
Min Luo (minl@us.ibm.com)
Executive Architect
IBM

Jin Ge (jinge@cn.ibm.com)
Advisory I/T Architect
IBM

Jia Tan (tanjia@cn.ibm.com)
Staff Software Engineer
IBM

Lei Zhang (zhanglei@cn.ibm.com)
Associate I/T Architect
IBM

Peng Tang (tangpcdl@cn.ibm.com)
Software Engineer
IBM

HeQing Guan (Hawking) (guanheq@cn.ibm.com)
Staff Software Engineer
IBM

Service-oriented architecture (SOA) and model-driven architecture and business development (MDA/D) provide a powerful combination in enabling business flexibility and agility through reuse and asset based industry solutions. This 2 part series will discuss how to take advantage of many proven best software engineering practices, especially the meta-data driven architecture types that are used to model the common structural and in some cases non-structural business entities. In part 1 of the series, we will discuss the software engineering foundation, the proposed approach and address the critical business and technical issues that significantly impact business flexibility, the ability to adapt to changes, and the agility.
1. Introduction

Creating flexible and agile systems that can be easily adapted to meet the ever changing business needs and truly provide expected business value at reduced IT costs have been driving the fundamental shift in how corporations position and utilize IT. This is not an easy task, as demonstrated by our decades of struggling to create and reinvent numerous old and new, stove-piped rigid legacy systems. We have designed, implemented and managed business solutions for similar business problems for all industries around the globe, but we could not easily replicate an existing solution to similar clients, not even able to reuse part of any existing known solutions in some significant fashion. Those solutions were made specific to individual industry, business domain and customer needs at their early design stage, no traceability could be established. Reuse and more importantly managing the ever changing business requirements become almost impossible. The dawn of object-oriented design and programming first brought us fine grained business object models (such as the much expected IBM’s San Francisco Project), then the componentizable, industry specific models and frameworks (e.g., IBM’s IFW/IAA for financial/insurance industries), but the way they were created, modeled, and ultimately their sheer size, complexity and inflexibility to effectively and efficiently transform or adapt them to changes have made it difficult to customize or extend them into specific solutions.

1.1 Two Major Issues for Business Agility and Efficiency

As many industry survey have been suggesting, notably the IBM annual CEO Study (most recent one from 2008[1]), organizations have been inundated by both internal and external changes, and many have been struggling to survive in the new global economy with constant but uncertain changes. Nearly all CEOs have been trying to adapt their business models to satisfy their customers?ever changing business needs, while innovating new and improved business solutions in a much more agile and efficient way. On the other hand, [1] confirmed that the gap between their capability to adapt to those changes and the challenges ahead, the so called "change gap", has been accelerating and almost tripled from 2006 to 2008. Managing changes effectively and turning challenges into real opportunities become ever so important.

Unfortunately, dealing with changes is known to be conceptually challenging, technically difficult, and risky. From the business perspective, poor visibility of changes, the invisible and obscure reasons for the changes, and the lack of effective requirement and change representation and management mechanism make adaptation of an automated program to those changes intractable. On the IT side, there is no formal transformation mechanism to convert requirements and their evolving changes into executable structured programs. To make things even worse, those changes would not be known priori, and would be applied to software over potentially a long period of time, their traceability from source to programs is difficult to establish and maintain, and the ripple effects of necessary modifications could trigger tremendous amount of architectural, programming and potentially deployment effort. The lack of generic, adaptable software representations that can evolve gracefully, and the lack of sufficient and effective software engineering processes and tooling, and of course the accelerating pace of change have further worsened the situation.
Another important finding from [1] that further calls for our approach is that CEOs can no longer just focus on a narrow set of challenges but manage a broader spectrum of business domains with much greater uncertainty, since they have to collaborate with their partners and customers to achieve win-win. This is not an easy undertaking for most organizations, since no one, up to now, really had a reasonable and easy to manipulate view of their business: organization, processes, and especially the impacts of changes in any of those aspects. Again our existing IT systems, and especially the ways we have been designing, developing and integrating existing systems are not aligned with this "global" and "much broader business views and opportunities".

The application of the frequently used decomposition ("divide and conquer") technique to reduce problem complexity is much less successful in combating software complexity as compared with hardware, simply because it is much more difficult to achieve adequate and clean decomposability in business problems or the supporting software systems and the changes that must occur to support the business needs are in general more frequent and many times more drastic. In addition, it is much more difficult to associate or map what's changed and whether other parts of the system may have to be adjusted, some changes could significantly recompose the system that may make things even more difficult.

On the other hand, as we all know, there are tremendous amount of business similarities across industry and business domains. But in the past, we almost never intentionally or successfully utilized such wealth of information in creating, and reusing our knowledge and developed capability. Worse still, almost all enterprises allowed individual departments and project teams to design and implement business functionalities that only served their own needs without much consideration of potential reuse and proper architecture that could support it. As a consequence, there has been an explosion of look-alike components in the enterprises that caused further degradation of the overall software system structures and made business transformation and integration difficult. Worse still, similar models and code segments often are copied and duplicated with some modifications in many places in a program and more devastatingly across applications or even systems that have made tracking and adaptation hard and maintenance costs significantly higher.

The effective reuse and asset-based development requires clear visibility of requirements and their variations or changes. If we can effectively utilize those well known similarities, and be able to represent them in a humanly understandable and machine consumable and reusable fashion, we can not only improve the requirement management, but more importantly, we can transform those business similarities into components or even software similarities at all granularity levels, and remove or reduce redundancy across our systems. This could be achieved by unifying similarity patterns of evolutionary requirements and changes with generic structures, and thus make it feasible to reuse the knowledge gained from past experiences, and be able to effectively implement future ones.

Our proposed approach strives to facilitate the use of those business similarities and turn them into reusable and extensible component or software patterns. Built on top of many well known software engineering practices, we propose a new flexible and extensible business driven, industry aligned framework that can assist CEOs to understand the changes and their impacts, help CIOs to map
out a realistic plan to cope with those changes within the constraints of resources while maximizing the potential utilization of existing investments, and assist architects and developers to save time and energy to transform ever changing requirements into adaptable solutions.

