Exploring the fundamentals of architecture and services in an SOA, Part 2: The importance of business architecture, model-driven development, and reusing existing assets

Bertrand Portier
Gregory Hodgkinson

23 October 2007

In this second article in the series, get a closer look at architecture—this time at the business level. Learn about model-driven development (MDD), and reusable asset frameworks and types, which can be leveraged when architecting Service-Oriented Architecture (SOA) solutions.

View more content in this series

Why is business architecture important?

In Part 1 of this series you explored the fundamentals of software architecture. In this part, we take a look at a special type of architecture, namely, business architecture. We'll look at why it is important to have architecture at the business level in addition to the IT level, and we'll introduce you to an IBM® business architecture-related technique called component business modeling (CBM).

Defining business architecture

Business architecture is considered by the Open Architecture Group Architecture Framework (TOGAF) to be one-half of enterprise architecture, with the other half defined as software architecture (the latter is also referred to as IT architecture, which comprises data, application, and technology elements. (See Resources for a link to the Wikipedia entry on “software architecture.”)

Core components of a business architecture include:

- Product strategy
- Governance
- People organization
- Business information
- Key business processes
Similarly to software architecture, business architecture follows defined architectural principles, and uses reference architectures and architectural frameworks. (See Resources to read a paper from Carnegie Mellon that describes this in greater detail.)

Please note: The activities described in the following section could also be considered part of other disciplines, such as business strategy or business modeling.

**The importance of business architecture**

Business architecture is important for a number of reasons:

**Business architecture provides a well understood business road map.** Business architecture is the discipline that takes business principles, strategy, and goals as input and defines elements such as business processes, people, or information. Business architecture allows for enterprises to know where they stand today, define where they want to be, and, more important, define a road map that will enable them to get there. In this way, business architecture can be very important to the business even before you consider the benefits to IT.

**Figure 1. Sequence of business process changes**

Business architecture improves IT responsiveness. Businesses need to be able to recognize changes and react to them quickly. Often there is an unacceptable lag between a business change being identified and the required IT change being identified, assessed, and implemented. Business architecture seeks to provide a way of decreasing this lag by providing a linkage between business and IT. As a result, it enforces the responsiveness of an enterprise's IT to business demands, as described in IBM On Demand Business. (See Resources for more information.)

**Business architecture ensures IT systems are relevant.** By first considering the business architecture and describing the linkage to the software architecture, you can ensure that there are no gaps between the two. In other words, by noting how the software architecture addresses business requirements (both tactical and strategic), you can be sure of two things: that all the IT
system features are traced back to a business need and that all business needs for IT system automation have been met. A key goal of business architecture is to demonstrate the business value of the software architecture.

**Business architecture improves IT flexibility.** Perhaps the single most important element of business architecture is that by grounding software architecture (the IT view) in business architecture (the business view), you are creating IT systems that are more likely to adapt to changes in the business rather than inhibiting these changes. So not only will it be easier to see which software components are affected by a business change, it is more likely that the resulting changes will affect less of the overall software architecture, as illustrated in Figure 2.

**Figure 2. Mapping between business and IT improves flexibility**

![Mapping between business and IT improves flexibility](image)

**Component business modeling**

The IBM Component Business Model (CBM) is a strategic method that allows an organization to focus on its core business competencies (the parts of the business that differentiate them from their competitors), see how resources are consumed, and better align business and IT.

Critical to SOA, component business modeling is centered on business-aligned services. It allows for the decomposition of an enterprise as a map of business components (or functional areas) belonging to different accountability levels (levels of responsibility) and business competencies (business domains). Each business component offers or consumes business services to and from other business components (definition of behavior). CBM looks at strategic business architecture elements, such as goals, key performance indicators (KPIs), and business processes. (See Resources to get more information about CMB). Figure 3 gives a visual example of what this might look like by showing the CBM of a DVD rental company.
Reap the benefits of all models: Introducing MDD

Model-driven development (MDD), otherwise known as model-driven engineering (MDE), is a software engineering approach based on models and transformations.

