Business Process Management Integration Solution in Financial Sector

Silviu Florin TEODORU
Oracle Romania, Bucharest, Romania
teodorusil@yahoo.com

It is vital for financial services companies to ensure the rapid implementation of new processes to meet speed-to-market, service quality and compliance requirements. This has to be done against a background of increased complexity. An integrated approach to business processes allows products, processes, systems, data and the applications that underpin them to evolve quickly. Whether it's providing a loan, setting up an insurance policy, or executing an investment instruction, optimizing the sale-to-fulfillment process will always win new business, cement customer loyalty, and reduce costs. Lack of integration across lending, payments and trading, on the other hand, simply presents competitors who are more efficient with a huge profit opportunity.
Keywords: *Web Service, business process, integration, financial services, integration, modeling.*

1 Introduction

Financial institutions today combine a wide range of product and service offerings, across banking, insurance and asset management. They operate in global and cross border markets. They have increasingly sophisticated and mobile customer bases. Increased regulatory vigilance and new corporate governance rules have the potential to add new layers of complexity and cost. And there continues to be consolidation, merger and acquisition in the sector.
For all these reasons the effective management of complexity and change is a key determinant of future success. Those who automate and streamline their operations most effectively will gain significant advantage. Integration is now more than ever the key to efficiency, enabling lower transaction costs and increased sales volumes. This is true for capital markets, for retail financial services, and for the corporate sector.
Integration and process optimization not only has to extend across the enterprise, but must also embrace third parties who often supply key components of today's complex, multi-instrument financial products. A mortgage offer for example will typically involve underwriters, insurers, the customer's bank, credit reference agencies and others, as well as internal approval, accounting, collections, credit control, risk management, commission payment, incentive management, and business intelligence processes. Improve all the connections between all the elements of a transaction and performance automatically improves.
According to [7], the problem is that established financial organizations still have numerous, disparate, proprietary back-office systems which cannot keep pace. These limit the capabilities of even the most advanced front-end systems. The remodeling and implementation of new processes to meet the demands of dynamic change are severely constrained.
There are three objectives to aim for:
- ✓ driving greater efficiency and value from existing systems and processes;
- ✓ managing the risks associated with dynamic change;
- ✓ achieving greater visibility and flexibility across complex operations.

In next paragraphs it is described an approach to process management which delivers those aims. At last financial institutions have a solution that allows them to cater for and anticipate change, allowing far greater operational and marketplace efficiency, while meeting regulatory and governance requirements.

2. Basic concepts

A ***Web service*** is defined by the [2] as "a software system designed to support interoperable machine-to-machine interaction over a network". Web Services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

SOAP, originally defined as *Simple Object Access Protocol*, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) as its message format and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and HTTP for message negotiation and transmission. SOAP forms the foundation layer of the Web Services protocol stack providing a basic messaging framework upon which abstract layers can be built [1].

The **Web Services Description Language** is an XML-based language that provides a model for describing Web Services. The WSDL defines services as collections of network endpoints, or ports. The WSDL specification provides an XML format for documents for this purpose. The abstract definition of ports and messages are separated from their concrete use or instance, allowing the reuse of these definitions [6].

The **Web Services Interoperability Organization** (WS-I) is an industry consortium chartered to promote interoperability amongst the stack of Web Services specifications. WS-I does not define standards for Web Services; rather, it creates guidelines and tests for interoperability.

The [2] Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate over the HTTP protocol used on the Web. Such services tend to fall into one of two camps: ***Big Web Services*** and ***RESTful Web Services***.

"***Big Web Services***" use XML messages that follow the SOAP standard and have been popular with traditional enterprise. In such systems, there is often machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP *endpoint*, but it is a prerequisite for automated client-side code generation in many Java and .NET SOAP frameworks (frameworks such as Spring, Apache Axis2 and Apache CXF being notable exceptions). Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a Web Service.

More recently, "***RESTful Web Services***" have been regaining popularity, particularly with Internet companies. These also meet the W3C definition, and are often better integrated with HTTP than SOAP-based services. They do not require XML messages or WSDL service-API definitions.