### 1.2 Business Process Optimization Yields Better Benefits

Business process optimization can often yield much greater business benefits than any IT system improvements. In general, the challenge is how we can articulate the business goals and opportunities for potential improvement through both business and system transformation, where and how changes should be managed and processed, and how to quickly translate opportunities into flexible IT solutions that could more effectively support the evolving business needs.

Service-orientated architecture, model-driven business development provides us a new and powerful approach to facilitate the continuous business process improvements or optimization through more effective linkage of business issues with IT realization, especially through reuse and asset based industry solution design and development. But such transformation is difficult to achieve without significant revamping of how processes, components and services are modeled, linked or associated, mapped to IT systems with easy traceability to evaluate impacts if either changes.

Our approach will provide a natural foundation to facilitate the efficient mapping and transformation from business processes to IT capabilities, while utilizing similarities inherent in business processes themselves.

### 1.3 Leveraging Existing IT Investments

The proposed approach will also alleviate the classical legacy issues that have been plaguing enterprises over the past several decades. It would facilitate a much easier transformation of such legacy systems and service-enable them on top of our generic but extensible solution framework. For example, we could leverage many internal IBM assets (for example, IFW, IAA, WCC, SanFrancisco, etc.), many business modeling effort such as the Component Business Modeling (CBM) and industry maps, GBS engagement deliverables for real client projects. External industry reference models and standards (such as ACCORD, IxRetail) could also be a key input to our approach.

This work is based on our collective experience in leading and designing large end-to-end client engagements across many industries, while promoting various industry leading software engineering best practices.

### 2. The Foundation of the Shared Business Services (SBS) Approach An Overview

Major evolutionary developments in software engineering for the past several decades have provided us the instincts and drive for the proposed approach discussed in detail below:
2.1 Object Oriented Analysis and Design (OOAD) [2]

From the monolithic mainframe-application to client-server architecture, we have seen the evolution from system-driven user to user-driven system development, while object-oriented technology had the promise to bring us closer to the business-driven systems.

Objects are an abstraction of real world business concepts, such as Person, Account, or Car and the capturing of the relationships among those entities. It provides a business-user friendly mechanism that bridges the mapping from the real world to the possible system realization. Key techniques in OOAD include abstraction, decomposition, and separation of concerns (behavior vs. static characteristics), modularization and potentially inter-operable components.

Abstraction can hide some of the essential complexity from a programmer, but it is still necessary to fill in all missing details before a complete and executable program can be (automatically) generated from abstract models or program specifications. Only in some narrow application domains, whereby we can make assumptions about the application domain semantics and effectively represent them, would such generator-based solutions be applicable that could dramatically increase programmer productivity.

Decomposition is one of the most common techniques to make a complex problem tractable. We could decompose a complex system into collaborating parts to further simplify the modeling and development processes that could result in more reuse. Those "parts", in reality, are just more coarse-grained objects and are normally called "components". They can be derived as aggregates of more tightly coupled group of low level fine-grained objects. Thus Component based Business software Development (CBD) [3] could further abstract business and technical entities into a level that simplifies the modeling process and development.

In summary, in order to progress, we need to be able to combine business requirements, necessary program design and code into a unified representation with a comprehensible taxonomy, while designing information that is broadly applicable, easily extensible, and with reduced maintenance cost and much improved program understanding.

2.2 Analysis and Design Patterns

In general, patterns[4] are well established and proven solutions to some reoccurring business analysis/design problems. [4] focused on software "design patterns" (such as the object structure, creation, and behavior) that enable architects and developers to successfully reuse existing proven software designs or solution components to create better quality applications more efficiently and without much new effort. [5] introduced the concept of "analysis patterns" that are models of generic business-oriented objects and their interactions expressed as meta-classes with stereotypical attributes and operations and they typify certain situations that can frequently be encountered in more business-oriented modeling. Mastered with those patterns, an software expert can easily differentiate himself from others, since he could recognize those recurring business and technical problems and knows whether there are available proven solutions and how to effectively apply them.

2.2.1 Layered System Architecture
One of the most frequently used architectural pattern is the layered system architecture, where a system is divided into an array of semi-independent layers and each layer provides a service to the layer(s) above and utilizes other services provided by the layer(s) below. The most famous such architecture is the OSI's 7 layer network model [6].

2.3 Archetypes [7]

As discussed previously, business similarity patterns are generally unexploited and the replication of those similarities in various artifacts have created tremendous problems for reuse, maintenance and ability to adapt changes. If we can turn many well known business domain and process similarities into business component or software that intrinsically represent and utilize those similarities, we can further reduce the complexity in delivering solutions. On the other hand, as those similarities remain implicit and dispersed, they could add to program complexity. A key to turning similarities from an adversity to an effective apparatus for software simplification is a technology capable of representing similarity patterns in generic and adaptable form. We also need an approach that enables systematic identification of similarity patterns in evolving business processes and the software systems, unify those similarity patterns with some generic, adaptable program structures. [7] extended the analysis patterns in business domains and coined "business architecture type" as "archetypes" that help us deal with the recurrent and similar business issues. Properly applied, archetypes have been proven ([7]) to simplify designing and programming based on reuse. This will serve as the foundation of our approach.

Archetypes are a specialized type of analysis patterns that can help us deal with the similarity issue and enable us to create universally applicable models at different granularities with sufficient capability to extend or customize. The fundamental assumption behind it is that many (if not most) business entities share lots of common characteristics (could be either static attributes or dynamic behaviors). Therefore, if we could gain "complete knowledge" from "partial understanding" through iteratively augmenting or extending them with new features, as we apply them to similar concepts or new contexts in new projects. As we learn the business problems and build business application systems to support them, we gradually gain enough knowledge such that the aggregated knowledge is sufficient to satisfy many similar needs within or even cross applications/departments/organizations. Archetypes are able to change their behavior to adapt themselves to specific business contexts while their core semantics and taxonomy remain the same. This will make it suitable to provide the required transparency of services in the current wave for SOA, since we do need to exhibit almost the same kind of characteristics from (business) services to promote real reuse and the ease of reuse of those business services. In addition, archetype concept also resonate well with the required business agility and collaboration among business partners, because artifacts based on archetypes already provide the core semantics for business communication, while being able to adapt to changes easily - such capability can be further extended with today's XML based metadata and model-driven development technologies, as we will discuss later.

