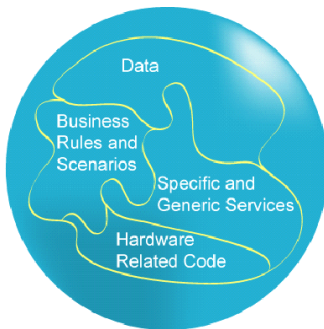


## SOA: “What, Why and How” - Roadmap to better software and knowledge engineering

Yefim (Jeff) Zhuk

Software engineering is an exciting field where constant innovations encourage us (who happen to be in the field) to learn every day new tools and methods. This article continues the subject of integration of software and knowledge engineering that was started in the book “Integration-ready Architecture and Design” [1]. The focus of this article is on Service-Oriented Architecture (SOA), its place in the software evolution and the next step.

### Software Architecture Evolution



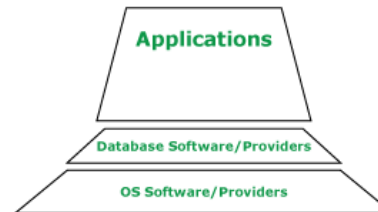
#### In the beginning there was chaos

Do you remember the beginnings of software, when no Operating System (OS) was available? Many highly dependent pieces of code, hardware drivers, data, and business logics, – were collected in a single program to make it work.

I wrote such programs in assembly and even machine code, and later in Fortran, Cobol, C, and Forth programming languages. In the Golden Age of Structural Programming the world of applications was flat and full of functions. I was proud that pieces of my code were copy/pasted by other folks. Some of them called this the magic word “reuse”. (The difference between copy/paste and reuse was not so clear at that time :-).

#### It took the industry about 15-20 years to move to the Object-Oriented paradigm and Layered Architecture

We stopped writing hardware drivers and database support procedures. Operating System and Database vendors took part in the process and allowed most software developers to focus on the application layer.



Today, the application layer is a mix of generic services and business specifics. Any change in business rules or services prompts subject matter experts to start a project to rebuild the application.



Divided by corporate barriers and working under "time-to-market" pressure, we often replicate data and services and produce software that is neither soft nor friendly, lacks flexibility and teamwork skill, and is hardly ready for integration into new environments. Producing "more of the same" and raising the number of product choices, we actually increase entropy and slow down the pace of technology [1].

**Long projects, inflexible, and fast aging applications (that cost a fortune to maintain!)** create more pressure for a better Business - Technology Convergence. Developed in silos, applications often duplicate business functions and, with their growing number of features, become unmanageable and unpredictable monsters. Application development is usually a long process that anyway delivers “almost-ready” and hardly changeable products at the “too late for market”. It takes multiple layers and teams to translate business requirements into Boolean Logic and bake it together with multiple old and some new functions. **The resulting cake is too firm in spite of its name – Software.**

#### What is Service-Oriented Architecture (SOA)?

SOA is a software architecture style that focuses on service components (services) reusable across multiple applications and enterprises. While Service-Oriented Architecture (SOA) is an old concept, current standards and technologies have paved the way to add efficiency to IT and gain strategic advantages for the enterprise to quickly introduce new, or change existing, business features.

### **What is a service?**

A high level service performs a business function. This ideal one-to-one mapping of business functions to their technology implementations (services) makes subject matter experts (SME) the biggest beneficiary of SOA. With SOA, SME has an easy way to re-structure business functions in a new package or/and change just one of functions without rebuilding the application. High level services are usually coarse grained composite services that include calls to other composite services and smaller fine grained services.

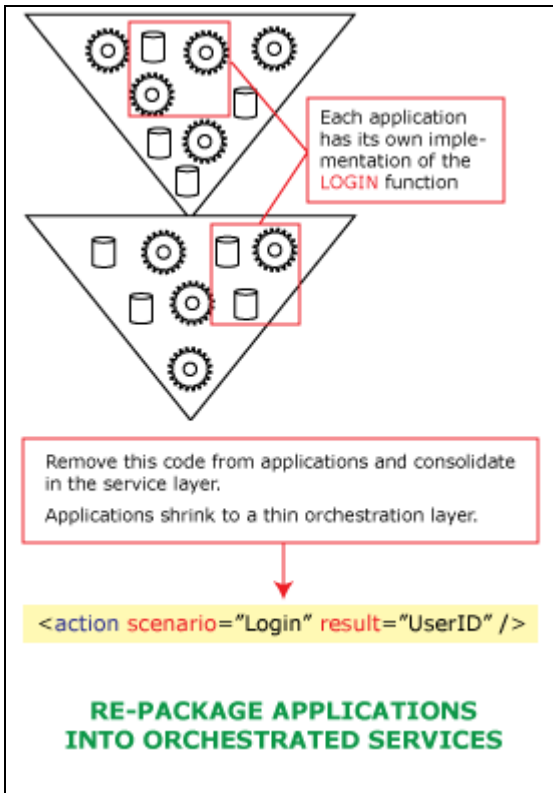
SOA enterprise architects working together with business architects (often same person :- ) make important decisions on internal and external reusability of services. They create a Business-to-Technology map or SOA Dictionary in the process of this analysis. Reusable services are registered at the Enterprise Service Bus (ESB) [2]. **Simple and well defined interface** with minimum parameters and parameter types (text is preferred!) makes services convenient to use. Services deployed just for internal usage are often become valuable enterprise assets easily converted to online external web services. This was a motivation and reword for SOA work in companies like VerySign, Amazon, and etc.