WS-Security (**Web Services Security**) is defined in [1] as a communications protocol providing a means for applying security to Web Services. On April 19, 2004 the WS-Security 1.0 standard was released by Oasis-Open. On February 17, 2006 they released version 1.1. Originally developed by IBM, Microsoft, and VeriSign, the protocol is now officially called WSS and developed via committee in Oasis-Open.

The protocol contains specifications on how integrity and confidentiality can be enforced on Web Services messaging. The WSS protocol includes details on the use of SAML and Kerberos, and certificate formats such as X.509.

WS-Security describes how to attach signatures and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages. WS-Security incorporates security features in the header of a SOAP message, working in the application layer. Thus it ensures end-to-end security.

A **business process** is a collection of related, structured activities or tasks that produce a specific service or product (serve a particular goal) for a particular customer or customers.

Business Process Execution Language (BPEL), short for *Web Services Business Process Execution Language* (WS-BPEL) is an executable language for specifying interactions with Web Services Processes in Business Process Execution Language export and import information by using Web Service interfaces exclusively [3].

Business Process Modeling (BPM) in systems engineering and software engineering is the activity of representing processes of an enterprise, so that the current ("as is") process may be analyzed and improved in future ("to be"). BPM is typically performed by business analysts and managers who are seeking to improve process efficiency and quality. The process improvements identified by BPM may or may not require Information Technology involvement, although that is a common driver for the need to model a business process, by creating a process master [7].

Service-oriented architecture (**SOA**) provides, according to [4], methods for systems development and integration where systems group functionality around business processes and package these as *interoperable services*. SOA also describes IT infrastructure which allows different applications to exchange data with one another as they participate in business processes. Service-orientation aims at a *loose coupling* of services with operating systems, programming languages and other technologies which underlie applications. SOA separates functions into distinct units, or services, which developers make accessible over a network in order that users can combine and reuse them in the production of business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services.

3. Business Processes Execution and Modeling

Web Service interactions can be described in two ways [3]:

✓ Executable business processes model actual behavior of a participant in a business interaction;

✓ Abstract business processes are partially specified processes that are not intended to be executed. An Abstract Process may hide some of the required concrete operational details. Abstract Processes serve a descriptive role, with more than one possible use case, including observable behavior and process template.

WS-BPEL is meant to be used to model the behavior of both Executable and Abstract Processes. WS-BPEL provides a language for the specification of Executable and Abstract business processes. By doing so, it extends the Web Services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

The origins of BPEL can be traced to WSFL and XLANG. It is serialized in XML and aims to enable *programming in the large*. The concepts of *programming in the large* and *programming in the small* distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes.

Programming in the large generally refers to the high-level state transition interactions of a process - BPEL refers to this concept as an Abstract Process. A BPEL Abstract Process represents a set of publicly observable behaviors in a standardized fashion. An Abstract Process includes information such as when to wait for messages, when to send messages, when to compensate for failed transactions, etc. *Programming in the small*, in contrast, deals with short-lived programmatic behavior, often executed as a single transaction and involving access to local logic and resources such as files, databases, etc. BPEL's development came out of the notion that programming in the large and programming in the small required different types of languages.

BPEL is an orchestration language, not a choreography language. The primary difference between orchestration and choreography is executability and control. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. Choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing interoperability. Such a protocol is not directly executable, as it allows many different realizations (processes that comply with it). A choreography can be realized by writing an orchestration (e.g. in the form of a BPEL process) for each peer involved in it. The orchestration and the choreography distinctions are based on analogies: orchestration refers to the central control (by the conductor) of the behavior of a distributed system (the orchestra consisting of many players), while choreography refers to a distributed system (the dancing team) without centralized control.

BPEL's focus on modern business processes, plus the histories of WSFL and XLANG, led BPEL to adopt Web Services as its external communication mechanism. Thus BPEL's messaging facilities depend on the use of the Web Services Description Language (WSDL) 1.1 to describe outgoing and incoming messages.