In UML terms, an archetype is nothing but a "stereo-typed" class that further abstract real world entities into truly reusable and easily extensible analysis/design and implementation artifacts. According to [7], a (business) archetype is a primordial thing that occurs consistently and universally in business domains and business software systems? Here is an example of an
archetype "Person", where all attributes were modeled as <<o>> ("optional") that are intended to be complete (or as complete as possible) and eventually most of the decision a software architect has to do is to select some of those optional features for the given business problem. For simplicity, we do not show any operations associated with it here.

**Figure 1. An example archetype - Person**

![Person Archetype Diagram](image)

If we follow OOAD by applying proper decomposition, aggregation, inheritance techniques, we can extend such a "party" into a more abstract but complete archetype that models the generic and universally applicable concept of an aggregate of such entities as organization, people, etc. with many shared commonalities, and other highly coherent archetypes and their relationships into the so called "archetype patterns", as the `##Party` demonstrated in Figure 2:

**Figure 2. An Archetype Pattern Example - "Party"**

![Party Archetype Pattern Diagram](image)

Representing and specifying those universally applicable entities is only a stepping stone in applying archetypes. Collaborations among them under various business contexts are more crucial. Such collaboration between business archetypes that occurs consistently and universally
in business environments and software systems is called a "business archetype (collaboration) pattern". One simple example for a collaboration among some typical archetypes are depicted in Figure 3.

**Figure 3. An Example Business Archetype Collaboration Pattern**

To improve its ability to adapt to changes, various "variability" mechanism has to be introduced to enable designers to choose (or "configure" as it is called in [7]) or modify the structural and behavioral aspects of those universal-applicable archetypes and patterns. Modern architecture and design tools (such as IBM Rational Software Architect with proper extension plug-ins) allow the capturing of those universal concepts and their related attributes, and their extension and selective use for a specific business domain/problem. We use <<o>>(optional), <<other stereo types>> to represent all possible enumeration of attributes and behaviors and then resolve the variability through a simple languages that supports "pattern configuration rule" (PCL [7]) specification. Such PCLs could easily be expressed in XML and interpreted by the archetype plug-ins to facilitate the customization (selection and extension) of the existing features.

Package vendors have been creating specific business domain applications (such as those for CRM, ERP, etc) that are targeted for a broad industry, and normally provide limited extension technology to address variations to some extend. For example, Siebel provides a rich configuration across all layers in architecture such that 1) Business Objects can be configured by adding new attributes, new elements and creating new relationships to an existing element; 2) Business Components can also be configured by applying changes to related business objects; 3) User facing screens can be configured in a similar fashion. But since their inner-making remains to be a black-box for most of the clients, it has been so difficult and frustrating to integrate, customize, and maintain such systems. In addition, it is very difficult to assess the impacts of introduced changes to either those IT components or business itself with changes. Therefore, reuse at design or service/component level is very hard to achieve. Our proposed approach would help to alleviate this problem.

### 2.4 Service Oriented Architecture (SOA) [8]

With the dawn of SOA, repeatable and potentially reusable business tasks that run across projects, business domains, and organizations have been expected to be designed and implemented as (business) services such that they could easily be exposed, shared and reused, modified, and adapted to changes. With SOA, the concerns of the service providers and consumers are properly separated, thus providing the necessary encapsulation, decoupling, and
modularity required for business and IT agility and flexibility. In addition, such services can be easily aggregated into new composite business application or services if such a need arises.

By delivering business (and some of the IT) capability as services, SOA enables business executives or process owners to drive the definition, creation and execution of such services. The ability to modify and implement these services rapidly allows organizations to compete more effectively. SOA, therefore, revolutionized the traditional business application development paradigm, away from monolithic and rigid application architectures and the associated expensive maintenance to a cost-effective and efficient business process driven architecture.

In order to achieve the maximum benefits, we need to refine the traditional or the modern iterative and incremental software engineering process to facilitate the evolving, business-service-oriented and asset based solutioning process. One such prevailing refined process is IBM's Service-Oriented Modeling and Architecture (SOMA) ([9][8]) and the more recent refinement available from www.ibm.com. The SBS approach would complement SOMA by providing a powerful yet flexible and extensible foundational framework with concrete and reusable model artifacts and enhanced it with easier service identification, realization and the required transformation capability.

2.4.1 SOA Solution Reference Architecture

To take full advantage of SOA, enterprises are advised to establish or adopt proven architecture standards and governance processes to design and manage service assets. One such reference architecture is IBM's SOA Solution Stack Reference Model (S3 [9], Figure 4), a 9-layer model that provides a detailed architectural definition of an SOA solution across all layers. It provides fundamental architectural building blocks, reusable functional elements that one or more components or products can realize as well as the relations among these blocks and among layers, interaction patterns, options, and architectural decisions. S3 not only enumerates the fundamental elements of an SOA solution, but also provides the ingredients and flexibility to model, architect, assemble, deploy, and manage that solution in a way that best fit a particular organization [9].
The nine layers are relatively independent, which allows organizations to select the degree of consumer-provider integration. A SOA solution might exclude a business process layer, for example, and have the consumer and service layers interact directly. The lower layers (services, service components, and operational systems) are provider concerns, while the upper layers (services, business processes, and consumers) are consumer concerns.

A brief explanation for the 9 nine layers are as follows [9]:

1. **Operational systems**
   This layer includes all application assets running in an IT operating environment that supports business activities, whether they are custom built or off the shelf.

2. **Service component**
   This layer contains software components, and each of which are the realization of a service or operation for a service. Service components reflect both the functionality and QoS for each service they represent.