MDD makes use of metadata, metamodels, levels of abstraction, automations, viewpoints, perspectives, modeling languages like the Unified Modeling Language (UML), and modeling frameworks like the Eclipse Modeling Framework (EMF) to increase the amount of automation applied in the software development process. (See Resources a guide to SOA terms to learn more about these terms.) The basic idea is that MDD prescribes using a set of models at various levels of abstraction as the primary engineering artifacts throughout the software development life cycle, and seeks to use the precision in these models to drive automated transformation processes that can generate lower-level models from higher-level models. Figure 4 demonstrates this transformation.

The Rational® Unified Process (RUP) describes a model as "an abstract representation or simulation of a system that provides a complete description of the system from a particular perspective."

In addition, the Object Management Group’s Model-Driven Architecture (MDA) defines a transformation as "converting one model of a system to another model of the same system."
The benefits of MDD

So now that you understand why software and business architecture are important, let’s look at the further benefits gained by applying MDD to these architectures:

- **Perspectives.** Although the software architect should have the breadth necessary to understand a variety of domains and models, most other roles are more specialized. Business analysts, for example, understand business processes or use cases, but not design models. Models are targeted to a group of users that will understand them, and the same system can be represented by distinct models so that it is understood by a variety of people. Models need to be precise and complete, and they also need to provide abstraction, which is the reason why graphical modeling is recommended. This is particularly relevant to the models of design of the software architecture, where it’s recommended using UML and specialized SOA UML profiles.

- **Automations and transformations.** Using MDD allows for quality and productivity gains because models at lower levels of abstractions are generated from models at higher levels of abstraction, which eventually lead to code generation. For example, the SOA service model can be transformed from its UML form to Web Services Definition Language (WSDL) and XML Schema definitions, and also to code such as Java™. Reverse transformations also exist from lower to higher levels of abstraction. For example, a reverse transformation from code to design will allow designers to verify how developers’ code respects their design. Note that such transformation capabilities should be included in Computer-Assisted Design (CAD) tools, such as Rational Software Architect. Tools can support this because of the existence of modeling standards that they implement, such as OMG’s UML and MDA.

- **Traceability.** Traceability goes from lower to higher levels of abstraction to make sure that “requirements” (needs) in the higher-level model are addressed in the lower-level model. For example, with SOA, it is important to “trace” a conceptual service to the elements that it was identified from (such as a business activity), or it is important to trace a design decision to the requirement(s) that drove the decision. Doing that also allows for the analysis of the impact a change in requirements (or higher level of abstraction models) has on all the lower level of abstraction models.

**Leverage existing assets**

Reuse is critical to both software architecture and SOA. Reuse comes in many forms, including software development artifact reuse (at all levels of abstraction), run-time service reuse, and experience and best practice reuse.
Patterns

We will talk in more detail about architectural styles and principles, and more specifically SOA as an architectural style, and the architectural principles at the heart of SOA in forthcoming installments in this series. For now, let's look at reuse of experience and best practice with patterns.

A pattern is a proven solution to a given problem in context and is typically specified with a name, context, problem, and solution description. More important, patterns can be automated using pattern implementations that realize pattern specifications on a given platform. That makes using patterns very attractive and powerful. Rational Software Architect is an example of a tool that provides full support for automated patterns and pattern-based engineering (PBE). (See Resources to a link to IBM developerWorks, which has a area that focuses on pattern solutions.)

Patterns are relevant to architecture and can occur across all of the various levels of abstraction. This means you can get business architecture patterns, high-level design patterns, low-level design patterns, and implementation patterns.

Reuse of all of these types of patterns makes for better quality software, reduced risk, productivity gains, consistency, and better interoperability across the enterprise.

Design patterns are probably the most well known of the types of patterns just mentioned. You may be familiar with, for example, the Gang of Four (GoF) patterns. For SOA, a specific set of software architecture patterns will be described in the next article in this series. CBM (mentioned earlier) has its own set of business architecture patterns that cover a set of common industries.

Reference architectures

A reference architecture is a proven architecture for a particular domain (for example, a data center or call center reference architecture, or a reference architecture for the telecommunications industry). It is a full architecture that has already been created. A reference architecture respects one or more architectural styles and is typically made up of architectural patterns. Think, for example, of layered architectures, model view controller (MVC), client-server, or even SOA. A reference architecture covers the entire architecture by addressing both the functional and the nonfunctional requirements of a domain. For example, a telecommunications reference architecture will pay particular attention to messages payloads. Using a reference architecture allows you to reuse proven architectural practices and thus improve the quality of your architecture. A drawback is they are complete and, as a result, of substantial size, which may not be applicable to smaller projects or systems.