In addition to providing facilities to enable sending and receiving messages, the BPEL programming language also supports:

- ✓ A property-based message correlation mechanism;
- ✓ XML and WSDL typed variables;
- ✓ An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports XPath 1.0 by default;
- ✓ Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel);
- ✓ A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers;
- ✓ Serialized scopes to control concurrent access to variables.

Relationship of BPEL to BPMN

There is no standard graphical notation for WS-BPEL, as the OASIS technical committee decided this was out of scope. Some vendors have invented their own notations. These notations take advantage of the fact that most constructs in BPEL are block-structured (e.g. sequence, while, pick, scope, etc.). This feature enables a direct visual representation of BPEL process descriptions in the form of *structograms*.

Others have proposed to use a substantially different business process modeling language, namely Business Process Modeling Notation (BPMN), as a graphical front-end to capture BPEL process descriptions. As an illustration of the feasibility of this approach, the BPMN specification includes an informal and partial mapping from BPMN to BPEL 1.1. A more detailed mapping of BPMN to BPEL has been implemented in a number of tools, including an open-source tool known as BPMN2BPEL. However, the development of these tools has exposed fundamental differences between BPMN and BPEL, which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models. Even more difficult is the problem of BPMN-to-BPEL *round-trip* engineering: generating BPEL code from BPMN diagrams and maintaining the original BPMN model and the generated BPEL code synchronized, in the sense that any modification to one is propagated to the other.

4. Service-oriented architecture

According to [4], companies have long sought to integrate existing systems in order to implement information technology (IT) support for business processes that cover all present and prospective systems requirements needed to run the business end-to-end. A variety of designs serve this end, ranging from rigid point-to-point electronic data interchange (EDI) interactions to web auctions. By updating older technologies, for example by Internet-enabling EDI-based systems, companies can make their IT systems available to internal or external customers; but the resulting systems have not proven flexible enough to meet business demands, which require a flexible, standardized architecture to better support the connection of various ap-

plications and the sharing of data.
SOA offers such architecture. It unifies business processes by structuring large applications as an *ad hoc* collection of smaller modules called "services". Different groups of people both inside and outside an organization can use these applications, and new applications built from a mix of services from the global pool exhibit greater flexibility and uniformity. One should not, for example, have to provide redundantly the same personal information to open an online checking, savings or Individual Retirement Arrangement (IRA) account, and further, the interfaces one interacts with should have the same look and feel and use the same level and type of input data validation. Building all applications from the same pool of services makes achieving this goal much easier and more deployable to affiliate companies.
Service Oriented Architecture (SOA) provides a design framework for realizing rapid and low-cost system development and improving total system quality. SOA uses the Web Services standards and technologies and is rapidly becoming a standard approach for enterprise information systems.
Web Services face significant challenges because of particular requirements. Applying the SOA paradigm to a real-time system throws up many problems, which include response time, support of event-driven, asynchronous parallel applications, complicated human interface support, reliability, etc. This article defines SOA and includes detailed discussion on several issues that arise when applying SOA to industrial systems.
One can define a *service-oriented architecture* (SOA) as a group of services that communicate with each other. The process of communication involves either simple data-passing or two or more services coordinating some activity. Intercommunication implies the need for some means of connecting services to each other.
SOAs build applications out of software services. Services comprise intrinsically unassociated units of functionality that have no calls to each other embedded in them. They typically implement functionality most humans would recognize as a service, such as filling out an online application for an account, viewing an online bank-statement, or placing an online booking or airline ticket order. Instead of services embedding calls to each other in their source code, they use defined protocols which describe how one or more services can talk to each other. This architecture then relies on a business process expert to link and sequence services, in a process known as orchestration, to meet a new or existing business system requirement.
Relative to typical practices of earlier attempts to promote software reuse via modularity of functions, or by use of predefined groups of functions known as classes, SOA's atomic-level objects are often 100 to 1,000 times larger.
An application designer or engineer associates SOA objects by using orchestration. In the process of orchestration, a software engineer or process engineer associates relatively large chunks of software functionality (services) in a non-hierarchical arrangement (in contrast to a class hierarchy) by using a special software tool which contains an exhaustive list of all of the services, their characteristics, and a means to record the designer's choices which the designer can manage and the software system can consume and use at run-time.
Underlying and enabling all of these, one requires metadata in sufficient detail to describe not only the characteristics of these services, but also the data that drives them. Programmers have made extensive use of XML in SOA to create data which is wrapped in a nearly exhaustive description container. Analogously, the services themselves are typically described by WSDL, and communications protocols by SOAP. Whether these description languages are the best possible for the job, and whether they will remain the favorites in the future, remains an open question. In the meantime, SOA depends on data and services that are described using some implementation of metadata which meets the following two criteria:
✓ the metadata must be in a form which software systems can use to configure them-