3. **Services**
   The services layer consists of all the services defined within the SOA where a service is just an abstract specification of one or more business-aligned IT functions (operations). Such specialization provides consumers with sufficient detail to invoke the business functions exposed by a service provider.

4. **Business process**
   Organizations can assemble the services exposed in the services layer into composite services that are analogs for significant business processes. In the non-SOA world, these business processes are similar to custom applications. An example is the process of completing and submitting a loan application, which an organization typically performs through a custom application. An SOA in contrast would support that process by creating a composite service that choreographs the information flow among a set of services and human actors.

5. **Consumer**
The consumer layer handles interaction with the user or with other programs in the SOA ecosystem. Through it, an organization can deliver existing IT functions and data to applications or users according to specific user preferences. Consequently, an organization can quickly create the front-end of business processes and composite applications to respond to changes in the marketplace and enable channel-independent access to business processes and services exposed within a given SOA. This in turn allows the selection of rich client-user interfaces, which tend to use technologies that allow Web services to be invoked from the consumer layer. Such interfaces provide a framework that content providers can use to write consistent, interoperable Web services for portals.

6. **Integration**
   This layer integrates primarily layers 2 through 4, making it crucial to an SOA. Its integration capabilities let an organization mediate, route, and transport service requests from the service requester to the correct service provider. Such capabilities include but are not limited to those found in an enterprise service bus (ESB).

7. **QoS**
   The QoS layer lets an SOA signal noncompliance with service qualities in each SOA layer. Thus, it can capture (in an operational sense), monitor, log, and signal non-compliance with nonfunctional requirements that relate to the service qualities. In some cases (such as security), the QoS layer can actually realize such nonfunctional requirements.

8. **Information architecture**
   This layer ensures that an organization includes key considerations affecting data and information architectures, which an organization can also use to create business intelligence through data marts and warehouses. The layer includes stored metadata content.

9. **Governance and policies**
   The governance and policies layer covers all aspects of managing the business operations life cycle. This layer includes all policies from manual governance to WS-Policy (where the services layer and governance and policies layer intersect). It provides guidance and policies for managing service-level agreements, including capacity, performance, security, and monitoring.

### 2.4.2 SOA Value Proposition

As depicted in Figure 5, SOA's greatest potential come from two major perspectives: first of all, the "business services" that get so much emphasized in the last few years of SOA practices can really help bridge the gaps between IT and the business world, while the evolution from more monolithic, and tightly coupled software system architecture to multi-layered, loosely coupled ones enables reuse of existing assets and the seamless (in a "perfect world") integration of business operations across existing operational and business boundaries.
2.5 Model-driven architecture and development (MDA/D) [12] with XML based metadata

MDA/D further elevates the levels of abstraction and reuse. By revolutionizing a set of models spanning from business-centric platform/technology independent (PIM) models to platform-specific and eventually environment/implementation technology dependent models (PSM), MDA provides best practices for modeling the real world business problems and at the same time facilitate the necessary mapping and transformation toward deployable code assets. In this way MDA truly provides design-time interoperability for solution models, at least at the traditional analysis level.

Central to MDA is the notion of creating different models at different levels of abstraction and then linking them together to form an implementation. Some of these models will exist independent of software platforms, while others will be specific to particular platforms. Each model will be expressed using a combination of text and multiple complementary and interrelated modeling artifacts.

Currently, UML is the modeling language of choice. But there is also effort to define domain-specific languages (DSLs) using the MDA framework. Archetypes and the supporting interaction patterns and pattern configuration language could be seen as some kind of DSL, albeit still very informally.

Another important development is the merge of Agile practice [13] and MDA. Agile MDA [14] addresses the potential conflict between MDA and agile methods, which propose to address the problems associated with the "verification gap" (when requirement documents written by business owners or SMEs couldn't be verified to be realizable or executable) by delivering small slices of working code as soon as possible. This working functionality is immediately useful to the customer, who can interact with it; this might result in improved understanding on the customer's part of the system that needs to be built. As these delivery cycles can be short (say, a week or two), the system's development process is able to adapt to changing conditions and deliver just what the customer wants.
3. Proposed Approach for Reusable Common Business Components and Services

Our approach fully leverages and extends the concept of Archetype and the service-oriented MDA/D approach by providing xml-data driven configurability of business objects/components and services. Figure 6 depicts a high level view of the SBS approach. We will explain the major components and how they are established and applied in this section.

**Figure 6. Overview of the SBS Approach**

3.1 Archetype and Archetype Patterns and Model Library (Archetype Model)

As discussed before, one of the fundamental assumption behind the archetypes is that many (if not all) of real world concepts and their relationships are similar and thus can be sufficiently understood, modeled, and selectively used for any specific business context. With the evolutionary approach, we can start small with a subset of the real world entities, and gradually growing our universe. Therefore a library (repository) for various existing archetypes and their patterns is needed.

3.1.1 4-Layer Common Business Components and Services Model

Proper decomposition of the complex business world, especially with the objective of providing a universally applicable business model and architecture, is vital to our proposed approach. As discussed in Section 2, object/component based technology and meta-data driven architecture types (archetypes) enable us to model business entities with an unified and extensible representation with archetypes. In order to address the consequences introduced by abstracting those common elements out of the specific industry model and be able to enable easy and meta-data based transformation, we propose initially to decompose the business components into the following four layers, based on similarity/commonality patterns and their respective inherent relationships:

1. Common
2. Business Domain specific
3. Industry specific
4. Client/Project specific
3.1.2 Layer Definitions and Clarifications

3.1.2.1 Base Common Business Components and Services

This layer contains the foundational and common archetypes that represent most universally applicable business concepts in abstract forms (including attributes and their operations/services that can be exposed as services). It is very general, and doesn't have any business domain or industry specific limitations. Thus they can be broadly reused.

For example, we frequently encounter or have to deal with such concepts as Product, Order, Party, Organization, Account, etc. in many of our business applications.