### **Service types and layers**

We can easily distinguish simple and composite service types. It is even more important to recognize different service layers. We'll focus on three layers: Business, like Order or Customer services; Utilities, like Single Sign-On, Search, or Scheduling services, and Data Layer services that can be called from Business or Utilities services (but not directly from applications!). Business services, like "Order" or "Product" services, are usually named after business functions they represent. The "Order" service is usually implemented as a service orchestration or a sequence of composite services. Some of these underlying services, for example, the "Customer" service, can also belong to the business layer, and some others belong to Utilities and Data Layer services.

### **Where does SOA work best?**

SOA fits best into big corporate IT structures with multiple applications built-in-silos that often duplicate their main business functions. SOA will simplify IT infrastructure and accelerate the process of transforming business idea to its implementation. In the transition to SOA, current applications (that represent a mix of business and service logics) will become a thin layer that orchestrates services. Services can be exposed to service consumer programs via XML and to people via portlets and **portal** faces.



## Here is an example of re-packaging applications into service orchestrations

Several applications implement user authentication functionality. **Single Sign On (SSO)** solution should take care of this problem by creating a single service. This service will consolidate functions from current applications. Of course, such consolidation will require collaborative work to define business and security rules, define and consolidate (if possible) data sources, and etc.

Resulting service will be deployed and exposed via the ESB and application code will be replaced by a single line, similar to this yellow line on the left.

It's not necessary (although preferable :- ) to have a single data source. **It's important to have a single service that knows where to go for data.** In the case when a data source or a select statement needs to be changed, the change will be encapsulated in this single service without any impact to calling applications.

Another example of a common enterprise utility service could be the **Search Service**. In our quest for information we often need to go beyond existing data hierarchies looking across enterprise data. Multiple applications (as well as company employees :- ) are potential users of such service.

## Applications will shrink to orchestration scenarios written in BPEL or simpler subset languages.

In the example on the right we can recognize the "Login" Scenario and the more complex composite "Order" scenario. This is just an example that underplays BPEL complexity. The composite "Order" service consists from simpler service scenarios and actually represents a workflow.

Composite scenarios can also include some business logics. Here we come to a related subject: moving away from if/else/case lines (that take significant part of source code) and capture business rules in more natural way.

**The Delegation Design Pattern and Service Component Architecture (SCA)** [3] highly encourages us to separate business and service layers and delegate often changeable business logics to a special "business rule service". This pattern can lead the way to elevate business intelligence (captured in business rules and scenarios) to the top of a current software stack.

## With SOA to Business Service Scenarios

```
<scenario name="Login" type="Service">
  <prompt msg="Your Login name?" result="LogName" />
  <prompt msg="Password:" result="PSW" />
  <action service="Login.CheckPsw(LogName, PSW)" result="UserID" />
</scenario>

<scenario name="Order" type="CompositeService">
  <action scenario="Login" result="UserID" />
  <action scenario="GetOrderData" result="OrderID" />
  <action scenario="PlaceOrder(UserID,OrderID)" />
</scenario>
```

**NEXT: 1) Rules instead of if/else; 2) Extend BPEL with Ontology (NL) and Simplify Business Participation (More at <http://JavaSchool.com>)**  
 SME can quickly re-wire business with new scenarios  
 Reference: <http://jvaschool.com/about/publications.html>

## Rules Engine and Semantic Approach

This design pattern can lead the way to elevate business intelligence (captured in business rules and scenarios) to the top of a current software stack. IBM **ILog's JRules** [4] and some other rules engine products offer nice forms and GUI to express business rules. The next step is to use **Semantic Approach** and allow SME to build SOA Dictionary and converse with rules