selves dynamically by discovery and incorporation of defined services, and also to maintain coherence and integrity;

✓ the metadata must also be in a form which system designers can understand and manage at a reasonable cost and effort.

SOA has the goal of allowing fairly large chunks of functionality to be strung together to form *ad hoc* applications which are built almost entirely from existing software services. The larger the chunks, the fewer the interface points required implementing any given set of functionality; however, very large chunks of functionality may not prove granular for easy reuse. Each interface brings with it some amount of processing overhead, so there is a performance consideration in choosing the granularity of services. The great promise of SOA suggests that the marginal cost of creating the n-th application is zero, as all of the software required already exists to satisfy the requirements of other applications. Ideally, one requires only orchestration to produce a new application.

For this to operate, no interactions must exist between the chunks specified or within the chunks themselves. Instead, the interaction of services (all of which are unassociated peers) is specified by humans in a relatively *ad hoc* way with the intent driven by newly emergent business requirements. Thus, the need for services with larger units of functionality than traditional functions or classes and the sheer complexity of thousands of such granular objects, overwhelm the application designer. Programmers develop the services themselves using traditional languages like Java, C#, C++, C or COBOL.

SOA services feature loose coupling, in contrast to the functions a linker binds together to form an executable, a dynamically linked library, or an assembly. SOA services also run in "safe" wrappers such as Java or .NET, and other programming languages that manage memory allocation and reclamation, allow *ad hoc* and late binding, and provide some degree of indeterminate data typing.

As of 2008, increasing numbers of third-party software companies offer software services for a fee. In the future, SOA systems may consist of such third-party services combined with others created in-house. This has the potential to spread costs over many customers, and customer uses, and promotes standardization both in and across industries. In particular, the travel industry now has a well-defined and documented set of both services and data, sufficient to allow any reasonably competent software engineer to create travel-agency software using entirely off-the-shelf software services. Other industries, such as the finance industry, have also started making significant progress in this direction.

As architecture, SOA relies on service-orientation as its fundamental design principle. In a SOA environment, users can access independent services without knowledge of their underlying platform implementation. SOA relies on services exposing their functionality via interfaces which other applications and services can read to understand how to utilize those services.

The layers of a service-oriented architecture

An abstract view of SOA depicts it as a partially layered architecture of composite services that align with business processes. Figure 1 depicts a representation of this type of architecture, as described in [5].

The relationship between services and components is that enterprise-scale components (large-grained enterprise or business line components) realize the services and are responsible for providing their functionality and maintaining their quality of service. Business process flows can be supported by choreography of these exposed services into composite applications. Integration architecture supports the routing, mediation, and translation of these services, components, and flows using an Enterprise Service Bus (ESB). The deployed services must be monitored and managed for quality of service and adherence to non-functional requirements.