3.1.2.2 Business (sub) Domain Specific Components and Services

Business domains represent some particular and often tightly inter-related group of business entities that collaboratively work together to fulfill some business functionality. For example, banking and insurance industries often need to address such business areas as product management, financial management, account management, customer relationship management, etc.

You can easily see that many of those business domains are actually applicable across many industries. Customer Relationship Management, for example, is almost universal in every industry.

3.1.2.3 Industry Specific Business Components and Services
To create reusable industry specific solutions, we must also identify the commonality of various business problems and their possible solution artifacts and then properly extend the base and the business domain layers to cover the unique needs of a particular industry.

We will further address the proper modeling between industry layer and the business domain layer in Section 3.1.3.

### 3.1.2.4 Client/Project Specific Business Components and Services

Our dream is to be able to build business solutions using the broadly reusable, archetype-based building blocks from the first three layers. Ideally, any client or project specific components and services can be configured from the artifacts in those layers. Of course, due to the difficulty of obtaining complete knowledge over those common business entities, certain extensions are normally required for any specific realization of a business solution. But as we solve more problems, and harvest our solutions based on the proposed approach, we will grow and evolve our solution assets for broader coverage.

### 3.1.3 Design and Architecture Consideration

#### 3.1.3.1 Industry vs. Domain

One of the earlier design decisions we have encountered in the initial model is whether it is better to swap the positioning of the industry layer and the business domain layer. Following the strict layering, it would be difficult to isolate the bi-directional interactions for those two layers. Therefore, in reality, we will adopt the extended layering for those two layers and we do not need to distinguish which layer is really built on top of the other.

Our original consideration is that industry specific processes and models tend to be more organically related, and business sub-domains within a certain industry (for example) enjoys easier abstraction and specialization, and thus promote better reuse.

#### 3.1.3.2 Fully extended model

To combat all the potential debates and provide a much more solid foundation, we have proposed the bidirectional mapping and traceability for L2 and L3 in such a way that the SBS model framework becomes an extended layered architecture where

1. Both L2 and L3 could use services provided by L1, and provide services to L4;
2. L2 and L3 could provide or consume services to/from each other;
3. Allows different L2/L3 components provide or consume services to/from each other.
Figure 8. Fully Extended 4-Layer Archetype Model

With the extended model, we can significantly improve the flexibility and traceability of the modeling artifacts in those two core layers, and make it much easier to communicate to business users, and also easier to adapt existing industry business domain specific assets into reusable ones.

3.1.3.3 Aligning with S3 Reference Model

Comparing our 4-layer business model and the S3 Reference Architecture, we could easily see that the 4-layered archetype based business components and services model mostly covers the Business Services and Service Component Layer in S3, with some limited coverage and linkage to the Business Process and the Integration.

3.2 SBS Model Extension Framework

As described in Figure 9, to accelerate the SBS model extension process and make it more like an engineering practice, a set of templates are created. In this fashion, new model artifacts at all layers can use the corresponding templates to extend the model at that layer. XML-meta data is used to drive the conversion from variation analysis into newly extended models. Essentially the traditional MDA is extended, where models are decomposed and then specialized into a specific PIM, and eventually a SBS Enabler will transform those models and generate a large percentage of code, as will be explained in section 5.
3.3. Use of the SBS Models and Approach to Solve Real World Business Problems

Now we have some basic understanding about the SBS model and its approach, let's move on to discuss how it could be utilized.

3.3.1 Creating the Base SBS Models (Step 1)

In the ideal world, the Archetype Pattern library would encompass ALL the information we need to model a business problem and provide a customizable and instantiable framework for the given business context. Therefore what a software architect/engineer mostly needs to do is to select/reuse and extend the base archetypes and patterns.

In reality, though, we have to create the base models at all layers from scratch. Based on some business and technical decision on the initial focus areas, we analyzed many existing assets (mostly from some well known/publicized previous projects) and industry standard models from several industries (including governments, banking, insurance, retail, etc) and abstracted fundamental concepts such as Party, Customer, Product, etc. and augmented the base model from [7] substantially. Variation at business domain and industry layers were also represented with required linkages.

Again, not only we established a component model at business level, we also established typical or standard business services that those components would provide or consume in the base model. In addition, we also analyzed and linked in some potential service providers (either existing application, industry solution frameworks such as IFW, or 3rd party packages).
3.3.2 Selecting Proper Archetypes and Patterns and Generating Variations from the Existing Base Models

1. We have created a SBS Analyzer (currently just some Excel spreadsheets with predefined and yet-to-be-collected information) to facilitate the analysis of the business requirements, the matching/mapping of the required business capabilities to what the existing SBS model can readily offer, and what the potential service provider would be able to support. From this tool, we could identify candidate business components and services, and more importantly, the gaps that need to be filled, and finally some realization options (Steps 2-3).

   Currently, the Analyzer consists the following 3 Excel spreadsheets:
   - Required Business Capabilities (features): The best possible information we can utilize here is from the business process decomposition and supporting description (best scenario, but no such luxury for many projects in reality), capability statements and keywords used in use case documents (as for most projects). We then enlist the identified required business functionalities (features) in the Analyzer.
   - SBS Asset/Inventory Map that includes key information such as business component, business services and operations, input and output messages, service candidates from possible providers (existing assets and industry models, 3rd party packages, etc.) and their existing forms of input and output messages
   - SBS Characteristics Map that includes business necessity (based on the given business problem and business/IT context), readiness assessment (both model and component/services), and the needs to transform those assets into SBS compliant services based on potential reuse across industry and business domains.

   It would be easy to create a separate plug-in that integrate all those spreadsheet and the information they contain, and establish the relationship and some navigation/search capability to further assist the problem-to-model matching and selection.

2. With those information, we then proceed to map each required business functionality to some business component(s) and service candidates. Normally we will choose some model components at all layers (Step 4). Best practices in assigning responsibilities to objects can be applied here (for example, the GRASP pattern discussed in [15]. Note that the archetype models may have many optional features (stereotyped as <<o>>, where they could be objects, relationships, attributes and operations at all layers) that we may not need for the given business problem.

