each involved system has to know details of the systems with which it wants to communicate. Over time, as the number of systems and interfaces increases, it rapidly becomes very cumbersome to maintain. In addition, all these technologies suffer from an inability to communicate natively with each other, therefore causing locking within some technology boundaries. The other big problem is that systems become tightly coupled, having business processes embedded within the business logic of components. Changing or re-engineering business processes, implemented this way, is expensive and is a time consuming endeavour that seriously affects time to market and cost.

The most successful of all point-to-point integration technologies was Web Services. Web Services leveraged the well-known concept of loosely-coupled interfaces, previously utilized in Message-Oriented Middleware, commonly known as MOM. Essentially "Web Services" is a well-formalized loosely-coupled point-to-point integration technology. The basic concept of loosely-coupled interfaces relies on a communication channel that can carry self-contained units of information—i.e., messages. Messages may be in either text or binary form. Using messages for information exchange makes systems abstractly decoupled; therefore systems do not need to know details of each other.

Message-Oriented Middleware provides the abstraction of a messaging queue that is reachable from the network [1]. MOM does not model messages as method calls; it is based rather on sets of messages. Clients send and receive messages via the MOM-provided set of APIs. Applications communicate directly with each other using the MOM and the messages generated by those applications. MOM acts as a message router and in some cases as a queuing system. MOM supports synchronous and asynchronous messaging; however, it is mostly associated with asynchronous messaging using queuing.

2.4. Comparison Hub-and-Spoke versus Point-to-Point Integration

This section discusses the positive and negative sides of each technology. A simple visual comparison between the two integration topologies makes obvious that the number of connections is significantly smaller for hub-and-spoke, which makes it easier to maintain, due to a lesser number of dependencies.

Another significant reason to adopt hub-and-spoke topology instead of large numbers of point-to-point interfaces is the decoupling of applications. Applications, then maintain only the connection with the hub, instead of relying on multiple communication channels with other applications. This is very a powerful way for applications to achieve independence of other systems and, hence to allow asynchronous communication.

In the hub-and-spoke topology, all data pass through the hub, therefore providing an ideal place to centralize automation of the business process.

In some cases the cost of handling system failure could potentially outweigh the maintenance cost of the integration software. A hub-and-spoke-based environment centralizes all of the data and processes, thus centralizing the handling of the hardware and software failures.

Considering the number of interdependencies that exist for any application within a point-to-point integration topology, an upgrade to a new version of the application software could require changes to be made to all other applications within a given point-to-point topology. Changes in the hub-and-spoke-based applications are isolated to the interface between them and the hub. Therefore adding a brand new application or interface to the hub-and-spoke environment is even easier.

From this discussion it is obvious that hub-and-spoke has many advantages over point-to-point topology. However, one might wonder if there are any scenarios in which point-to-point is a better solution. An obvious scenario in which it is more efficient to use a point-to-point topology is when there are only two applications. In that case, point-to-point requires fewer integration steps than hub-and-spoke, therefore offering a better and simpler solution.

For all these reasons the industry trend is gradually to replace expensive point-to-point environments with a hub-and-spoke architecture. A hub-and-spoke offers an opportunity to improve overall system architecture for centralization of the business processes, as well as more cost efficient long-term maintenance.

3. RECOMMENDED APPROACH

3.1. Target Architecture

Just by reviewing the topologies and distributed technologies listed above, it is almost self-explanatory that the hub-and-spoke topology is a preferred approach, because of its simplicity. This topology, combined with one of the loosely-coupled techniques, significantly increases the flexibility and agility of the system and as significantly improves time to market. Additional flexibility can be accomplished by creating the integration strategy based on industry standards.

The most commonly used technique to formalize these goals is Service-Oriented Architecture (SOA). SOA is an architecture framework for integrating business processes and technology infrastructure, based on platform independent standards, loosely-coupled, well-defined and self-contained sets of interfaces—i.e., services. Platform-independent standards are necessary, as they address heterogeneous environments, platform-independent modelling, meta-data and operational standards. Therefore, a physical implementation of the architecture from Figure III, due to the number of disparate systems and technology platforms that need to be integrated, can be achieved only by using products based on a standard technology stack.

From the business perspective, SOA represents a set of flexible services and processes that need to be exposed to customers and partners. On the other side, from the technical perspective, SOA defines interfaces as well-defined, stateless services, implemented using software components that perform specific business tasks.

SOA is not a new idea. In the last two decades there have been solutions built on SOA principles, but implemented in a proprietary way [2]. These solutions have been possible on a departmental or multi- departmental level, however, the main drawbacks of previous SOA-like solutions were the inability to scale across large enterprises and their partners, and as well the cost of implementation.