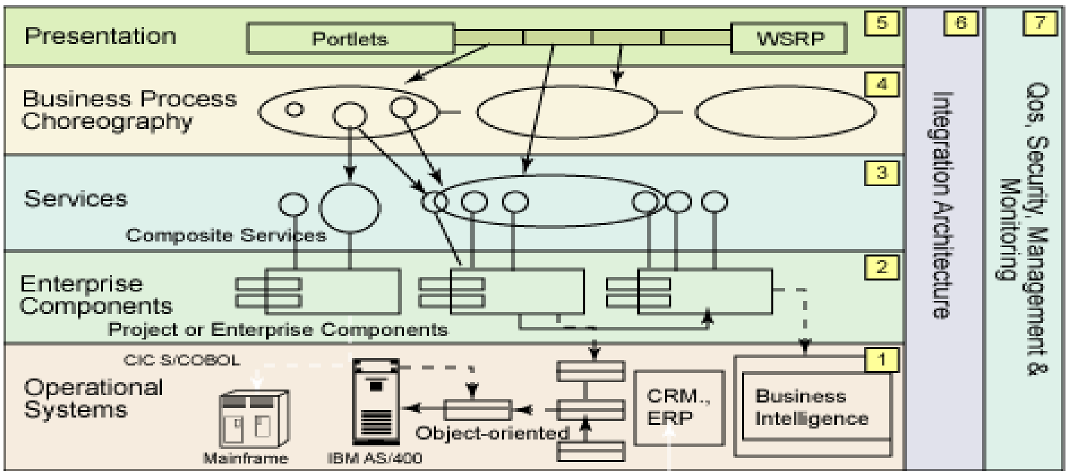


Fig. 1. The layers of a service-oriented architecture

Layer 1: Operational systems layer. This consists of existing custom built applications, otherwise called *legacy* systems, including existing CRM and ERP packaged applications, and *older* object-oriented system implementations, as well as business intelligence applications. The composite layered architecture of an SOA can leverage existing systems and integrate them using service-oriented integration techniques.

Layer 2: Enterprise components layer. This is the layer of enterprise components that are responsible for realizing functionality and maintaining the quality of exposed services. These special components are a managed, governed set of enterprise assets that are funded at the enterprise or the business unit level. As enterprise-scale assets, they are responsible for ensuring conformance to SLAs through the application of architectural best practices. This layer typically uses container-based technologies such as application servers to implement the components, workload management, high-availability, and load balancing.

Layer 3: Services layer. The services the business chooses to fund and expose reside in this layer. They can be *discovered* or be statically bound and then invoked, or possibly, choreographed into a composite service. This service exposure layer also provides for the mechanism to take enterprise scale components, business unit specific components, and in some cases, project-specific components, and externalizes a subset of their interfaces in the form of service descriptions. Thus, the enterprise components provide service realization at runtime using the functionality provided by their interfaces. The interfaces get exported out as service descriptions in this layer, where they are exposed for use. They can exist in isolation or as a composite service.

Level 4: Business process composition or choreography layer. Compositions and choreographies of services exposed in Layer 3 are defined in this layer. Services are bundled into a flow through orchestration or choreography, and thus act together as a single application. These applications support specific use cases and business processes. Here, visual flow composition tools can be used for the design of application flow.

Layer 5: Access or presentation layer. Although this layer is usually out of scope for discussions around a SOA, it is gradually becoming more relevant. It was depicted here because there is an increasing convergence of standards, such as Web Services for Remote Portlets Version 2.0 and other technologies, that seek to leverage Web Services at the application interface or presentation level. We can think of it as a future layer that should be taken into account for future solutions. It is also important to note that SOA decouples the user interface from the components, and that we ultimately need to provide an end-to-end solution from an access channel to a service or composition of services.

Level 6: Integration (ESB). This layer enables the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, often described as the ESB. Web Services Description Language (WSDL) specifies a binding, which implies a location where the service is provided. On the other hand, an ESB provides a location independent mechanism for integration.

Level 7: Quality of Services (QoS). This layer provides the capabilities required to monitor, manage, and maintain QoS such as security, performance, and availability. This is a background process through sense-and-respond mechanisms and tools that monitor the health of SOA applications, including the all important standards implementations of WS-Management and other relevant protocols and standards that implement quality of service for a SOA.

5. Business process management in financial institutions

On one side of the chasm there is the business need (and the process to satisfy it) as modeled by the business analyst. On the other side of the chasm is the application development resource needed to deliver a system solution. The key is to create a conversation between the two sides. For a conversation to make sense we need a common language. The emerging standard for that language of process execution is BPEL - Business Process Execution Language.