In part 1, you are introduced to an SOA reference architecture, the SOA Solution Stack, which defines the layers and building blocks necessary to correctly architect service-oriented solutions.

Existing software assets

A critical success factor for SOA is the ability to reuse existing assets in the architecture. In this context, existing asset means existing software that is running in the production environment.
(legacy applications) that is specifically not service-oriented (the next section discusses existing software development artifacts). SOA transformation projects allow companies to embark on the SOA journey and gradually transform into an enterprise where services play a central role. They typically involve the service orientation of key software assets that are at the core of the enterprise (legacy application). With SOA transformation projects, before the transformation can occur, it is necessary to identify assets that can potentially play a role in the architecture, and that happens during Existing Asset Analysis (EAA). Existing assets can include IBM CICS® or IMS® transactions, or COBOL programs. These assets have run for years and are part of an enterprise's key differentiators. They should be used and leveraged in the SOA. For example, they can be exposed (wrapped) as services in the architecture. Analysis level techniques exist, also referred to as bottom-up, to surface the assets that would make sense for a particular SOA project. For example, if the SOA project is about an insurance claim business process, there are probably programs or transactions centered on claim data that can be leveraged. Note that this is an area where analysis tools such as IBM WebSphere® Studio Asset Analyzer play a key role to surface relevant information out of a huge production environment. Also note that you should not blindly expose existing assets as services and follow proven processes such as RUP Service-Oriented Modeling and Architecture (SOMA) when identifying services.

Software asset management and asset-based development

Asset-based development (ABD) is a large concept and is worthy of its own series of articles. However, in this section, we simply introduce the ideas behind it.

ABD is a software engineering approach based on using software assets to build solutions. In this context, an asset is not an existing asset as defined above, but any type of software development asset at any level of abstraction, including design models, patterns, and code implementation. Although people have talked about ABD for years and consulting companies dream of successfully adopting an asset-based model as opposed to a labor-based one, very few organizations have successful adopted it. This is not just because of culture and people preferring reinventing the wheel rather than reusing others’ work, but more because of the lack of governance and platform support. Fortunately, companies like IBM are tackling the issue with new products and processes, such as IBM Rational Asset Manager and the IBM Rational Asset Governance and Asset-Based Development process. Assets allow people to invent only what has to be invented and to reuse proven elements elsewhere as much as possible.

ABD provides consistency across an organization (the same requirements lead to the same design decisions because the same assets are used). ABD and MDD are not mutually exclusive. Quite the contrary is true: It is very powerful to package models as assets for others to reuse. (See Resources for more information about the Rational Asset Manager product.)

Service registries and repositories

Two key components of an SOA environment, service registries and repositories enforce the governance that has been put in place by supporting services throughout their life cycles.

One of their key characteristics is that they enable service reuse. Although most companies have not done many SOA projects, the promise of SOA can only be fulfilled if previously developed
services can be reused by future projects. With the application of a forward-looking architectural style, a lot of effort will have been put into making sure that these services are reusable.

Software architects, designers, and implementers should consider exploring service registries when architecting, designing, or implementing an SOA solution (at development time). Service registries provide information such as service specifications and descriptions, usage policies, dependencies and interactions. If a service has to be developed (if it does not already exist), then the software architect should consider submitting (publishing) it to the enterprise’s service registry. Also, service-oriented architecture can make use of service registries and repositories at run time as they support efficient dynamic service interactions with for example endpoint selection, availability management, and policy enforcement. Repositories also provide a management component that can help with service versioning, ensuring that services are used optimally, and meeting service-level agreements. (Please refer to Resources for more information about IBM WebSphere Service Registry and Repository.)

**Summary**

In this article, you learned about business architecture, model-driven development, and reusable assets. You understand now how business architecture provides a road map, improves IT responsiveness and flexibility, and helps make IT systems business-aligned. You also learned about the IBM Component Business Modeling (CBM) technique as well as model-driven development and its benefits in terms of perspectives, automations, and traceability. Finally, you explored reusable assets in the context of SOA and architecture, including patterns, reference architectures, existing software assets, and asset and service registries.