3. In the above step, we may find that some business requirements cannot be mapped to existing business components or services, some may need significant enhancements over existing ones. Therefore we will create another spreadsheet to summarize all the gaps and the necessary changes. We then translate the identified gaps and changes into formal XML-based Variations as Pattern Configuration Rules (PCRs) (Step 5-6).

3.3.3 Configuring the Selected Model (Step 7)

After the selection, the selected base model artifacts will be instantiated to become the configured model. In addition, the established specific pattern configuration rules (PCRs) are applied to the instantiated model to meet the specific needs of the problem on hand. All the chosen optional
attributes \(<\langle o \rangle\) in the original SBS model library will become \(<\langle o-y \rangle\)\>, meaning they are needed and thus selected for the given business problem, all other \(<\langle o \rangle\) features will be eliminated.

Some validation for the correctness of the model are also conducted to make sure that all the model artifacts are well formed\[7\].

3.3.4 Extending the Configured Model (Step 8-9)

With the current version of the SBS models (and in the foreseeable future even after it is sufficiently augmented and extended), a generic universally applicable model would never be able to satisfy every individual customer project needs. Therefore, the gap analysis would suggest behavioral or attribute extensions.

1. Behavior Extension: The following two major types of behavior extension could be possible.
   • Adds a new business transaction. For example, a new reminder business object may be added with associated addReminder, getReminder and updateReminder transactions.
   • Customizes or adds new operations/functionality to an existing transaction. For example, send a notification on the postExecute of a particular transaction

2. Attribute Extension: For example, a loan application would require the addition of a risk score attribute for the existing generic Customer object.

At this stage, the joint review of the business and IT team could be conducted to make recommendations and decision on whether the configured/extended SBS models could serve the business needs. Note that currently those extensions would be incorporated into the configured model in a more ad-hoc way, and several rounds of review and adjustments will be necessary.

After all these transformation, what a SBS model really looks like? Essentially, the SBS approach effectively leverages and extends the concept of Archetype to provide configurable business components and services. As depicted in Figure 10, a SBS Model consists of a set of component/object/service models with the addition of variation logic (expressed in xml-based Pattern Configuration Rules) that can be managed by xml-metadata and supported by some tooling extension.

**Figure 10. Base Models + Variation = Configured and Extended SBS Model**
Represented as a pure object model, a sample model for a business component PartyManagement can look like this:

**Figure 11. An Example of Configured and Extended SBS Model**

![Diagram](image)

As you can see, we have created several new stereotypes to define, represent and process those variations.

After those steps, a Layer 4 model specific to the assigned business problem is created.

### 3.3.5 Following Regular MDA Approach to Deployable Components and Services

1. (Step 10) With the Layer 4 problem specific model, we can then follow the normal MDA approach to further extend it into a platform-independent model. In this step, we mainly try to apply both analysis and design patterns to make the model better formed that would increase its potential for reuse.
2. (Step 11 - 12) Here we will conduct review sessions with the client business team and IT team to understand their preference and constraints in existing application or infrastructure capabilities, and transform the PIM into a platform & technology specific model, and eventually into deployable components and services with a large percentage of code automatically generated.

### 3.4 Extended Tooling Support

To support such a SOA based, model and business driven development, existing tooling, especially many necessary transformation and integration capability will be needed as extension on existing tooling (Eclipse or RSA).
As depicted in Figure 11, we designed and implemented a SBS Enabler, an RSA/Eclipse plug-in, that is capable to accept customized SBS instance model and generate respective code modules such as .java, .wsdl, .xsd files and other configuration files. Specifically,

1. SBS Enabler accepts the Layer 4 SBS Instance Model, which is customized from the built-in base SBS Model and technical configuration specification (as discussed in previous section);
2. Then SBS Enabler generate workable code modules for different layers (corresponding to the S3 Model) as the figure depicts.

As a matter of fact, we have used the Enabler to generate around 100,000 line of quality code (“expert level” code, as claimed by [7]) for the 3 early real projects that used SBS as pilots. All of them have passed industry development team testing, and have been successfully integrated into the respective builds and releases, as will be reviewed in Section 5.

**Figure 12. SBS Enabler - From SBS Model and Configuration to Deployable Code**

4. Developing and Transforming Reusable Assets with the SBS Approach

4.1 Creating New SBS Compliant Reusable Assets

Creating new reusable assets following the SBS approach would require only minor modification to our normal iterative and incremental development process. As depicted in Figure 13, several new tasks related to asset discovery (1), gap/variation identification (2), and configuring/ extending the base SBS Model (3-4), harvesting newly developed model elements into the SBS library if they enjoy similar “universal applicability” as existing ones (N+1), and finally we do need to promote and educate our new capabilities to increase the opportunity of reuse.
4.2 Transforming Existing Assets into SBS Compliant Reusable Assets

One of the key drivers for the proposed approach was to tackle a critical issue facing IBM Global Services, namely, how we could transform many proven existing industry applications or assets into reusable forms so that similar projects could significantly reuse them. Unfortunately, most of the existing applications or assets are tied with the original client business needs very early on, tightly coupled with some specific business domains, and solutions or assets themselves are mostly constrained heavily by the original chosen development platform (OS, programming languages, and DBMSs) and the operating environment. Some of those applications or assets were decomposed into components, but many just consisted of a flat set of fine-grained objects, or even worse just a pile of spaghetti-shaped program structure. To make them reusable, we do need to apply OOAD principles to follow our 4-layer SBS models and truly separate concerns of business domains, industry and specific client considerations into manageable and transformable representations. SBS provides not only proven layered structure and templates that makes the transformation, analysis and design tooling support feasible, it also serves as best practices that both business analysts and IT professionals can easily follow.