According to [7], BPEL allows an organization to treat processes and the applications that underpin them as utilities: to define, translate and transpose a business process, and make the applications which Web Services expose available to build a new process. Using the BPEL tools allows an organization to effectively orchestrate (or manage) Web Services and the integration architecture as a whole.

A recent Forrester Research note crystallizes why BPEL moves the integration story forward so significantly: *"BPEL will provide not only a way to integrate applications, but also a way to create services from them and put them into business processes"*.

Just as HTML allows content to make sense and to be published and used by anyone who has a web browser, BPEL allows processes to make sense, and for new processes to be planned, developed and deployed with exactly the same universal simplicity and ease of use.

Gartner recently commented: *"BPEL will emerge as the leading industry standard for Web Service orchestration and coordination of business processes."*

For financial service companies with entrenched legacy systems, and departmental silos based around proprietary or highly specialized applications, the opportunity offered by BPEL has one other major advantage: it does not add more software to the inventory.

In fact, it enables the organization to get more from its existing investment in applications and processes. Not only more value in terms of more process development potential, but more competitive benefit from faster time to market for new products and services through more effective service provisioning.

The standards emerging around Web Service orchestration such as SOAP, WSDL, XML Schema and BPEL enable financial institutions to address their integration and business process management requirements in a vendor independent fashion.

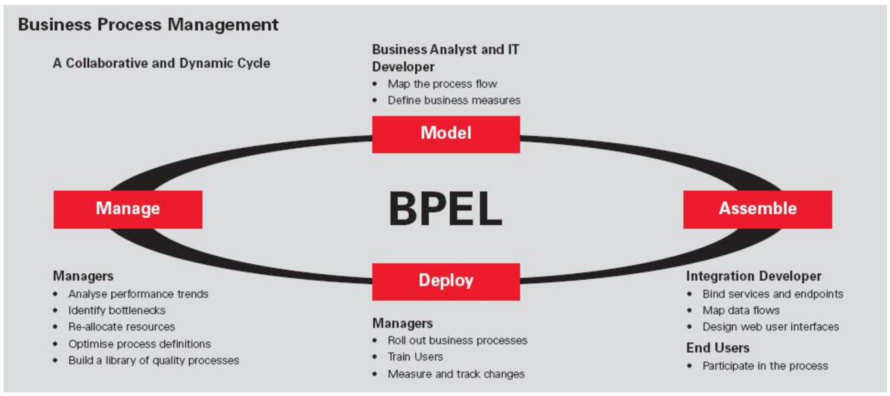
It also makes sense for financial institutions to build their systems with a loosely coupled, service-oriented architecture so that they will be able to get the efficiency of highly integrated systems while minimizing the cost, time and resources required building and maintaining them.

Using a BPEL approach allows financial institutions to model business processes very quickly, connecting together the applications and partners that support it, and to deploy the process directly. They can then test and refine these business processes more effectively. Finally using BPEL they can monitor the effectiveness of this process, and draw intelligence from the information running through it. In other words they obtain real time business intelligence to refine the process and validate it.

In most of the cases it is required both integration of existing functionality *and* new ap-

plication development. It incorporates both Internet/B2B (business to business) style integration and Intranet/A2A (application to application) integration.
Implementing the system requires to integrate disparate developer skills, methodologies and infrastructures into a maintainable application.
The approach - using Web Services as a standard service interface and BPEL for process orchestration – works equally well for enterprise-wide Intranet-based integration or for collaboration beyond the organization.
Key issues for financial services organizations seeking to leverage a process-centric approach are:
✓ Extract functionality efficiently using Web Services;
✓ Reduce the set up costs of new processes;
✓ Bring new services to market as quickly as possible;
✓ Increase standardization and interoperability;
✓ Enhance application reliability;
✓ Ensure security.