The major reason the term "SOA" is becoming so popular is the emergence of new technologies that make SOA more scalable and cost efficient. These new technologies are Web Services, the Enterprise Service Bus (ESB) and the Business Process Execution Language (BPEL) [2, 5].

Web Services standards [11, 12, 18] are important because they offer interoperability between proprietary technologies. Web Services are based on the following standards: Simple Object Access Protocol (SOAP) [16, 17, 20], for message definition in a distributed environment; Web Services Description Language (WSDL) [19], for access and invocation of the services; Universal Description, Discovery and Integration (UDDI) for service discovery and WS-I for interoperability [6].

The Enterprise Service Bus (ESB) represents an automated, self-managed SOA realization [5]. The ESB provides essential infrastructure services: transport, quality of service-based routing, mediation and gateway services [2]. At the core of the ESB communication is a MOM capable of supporting standard access mechanisms and protocols, while physical topology closely resembles hub-and-spoke. The ESB provides the abilities of: spanning distributed locations, traversing firewall security [13] using certificate-based authentication, delivering messages reliably and a store-and-forward capability.

Due to the fact that business processes are essential to a successful SOA implementation, the model includes the BPEL [10] as an integral part of a Business Process Orchestration Engine. BPEL is used by business analysts to model business processes and by designers to design services.

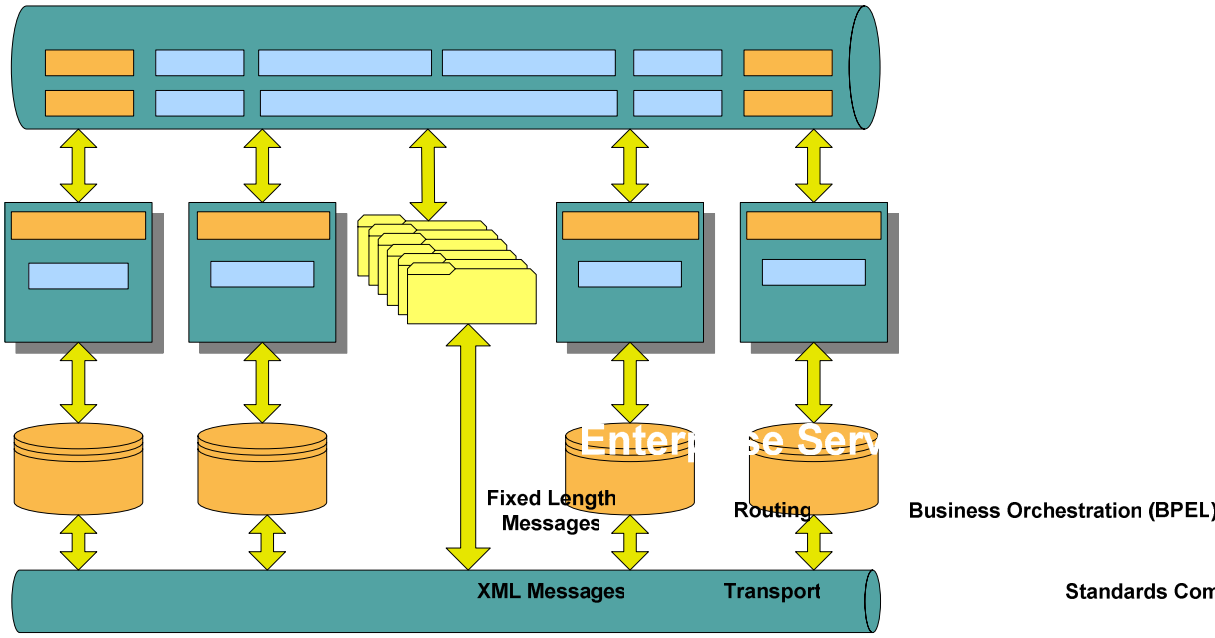

Figure III: Target Integration Architecture

3.2. Integration guiding principles

A number of themes underlie and connect the Integration guiding principles described in this paper. Together, these themes enable the overall integration vision for an enterprise, and provide a concise list of the major objectives, fundamental to each one of the principles. The following list represents the core architectural guiding principles for the integration:

- **Plan for integration** – Traditionally, application integration has often been an afterthought, resulting in missing or inferior interfaces built quickly on a tight budget and therefore increasing deployment costs and time to market, making the application high-maintenance during future evolutions.

- **Platform independence** – Independence of the solution from the underlying technology allows it to be developed, upgraded, and operated in the most cost-effective and timely way. This will allow a variety of deployment strategies to be developed without modifications of the solution. Platform independence is the recommended approach for complex software solutions, but especially so within heterogeneous environments.

- **Scalability** – The solution must scale to address current and future volume needs driven by an enterprise's expanding business requirements and to minimize the need to keep reinvesting in new technologies. A scalable architecture supports large numbers of users with minimal hardware investments and IT support.

