

Turning reuse into reality

Tan Puay Siew, Lee Eng Wah and Lee Han Boon

The Service Oriented Architecture (SOA) paradigm is based on the concept of "build once, reuse many times". Instead of building each application from scratch, the SOA is about assembling and configuring where possible, and building only when absolutely necessary.

The concept of reuse is not a revolution, but an evolution that started back in the days of object orientation, and has evolved into objects, components and now, services. In SOA, service orientation is the key to assembling applications, and web services can be viewed as an implementation of the SOA paradigm. One essential requirement of these services is that they must be robust. The Wikipedia defines robustness in computer software systems as: "A system that does not break down easily or is not wholly affected by a single application failure." Applying this definition to SOA, it would translate to "a system that does not break down easily or is not wholly affected by a single service point failure."

Figure 1: Implementation of the Oasis Functional Elements Specification in SimTech's Web Services Component Suite



For an application to be considered robust, the application or the process should not fail, and replacement of the failed service with one of similar functionality should be possible.

An important step forward in ensuring robustness and supporting reuse within the SOA is the definition and standardisation of common services that are needed across multiple applications. This is being addressed by the Oasis Framework for Web Services

Implementation Technical Committee (FWSI TC) through its Functional Elements Specification work.

A Functional Element is defined as a building block representing a common reusable functionality for web service-enabled implementations, from an application point of view. Functional Elements are expected to be implemented as reusable components. They are not meant to replace any existing standard or product. In fact, Functional Elements aim to adhere to existing web services standards such as SOAP, WSDL and WS-Security, and also leverage on other specifications like the J2EE specification support for web services and products built on such frameworks.

Instantiated Functional Elements are expected to be a value-added layer above the software products that are already in the market. Furthermore, as the name implies, the focus is on the functionality to be offered by these common services. Therefore, the specification details the features of each Functional Element and where appropriate, the interaction among the Functional Elements.

As the potential spectrum of requirements for the Functional Elements can be extremely broad, the requirements for Functional Elements Specification version 1.0 have been restricted to four categories – Management, Process, Delivery and Security. Based on these four categories, individual requirements are identified based on the role that has been defined for Functional Elements in an application. From the consolidated requirements, potential Functional Elements are identified.

In version 1.0, 16 Functional Elements have been identified. For each of the Functional Elements, the following are specified:

- Motivation, which describes the reason and purpose of the Functional Element.
- Terms used, which provides an explanation on the terminology used in describing the features of the Functional Element.
- Key features, the only normative part of the specification, where the expected functionality of the Functional Element is listed and grouped into mandatory features (that must be implemented) and optional features.
- Inter-dependencies, where interactions with and dependencies on other Functional Elements (if any) are detailed.
- Related technologies and standards, which link the use of related web services standards.
- Use case model and flows, which provide an example of how a Functional Element can be implemented. It shows a complete implementation analysis that practitioners can use as a reference.

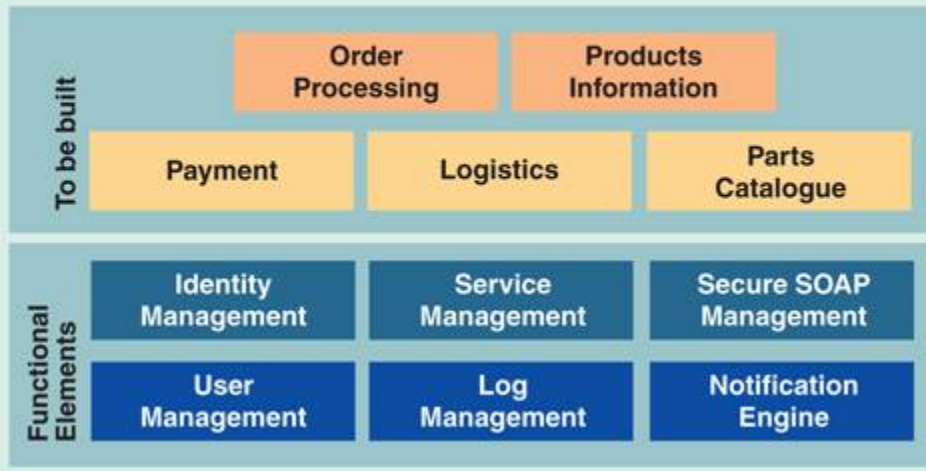
The Oasis Functional Elements Specification is aimed at providing reusable components to facilitate the development of web service-enabled applications. An example is the Web Services Component Suite (WCS) developed by the Singapore Institute of Manufacturing Technology's (SimTech) Web Services Programme.

Figure 1 shows the Core Services inside the WCS that were implemented based on the Oasis Functional Elements Specification version 1.0. Each of the Core Services has been designed to be an independently deployable component, to be used as a building block for web service-enabled applications. Similarly, each Core Service is also designed for reuse. As shown in the diagram, there are web services specific Core Services that address the needs of software implementations specific to web services.

As instantiated Functional Elements are basically independent software components offering specific standardised services, implementers can pick and choose the Functional Elements that are explicitly required by their applications.

Take the example of building an ecommerce application. Six instantiated Functional Elements are used to help accelerate the development of this application, as shown in Figure 2. In this example, the implementer has only to develop the five business-specific services on the upper layer. In short, there is no need to use or deploy all the instantiated Functional Elements into each and every application. Instead, implementers choose only those that are required. Furthermore, based on the SOA concept, these Functional Elements could be part of an application or they could be services hosted by third-party providers or partners.

Figure 2: An example of Functional Elements usage



The same instantiated Functional Elements could be easily reused for other implementations. To illustrate: Suppose the implementer decides that better reliability or a higher service level is required for the "User Management" functionality, it can be easily substituted if alternative "User Management" services that are available also possess the same standardised functionality. Or, due to a change in the business model, the implementer may now need to use a third-party service for its notification functionality.

Again, the fact that the initial notification component is based on standardised functionality, it is easy to switch to another external service that offers equivalent functionality. Likewise, if any of the services fails, a switch to an alternative service of equivalent functionality can be done.

There are known techniques for enabling such switch-overs such as the web services Invocation

Framework and the Dynamic Context Invocation Framework. Therein lies the benefits of having a standardised functionality for such - common services:

- (1) Implementations are not tied-in to any particular service or component,
- (2) Implementations are not prone to single service point failures, and
- (3) The instantiated Functional Elements can be used in multiple implementations. This reusability is further enhanced by the fact that the functionalities are standardised and known, making it easier for architects to design them into their implementations.

- Tan Puay Siew, Lee Eng Wah and Lee Han Boon are members of the team that looks into virtual enterprise integration using web services at the Singapore Institute of Manufacturing Technology (SimTech).

Sidebar

- [Why we need to pay attention to the SOA](#)
- [Usage scenarios](#)

[Printer friendly](#)

[Email this story](#)



[Contacts](#) | [Privacy Policy](#) | [Terms of Access](#) | [Hyperlinking](#)

Copyright IDG Communications (S) Pte. Ltd. 2005 (Co.

Reg No: 199401677G)

A proud member of the [IDG Network of Web sites](#) |

Websites hosted by Pacific Internet

IDG Network: [CIO Asia](#) [Computerworld Malaysia](#) [CIO](#) [Computerworld](#) [CSO](#)
[GamePro](#) [Gamestar.com](#) [Infoworld](#)
[JavaWorld.com](#) [Macworld](#) [Network World](#) [PC Advisor](#) [PC](#)
[World](#) [Playlistmag.com](#)