Service-Oriented Architecture and the Next Step

From the book IT of the future

By Jeff Zhuk

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 barely ready for integration into new environments. By 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 and inflexible, fast-aging applications (that cost a fortune to maintain!) create more pressure for a better Business - Technology Convergence. Developed in isolated departments, applications often duplicate business functions and, with their growing number of features, become unmanageable and unpredictable monsters.

Application development is usually a lengthy process that often delivers products which are barely ready and not designed for flexibility at the “too late for market” stage.

Why?

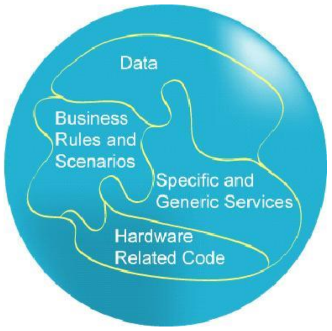
It takes multiple layers and teams to translate business requirements into Boolean Logic and bake it together with many old and new functions.

The resulting cake is too firm in spite of its name – Software.

Software engineering is an exciting field where constant innovations encourage us to learn new tools and methods every day.

This chapter 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 chapter is on Service-Oriented Architecture (SOA), its place in the software evolution and the next steps.

Software Architecture Evolution: In the Beginning There was Chaos



Do you remember the very beginnings of software, when no Operating System (OS) was available? Many highly dependent pieces of code, hardware drivers, data, and business logics were included in each

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 referred to this as 'reusing the code'. The difference between copy/paste and reuse were not so clear at that time :-).

It took the industry several decades to transition to the Object-Oriented paradigm and Layered Architecture.
We stopped writing hardware drivers and database procedures. Operating System and Database vendors took part in the process and allowed most software developers to focus on the application layer.

Instead of writing functions that called other functions, we focused more on data. We presented repeatable objects as classes with their structures. We described the functions of an applications as the messages or methods of these objects.

Years later, after creating massive software packages, including monster applications, we came across a serious problem. Many similar pieces of software were repeated in hundreds and thousands of applications making maintenance cost a hot subject. Any change in business rules or services prompted subject matter experts (SMEs) to start a project to rebuild the application.

What is Service-Oriented Architecture (SOA)?

SOA is a software architecture style that focuses on service components (services) that are 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 (SMEs) the biggest beneficiaries of SOA. With SOA, an SME has an easy way to re-structure business functions in a new package and/or change just one-function without rebuilding the application. High level services are usually course-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 the 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]. A s**imple and well defined interface** with minimum parameters and parameter types (text is preferred!) makes services convenient to use. Services deployed just for internal usage often become valuable enterprise assets easily converted to online external web services. This is a motivation and reward for SOA work in companies like VerySign, Amazon, and etc.

Service types and layers

We can easily distinguish between simple and composite service types, but it is even more important to recognize the different service layers.
We'll focus on three layers here:

- Business, such as Order or Customer services;
- Utilities, such as Single Sign-On, Search, or Scheduling services, and
- Data Layer services that can be called up from Business or Utilities services (but not directly from applications!).

Business services, like Order or Product services, are usually named after the 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?

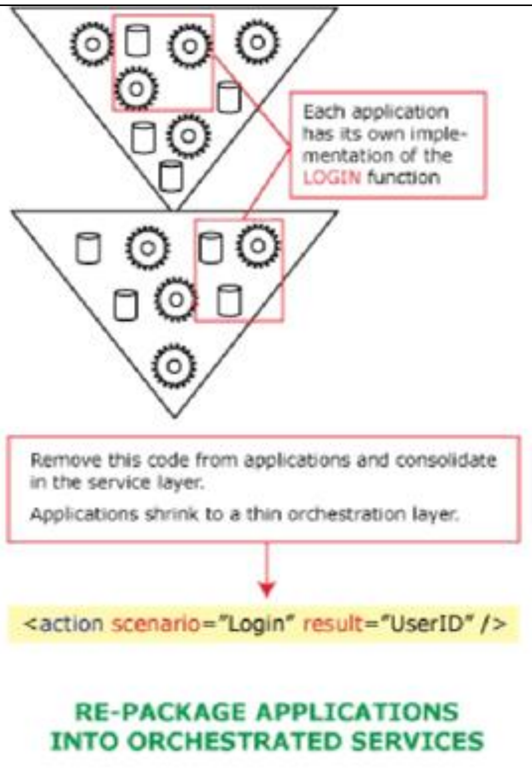
SOA fits best into big corporate IT structures with multiple applications that often duplicate their main business functions. SOA will simplify IT infrastructure and accelerate the process of transforming a business idea into a finished product. In the transition to SOA, current applications that represent a mix of business and service logics will become a thin layer that orchestrates services. Services can be exposed both 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. The **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, etc.

The 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 right.

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 where 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


Each application has its own implementation of the LOGIN function
Remove this code from applications and consolidate in the service layer.
Applications shrink to a thin orchestration layer.
RE-PACKAGE APPLICATIONS INTO ORCHESTRATED SERVICES

```
<action scenario="Login" result="UserID" />
```

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 above 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 of 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 lines (that take up a significant part of source code) to capturing business rules in more natural way.

The Delegation Design Pattern and Service Component Architecture (SCA) [3] teaches 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 the current software stack.

The Next Steps:

Rules Engine and Semantic Approach, Integrated Software and Knowledge Engineering

IBM ILog's JRules [4] and other rules engine products offer business architects convenient ways to express business rules.
The next steps are:

- To use the Semantic Approach and
- Allow an SME building a/the SOA Dictionary to converse with the rules engine using natural language.

Expressed by subject matter experts (SMEs) 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 SMEs' efforts in creating rules.

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

Leading software companies such as IBM, Oracle, Microsoft, WebMethods, and BEA have initiated a semantic approach in their work on SOA tools. In the ontology world, Cycorp, Inc. [6] has released its OpenCyc and ResearchCyc products, a powerful ontology repository and reasoning engine.
A very promising direction is Cognitive Computing, which is in development by several companies

including IBM and Saffron Technologies. Cognitive Computing adds to the Semantic approach the power of Big Data and the lessons we've learned about our brain work.
Stay tuned, we will take a close look at this new world of ideas and tools in the Chapter 4 (Part 4).

Enterprise Architecture Ontology

A common problem for SOA is that the processes of creating requirements, architecture, and development are almost completely disconnected.

There is a need for a semantic bridge that connects these processes in a controllable fashion where a computer system could help tracking initial goals and compare them to delivered solutions. These semi-automated check points must be embedded in the each step of these processes.

Enterprise Architecture is no longer a pure IT discipline but an embedded business tool that can focus on business-centered needs. To support these needs, Enterprise Architecture and its related tools are evolving from Reference-only (text and models) to Interoperable (standard-based) and to Executable (model-based and semantic-enabled). Each new model (according to SOA/FEA) must be connected to a higher level model, enhancing a growing model of enterprise. This approach is in sync with Zachman's Architecture Framework, the most commonly used architecture practice.

Part 3 References:

1. Integration-Ready Architecture and Design, Jeff (Yefim) 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) 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 (Jeff) Zhuk, US Patent, Streamlining development and driving applications with business rules & situational scenarios
6. OpenCyc by Cycorp, Inc., http://cyc.com, Commonsense reasoning Open Source knowledgebase