- **Use of a standard infrastructure** – Technological diversity should be controlled to minimize the non-trivial cost of maintaining expertise and connectivity between multiple processing environments. There is a real, non-trivial cost of the infrastructure required to support alternative technologies for processing environments. There are further infrastructure costs incurred to keep multiple processor constructs interconnected and maintained. Limiting the number of supported components will simplify maintainability and reduce costs. The business advantages of minimum technical diversity include: standard packaging of components, predictable implementation impact, predictable valuations and returns, refined testing, utility status, and increased flexibility to accommodate technological advancements.

- **Flexibility** – A flexible system reduces the costs of adding a new solution to the system, as well as migrating from one solution to another. Flexible solutions simplify the migration strategies from one solution to another, as changes would be managed by the Integration Solution only and do not affect other applications.

- **Use of standard logical data definitions** – The data that will be used in the development of applications integration must have a common definition throughout the enterprise to enable the sharing of those data. A common data and message structure should facilitate communications between systems, data exchange with vendor applications, and will furthermost promote SOA-based reusability.

- **Maintainability and Change/Configuration management** – The integration tier is replacing hard-coded routing/transformation logic that used to belong to the application; therefore the process of changing such routing/transformation logic within the integration tier should be controlled in the same way as the development process used to maintain a piece of software, e.g., change-managed and with the use of configuration management tools for the versioning of integration logic.

- **Process flow capability** – As the number of applications integrated to the integration tier grows, the message processing may also become more complicated. The integration tier should be able to process the message in different sequences. Additionally, the process flow should be easily defined in the integration tier.
- **Ease of deployment** – During a development process, the process flow and mapping should be easily deployed to either the development or testing environment without having to change the flow definition. With this capability, overall development and testing time may be saved. Having a GUI tool, the support centre should be able to perform localization or maintenance, as the process flow and mapping will be visible from the tool.
- **Content-based routing** – For flexibility, the message should be able to be routed based on the message content information. In error-handling scenarios, the integration tier should be capable of choosing different process flows based on the response message content.
- **Reliability** – As the central hub for message exchange amongst the applications, the integration tier is becoming critical and should be up and running 24 by 7. If the integration tier is down, the whole system will be affected.
- **Error Handling Mechanism** – Various types of non-critical transactions may post into an audit log and report on the next business day; however, for the critical transactions, the integration tier must perform compensating tasks to rollback previous updates, and to notify the requesting client.
- **Monitoring, Auditing and Logging** – Certain transaction logs, due to auditing requirements, need to be viewed and monitored; therefore the tool must provide a feature to identify a single transaction or a group of transactions from numerous transaction logs. For planning and analysis reasons, there is a need to report, by transaction type, the total number of transactions at certain periods of time.
- **Security** – Users must be authenticated with an internal access control list or external LDAP directory before granting integration system resource access rights. All incoming messages must be verified to ensure they come from secured system resources.
- **Support File Interface** – In order to overcome deficiencies of the Extract, Transform and Load (ETL) integration, a system should support file interfaces for online and batch modes.

- **Guaranteed Delivery –** Systems need to provide a feature for persisting messages, so that the messages are not lost even if the messaging (sub-) system is not available.

- **Asynchronous delivery –** As asynchronous messaging is the strategy of choice for loosely-coupled, service-oriented architectures because it overcomes the inherent limitations of remote communication, such as latency and unreliability, it must be supported.

- **Transaction Support** – One of core requirements in financial (and similar) institutions is to maintain logical consistency of information across one or more database operations, and this consistency is provided by the availability and use of transactions.

3.3. System design considerations

In this type of architecture, the following system design considerations are critical, and must be properly addressed:

- Service granularity.
- Data integration, by making the choice either to adapt a legacy application or instead to use transformations inside the ESB.
- Latency limitation, by careful design of caching throughout the information system.

Integration should be achieved with a combination of ESB and ETL tools. The ESB technology based on Web Services is standard, and therefore will be the preferred choice, instead of proprietary integration techniques, as presented in the Figure III.

The ESB layer is architected to handle online and batch transactions. It uses MOM-like products as an underlying infrastructure, but it can also rely on plain HTTP, although this transport mode does not of itself support guaranteed delivery. The ESB must provide routing, transformation and monitoring capabilities, in addition to the transport protocol.

Extract, Transform and Load combined with batch and FTP is dominating ways of achieving integration in most large enterprises [5]. Because of the unreliability of data transfer, lack of data validity, downtime and the latency of data gathering, it is recommended to use an ESB file service to prevent these drawbacks. The ESB provides means of handling these shortcomings out of the box, as reliable messaging, transforming and routing are integral part of most commercial products. The ETL layer should handle high-volume transactions through the ESB file service, or through direct links with database management systems.