More effective business process delivery
If business analysts can leverage the power of BPEL by operating a dashboard to control activities and create a new business process on their own desktop, they will be more productive and the business will be more agile.
If at the same time if a developer can rapidly implement a new service using the building blocks of existing processes which have already been published as Web Services, or using other components and metadata within the Service Oriented Architecture, they will be able to deliver against demanding time and cost targets.
The challenge for financial services organizations is rapidly leverage their own applications, systems and processes, and also those of trading partners and customers, to bring about business process optimization.
As described in figure 2, Business Process Management supports each stage of the process optimization lifecycle:
❖ **Model** - It provides business analysts with a GUI console to model new processes, referencing existing Web Services and applications;
❖ **Deploy** - This standard process model can then be run directly on any Application Server using the Process Manager Console;
❖ **Manage** - The processes can thus be monitored, refined and re-run quickly and effectively;
❖ **Integrate** - Business Process Management also allows the developer community to manage or 'orchestrate' the Web Services already being developed and which are available within and beyond the enterprise. Until now, these have been 'hard wired' together using traditional coding techniques.


Business Process Management
A Collaborative and Dynamic Cycle
Business Analyst and IT Developer
• Map the process flow
• Define business measures
Model
Manage
BPEL
Assemble
Deploy
Managers
• Analyse performance trends
• Identify bottlenecks
• Re-allocate resources
• Optimise process definitions
• Build a library of quality processes
Managers
• Roll out business processes
• Train Users
• Measure and track changes
Integration Developer
• Bind services and endpoints
• Map data flows
• Design web user interfaces
End Users
• Participate in the process
Fig. 2. Business Process Management – A Collaborative and Dynamic Cycle

The benefits are clear:
✓ Far shorter timescales to delivering and running processes;
✓ Closing the communication gap between the business and IT;
✓ Delivering a single consistent view of the process that both business and developers can understand;

✓ Effectively orchestrating integration between services and applications within the enterprise.

The largest prize is that by adopting this approach to integration, organizations can start to understand their processes and applications far better. They gain visibility of the resources they need, of the efficiency of processes, and of the dependencies and constraints that make or break process effectiveness.

Finally organizations can start to track information flows within transactions and services, responding to events, applying current information to historic overviews, and delivering true real-time business intelligence.

6. Conclusions

Success in highly competitive global financial services markets comes to those who are differentiated by virtue of their ability to innovate with new products, services and alliances, characterized by their speed to market, their responsiveness to change, their capacity to reduce costs, their excellence of customer service, and their control of risk and uncertainty. Process execution underpins all those ambitions.

A process-centric approach to developing the business and its underlying information management systems and applications will pay immediate dividends: performance will improve, costs will drop, and profits will rise.

Modeling and delivering new or modified processes becomes a single integrated corporate task. This is because there is a common standard language – BPEL – to support a unified conversation between the business need and the business solution. It ensures that delays and costs are slashed, while services and business activities in front and back office are optimized.

Business Process Management is a key to winning the benefits of process execution speed and differentiation. It links seamlessly with companies' current Web Services and SOA approach to accelerate their business, to ensure they lead the pace in a dynamic, fast moving world.

References

[1] S. Buraga, L. Alboaie, *Servicii Web. Concepte de bază şi implementări*, Iasi : Polirom, 2006

[2] W3C Working Group, *Web Services Architecture*, 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[3] *Web Services Business Process Execution Language Version 2.0*, 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel - v2.0.pdf

[4] *Reference Architecture for Service Oriented Architecture Version 1.0*, 2008, http://docs.oasis-open.org/soa-rm/soa-ra/v1.0 /soa-ra-pr-01.pdf

[5] *Service-oriented modeling and architecture*, 2004, http://www.ibm.com/developrworks/library/ws-soa-design1/

[6] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, 2001, http://www.w3.org/TR/2001/NOTE-wsdl-200103 15

[7] *Business Process Management in the Finance Sector*, 2004, http://www.oracle.com/industries/financial_services/BPM _WP_final .pdf

Silviu Florin TEODORU has a background in computer science and is interested in business intelligence and data warehouse related issues. He has graduated the Faculty of Economic Cybernetics, Statistics and Informatics from the Bucharest Academy of Economic Studies in 2002. He is currently conducting doctoral research in Economic Informatics at the Academy of Economic Studies. Other fields of interest include data modeling, management information systems, systems architecture and financial services.