With the proposed approach, we can leverage existing assets of many formats: UML models, logical or physical data models (in ddl), XSD schemas, and code modules. Such assets could be industry proven solution frameworks (such as IBM’s IFW, IAA, WCC) or deliverables that can be harvested from client engagements (for example, many state social services projects, core banking or insurance projects). To facilitate reuse, transformation of those assets into a representation compliant with our common archetype model libraries is necessary. When we conduct the transformation, we will strive to adhere, or sometimes with minimal extension, the prevailing industry reference models and standards (such as ACCORD, IXRetail, etc.). Making our high value but large industry models and frameworks SBS compliant will overcome the complexity and skill issues and also enable a more effective communication to business executives, architects and developers.

Again, three major tasks need to be conducted to make them SBS compliant:
1. Adapt existing solution/asset models to the 4-layer SBS model using the available templates at all layers, while utilizing existing SBS model elements as much as possible.
2. Configure the selected SBS model elements based on the original solution capabilities, and how existing SBS models could be best utilized to restructure the solution. Same SBS Analyzer can be used for this task.
3. Extend the transformed model with either behavior or attributes, as described in Section 3.3.4.

In some cases, coherent business components and services could be identified from the existing SBS model library that matches closely with the solution/asset capability and program structure. In this case we may just need to extend the SBS model by adding or enhancing relationships, adding new attributes, or adding/enhancing service operations. Other times, though, some new business components need to be created in the SBS model, and we should best leverage the existing solution/asset model. Of course, more work is involved in this case, but the good thing is that after it is done, the SBS models themselves could be significantly enhanced to cover more business domains or industries.

5. Pilot Projects and Promising Results from Real Engagements

The current SBS approach was put into "production" mode after only 6 months of an 9-person team, in addition to some limited early research and modeling effort. Since our Global Business Solution Center (GBSC) assigned individual industry sectors to various geo-based groups, the SBS approach was initially focused on customer information & relationship management, and case management.

5.1 Pilot 1 - Know Your Customer (KYC), a Banking Customer Information System

The purpose of KYC is to provide general applicable business services for accelerating customer on-boarding and maintenance processes with compliance to KYC regulations in banking Industry. After gaining full understanding of the KYC requirements, SBS team educated and led the Industry team to configure the SBS Party and Case domain model. Then in order to make all the artifacts generated by SBS meet the requirements, the SBS team developed and modified some of the service templates for the KYC specific service requirements. Soon after the expected artifacts were generated by SBS, the Industry team started their further development quickly.

The SBS team helped the Industry team to generate the implementation codes, including Entity, DAO and DDL in persistence layer; WSDL, XSD and Java implementation in service layer. All the artifacts are used as FO-KYC project infrastructure by the Industry team.

As a result, the 60% Service Operation of KYC Case Management reused SBS Case Management artifacts and the other 40% new functionalities, in turn contributed to enhance SBS Case Management models at most of the layers. 50% Service Operation of KYC Customer Management reused SBS Customer Mgmt and the other 50% again helped to enhance SBS Case Management.

As Table 1 indicates, SBS generated around 50,000 line of code (LOC) which is 68% of total line of code in FO-KYC:
Table 1. SBS generated artifacts in FO-KYC

<table>
<thead>
<tr>
<th>Generated Artifacts</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Classes for Data Access Services</td>
<td>162</td>
</tr>
<tr>
<td>LOC of Data Access Services</td>
<td>43161</td>
</tr>
<tr>
<td>Java Classes for Business Services</td>
<td>20</td>
</tr>
<tr>
<td>LOC of Business Services</td>
<td>4865</td>
</tr>
</tbody>
</table>

The total LOC of FO-KYC is 69874, while LOC of the generated code is 48026. So the reused percentage is 48026/69874 = 68.7%. We are not going to evaluate the financial benefits in this paper, even though significant savings are obvious. The other benefit for applying our SBS approach is that the cycle to implement and maintain a business component is shortened, and the skill level required decreased significantly.

The following table listed samples of generated artifacts for two services:

Table 2. Sample code artifacts generated by SBS Enabler

<table>
<thead>
<tr>
<th>Type</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd</td>
<td>OnboardingCase.xsd</td>
</tr>
<tr>
<td></td>
<td>Organization.xsd</td>
</tr>
<tr>
<td>wsdI</td>
<td>OnboardingCaseMgmt.wsdl</td>
</tr>
<tr>
<td></td>
<td>OrganizationMgmt.wsdl</td>
</tr>
<tr>
<td>SCA Component Descriptor</td>
<td>OnboardingCaseMgmt.component</td>
</tr>
<tr>
<td></td>
<td>OrganizationMgmt.component</td>
</tr>
<tr>
<td>SCA Service Implementation</td>
<td>OnboardingCaseMgmtImpl.java</td>
</tr>
<tr>
<td></td>
<td>OrganizationMgmtImpl.java</td>
</tr>
<tr>
<td>DAO Class</td>
<td>OnboardingCaseDAO.java</td>
</tr>
<tr>
<td></td>
<td>OrganizationDAO.java</td>
</tr>
<tr>
<td>POJO Class</td>
<td>OnboardingCasePK.java</td>
</tr>
<tr>
<td></td>
<td>OnboardingCase.java</td>
</tr>
<tr>
<td></td>
<td>OrganizationPK.java</td>
</tr>
<tr>
<td></td>
<td>Organization.java</td>
</tr>
</tbody>
</table>

5.2 Pilot 2 - Customer Insight

This application is more focused on case management. As a result, 60% Service Operations reused SBS Case Management artifacts, and about 30% new functionality contributed back to SBS. SBS generated 35,000 line of code for this project, and an estimated of 42% effort was saved.