4. CONCLUSION

This paper discussed a type of integration architecture, based on Web Services, Enterprise Service Bus and Business Process Execution Language. The historical evolution of the two basic topologies was presented, along with the distributed technologies used to achieve integration. A number of positive and negative attributes was identified, leading to a conclusion that a mix of hub-and-spoke architecture and MOM infrastructure, based on standards, is the recommended approach. That type of hub-and-spoke or "Enterprise Bus" integration architecture was originally introduced to avoid point-to-point integration, since this was leading to a less manageable and more expensive spaghetti-like connectivity. Over the years this architecture has evolved with the emergence of standards such as Web Services. Shortly after, business process orchestration, transaction and security tools were also introduced, while the integration technologies were formally standardized (ESB). With all these standards in place, it is becoming much easier to support critical Straight-Through-Processing (STP) improvements and business process re-engineering efforts in a more flexible and thus cost-efficient manner.

The most important aspect of successful implementation of the SOA software model is that it must be realized using platform-independent standards.

A SOA strategy should be based on current architecture and its progressive and evolutionary roadmap. The roadmap should represent an iterative process. In order to reduce risk an organization should go through validation and readiness assessments. The next step in adopting SOA should be Line-Of-Business (LOB)-centric implementations that increase agility and flexibility. Enterprise adoption involves creation of the business architecture that spans across an entire enterprise. Business-to-business adoption is required only when an enterprise creates business value as a service consumer, provider or broker.

SOA as a business re-engineering framework needs a strong governance model to ensure appropriate handling of the non-SOA products and to ensure compliance of vendor products. SOA governance is built around guiding principles that are derived from business drivers, best practices and technology standards. The governance model should provide the rules, processes, metrics and organizational structure required for effective decision making and control of the SOA deployment.

As mentioned before, SOA is particularly applicable to heterogeneous environments that are subject to frequent changes. Therefore in situations where this is not the case SOA might not be the right solution, due to its performance overhead, the increased need for architectural discipline and the requirements for companies to change the way they consume and produce their software assets. Companies that have a homogeneous IT environment and use a single vendor's technology may find SOA being not cost-effective. Systems that require real-time performance and do not change frequently are not a good match for SOA. Applications deployed on a single computer are recommended to be tightly coupled, because loosely coupled components would introduce needless performance overheads.

SOA has many advantages and benefits within large enterprises, yet it is not the only solution for all the problems. Distributed objects, and other technologies, depending on the

design approach [3, 9, 14]; will still play a role in internal SOA implementations as well as in the above-mentioned circumstances. Distinguishing the situations where the applicability of SOA is inappropriate is a key success factor for SOA implementation.

A recommended direction for further development of this concept would be to virtualize processing and storage resources into a set of unified services, so that they can operate independently. This should contribute to achieving a reduction of needed processing power and the physical storage, therefore resulting in a decrease of the initial and the maintenance costs.

REFERENCES

[1] Bakken, D.E., *Middleware, Encyclopedia of Distributed Computing*, Kluwer Academic Press, 2001.
[2] Bieberstein, N., Bose, S., Fiammante, M., Jones, K., and Shah, R., "Service-Oriented Architecture Compass", IBM Press 2006.
[3] Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, 2005.
[4] Campbell, A.T., Coulson, G., and Kounavis, M.E., "Managing complexity: middleware explained", *Distributed Computing, IT Professional*, 1(5) (1999) 22-28.
[5] Chappell, A.D., *Enterprise Service Bus*, O'Reilly, 2005.
[6] Earl, T., *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*, Prentice Hall PTR, 2004.
[7] Fowler, M., *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional, 2002.
[8] Keen, M., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., and Verschueren, P., "Patterns: Implementing an SOA using an Enterprise Service Bus" IBM Press 2004.
[9] Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall PTR, 2001.
[10] OASIS, BPEL: Web Services Business Process Execution Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.
[11] OASIS, Web Services Architecture http://www.w3.org/TR/ws-arch.
[12] OASIS, Web Services Resource Framework http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=wsrf.
[13] OASIS, Web Services Security, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
[14] OMG, UML: Unified Modeling Language, Standard Specification, http://www.uml.org.
[15] Qusay, H.M., *Middleware for Communications*, Wiley, 2004.
[16] SOAP 1.1 Specification, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.
[17] SOAP 1.2 Specification, http://www.w3.org/2000/xp/Group/.
[18] W3C, Web Services. http://www.w3.org/2002/ws/.
[19] W3C, WSDL Web Services Description Language Version 2.0. Standard Specification, May 2005. http://www.w3.org/TR/wsdl20?.
[20] WS-Policy vs SOAP 1.2 Features and Properties discussion: http://lists.w3.org/Archives/Public/www-ws-desc/2003Oct/0144.html.