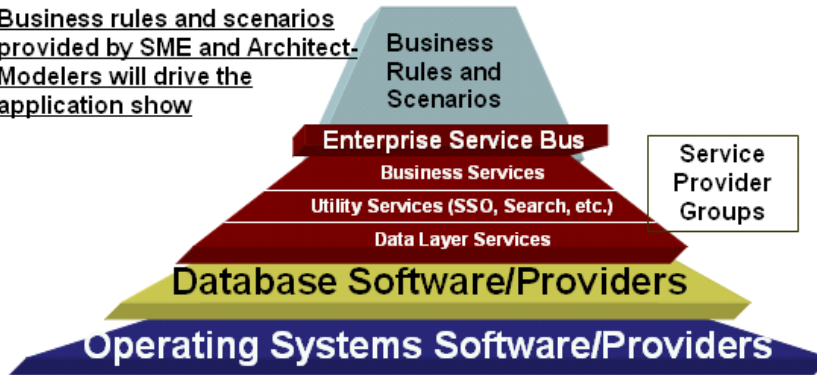
engine using natural language. Expressed by subject matter experts (SME) in natural language (NL) or close to NL terms, **business rules and scenarios will drive the application services**, effectively creating **Knowledge-driven Architecture** [5]. An intelligent rules engine will include a **semantic component** to simplify SME's efforts in creating rules.

## SOA: Separate Business and Services Layers Simplify IT and Easy Business Change

Service Component Architecture (SCA) encourages:

- separate business and service layers
- delegate often changeable business logics to a special "business rule service"

Business rules and scenarios provided by SME and Architect-Modelers will drive the application show



Next Step: Allow subject matter experts (SME) to talk business in NL terms

## Integrated Software and Knowledge Engineering

Powered by a semantic component, **integrated software and knowledge systems** can understand ontology of a business domain and related rules and can help us bridge the barrier between technologists and subject matter experts (SME).

Several software companies including IBM, WebMethods, and BEA initiated semantic approach in their work on SOA tools. In the ontology world, Cycorp, Inc. [6] released its **OpenCyc** and ResearchCyc products, the most powerful ontology repository and reasoning engine available today.

## Recommendations on how to start and continue moving with SOA to Knowledge-driven Architecture

### Phase 1 SOA Introduction

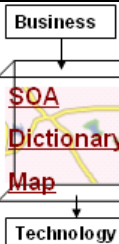
1. Conduct SOA Best Practices Sessions for architects and developers
2. Create SOA Dictionary (start small) to map Business Functions to Services
3. Provide Development and Deployment Policies for Internal and External Business, Utility, and Data Services
4. Establish interdepartmental Enterprise Service (SOA) Governance Group to make policy decisions and Service Provider Groups (within existing development groups) to implement and support enterprise services

### Phase 2 Start Enterprise Transition

1. Prioritize internal and external enterprise services in 3 groups:
  - 1.1 Business Services, like **Customer, Product, Order**, and etc.
  - 1.2 Utility Services like **Single Sign On (SSO), Enterprise Search**, and etc.
  - 1.3 Data Layer Services
2. Establish Enterprise Information Bus and plan to retire similar mechanisms that push and pull data around
3. Establish service access via Enterprise Service Bus and Data Layer Services (called by business services)
4. Integrate service presentation layer via Portal and Portlets

### Phase 3 Extend Enterprise Transition

1. Include all parts of IT and business in the transition
2. Establish Enterprise Knowledge Bus to capture business rules and scenarios for workflows and applications
3. Add semantic component to allow SME to converse with SOA Dictionary during development process and translate requirements into orchestrated services
4. Provide a conversational interface that helps SMEs to capture business rules related to complex data manipulations



Well Monitored and Managed Enterprise Services

Data Service Layer

ESB

Integration with Portal

Semantic Rule Engine and NL conversational Interface

## References:

1. [Integration-Ready Architecture and Design](#), Jeff Zhuk, The book on Software and Knowledge Engineering, Cambridge University Press
2. Enterprise Service Bus, David A. Chappell, First Edition June 2004
3. Service Component Architecture (SCA)  
<http://www.osoa.org/display/Main/Service+Component+Architecture+Home>
4. IBM, ILOG Business Rules, <http://www-01.ibm.com/software/websphere/products/business-rule-management/#>
5. [Knowledge-Driven Architecture](#), Yefim Zhuk, US Patent, Streamlining development and driving applications with business rules & scenarios
6. OpenCyc by Cycorp, Inc., <http://cyc.com>, Commonsense reasoning Open Source knowledgebase