Table 3. SBS generated artifacts in Customer Insight

<table>
<thead>
<tr>
<th>Generated Artifacts</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Classes for Data Access Services</td>
<td>82</td>
</tr>
<tr>
<td>LOC of Data Access Services</td>
<td>32026</td>
</tr>
<tr>
<td>Java Classes for Business Services</td>
<td>10</td>
</tr>
<tr>
<td>LOC of Business Services</td>
<td>2934</td>
</tr>
</tbody>
</table>
6. Conclusions and Future Directions

6.1 Encouraging Observations
From designing and developing the SBS approach and quickly apply it to some real world customer engagements, we further validated the following:

1. The proposed SBS approach successfully integrated some state-of-the-art technologies, including SOA, meta-data based MDA/D, and more importantly the universally applicable archetypes were proven viable and easily configurable and extensible.
2. The 4-layer common business component and service model, effectively applying and extending the archetype concepts with configurable and transformable extensions that could be packaged as some plug-ins for RSA, where those configurations and extensions can be systematically derived from the SBS Analyser.
3. Such layered common business components and services models may well serve as our long after "business architecture" representation, especially if we could provide further enhancements to link business processes to those model artifacts with traceability.
4. The SBS Enabler completed the end-to-end MDA/D life cycle, since it provides the critical capability to transform SBS PIMs to PSMs (JAVA/J2EE based) and eventually to quality deployable components. Real world application of the SBS approach has obviously exhibited considerable effort, time and cost savings.

With the SBS approach, we are getting real close to create loose-coupled components with high cohesion among those business components, and our application of proven analysis and design patterns would make such models better and more consistent quality models that will eventually help generate quality codes, with much less needed advanced skills. In this sense, we believe the SBS approach will really push current software practice not only more agile, but more critically toward real "engineering" paradigm.

6.2 Some Issues that may call for further refinements
Here are some of the issues identified by early reviewers and adopters:

1. Currently the SBS library is still heavily focused on Customer and Case Management, significant effort is need to expand it into more business domains (such as Product, Sales, etc.).
2. The overall approach needs to be better communicated to business owners/analysts. Especially, the SBS Analyzer should be integrated into RSA, rather as a a set of separate spreadsheets, and the inner working of the SBS Enabler.
3. Identification of common services is still significantly dependent on domain knowledge and modeling skill. Fortunately, such general problem may actually be alleviated by the SBS approach, since practitioners would be able to learn to experiment from a set of integrated but traceable models to verify their concerned business components and services.
4. It might be best to integrate SBS with the current SOMA Modeling Environment (SOMA-ME) plug-in.

6.3 Possible Future Directions

1. Business Processes and SBS service mapping/transformation.
2. Integration with existing database schema and data/message definition, leveraging prevailing industry standards.
3. Business requirements verification/validation
4. Apply the SBS approach to transform high value but complex industry frameworks and assets (such as IFW/IAA).
5. Future Tooling Enhancements over recommended/implemented ones, and they could be used to guide our future tooling alignment and integration effort across several brands.

Part II is on SBS Tooling for Model-Driven Mapping and Transformation. It will discuss several transformation techniques in order to track the evolution of the base, business domain and industry specific artifacts, and more importantly the configurability and transformation necessary to move model from potentially universally applicable form to platform independent form, then to platform and technology dependent representation, including automatic code generation. Some of those realized tooling capability can be used as plug-ins for common modeling/solution architecture tools.

7. Acknowledgements

This work has been guided and funded by Ray Harishankar, IBM Fellow, CTO Global Solutions & Asset Management, since its inception. Without his trust and confidence in the proposed approach, we would still stay at conceptual phase. Several other GBSC-GCG team members contributed to the modeling or tooling or pilot projects. We would also like to thank the project teams who decided to follow the SBS approach and utilize the SBS Models, that eventually created a wonderful win-win for us and the project teams. Liang-jie Zhang of IBM Research also involved some early discussion, reviewed the early draft and gave us some valuable feedback.
Resources

- [4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995
About the authors

Min Luo

Dr. Min Luo is currently an Executive Certified Architect, Strategy & Technology, IBM SWG. He has over 18 years of IT industry experience with more than 14 years of managing large-scale, whole life cycle of software application design and development. He fully understands the impact of various technologies on business, and knows how to effectively and efficiently apply them to solve large scale and complex real world problems. He is an early adopter, advocate and educator of object-oriented analysis and design, component-based, and service-oriented computing and incremental development methodology. He has successfully designed and implemented solutions for transportation, financial, manufacturing industries and large-scale government social services projects. He worked in IBM Global Services? Center of Excellence for Service Oriented Architecture and Web Services and also Enterprise Architecture and Technology as Sr. Certified Architect for over 5 years. Before joining IBM, he served two Fortune 500 transportation companies as Manager, Sr. Manager, and Director, responsible for transportation network planning and technology. As a full time or part time faculty member at several universities, he also has taught undergraduate and graduate Computer Science courses for over 8 years. He obtained his Ph.D. in Electrical and Computer Engineering from Georgia Institute of Technology in 1992 and his M.S. and B.S. in Computer Science in 1987 and 1982 respectively. This work was mostly completed while he served as Chief Architect for IBM's Global Business Solution Center - Greater China Area from July 2007 to March 2008.

Jin Ge

Jin Ge is IBM Certified IT Architect, and his current assignment is the lead architect of Common Enablers in GCG Global Business Solution Center. He have 10 years work experience in IT industry, and he have rich experience on Linux Cluster and SOA.

Jia Tan

Jia Tan currently works for IBM Intelligent Transportation System solution as the developer lead. Prior to this, he has worked as technical lead in Shared-Business Service project to create a domain-based MDA tool to enable agile development. He has engaged in several SOA customer engaging projects in past 4 years. In his spare time, he enjoys reading and playing badminton.
Lei Zhang

Lei Zhang is the lead architect of Road User Charging (RUC) solution of Intelligent Transportation System (ITS) in government industry from Global Business Solution Center. WBSF is one of the important products in GBSC solution stack as part of SOA fundamental. Especially, in the 2007, Lei led a team to join the WBSF V6.1 BETA program, and verified the WBSF 6.1 is much valuable in the government solution development.

Peng Tang

Peng Tang is a software engineer from Global Business Solution Center, IBM, where he is currently building CBE (Common Business Enabler) to enhance the common business service usage among different industrial sectors.

HeQing Guan (Hawking)

HeQing Guan (Hawking) is a Senior Software Engineer on the IBM Global Business Solution Center team. He has a doctorate from the Chinese Academy of Sciences and has worked in the SOA field for almost 5 years.

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)