Subproject Architecture

Architectural Case: Sandvik

2004-05-12

Martin Henkel

SERVIAM-ARC-05

Version 1.2

9 pages

Table of Contents

1 Introduction

This report gives an overview of how Sandvik handles architecture and management of their Web services. The report is based on several presentations and demonstrations that Jan Nilsson, Ulf Domanders, Pontus Gagge and others held at Sandvik 11 may 2004.

Sandvik is well-known as one of the early adopters of the Web service technologies in Sweden. In fact, Sandvik is one of the cases that Microsoft uses as references for Web service implementations. Sandvik uses Web services technologies in several systems, the purpose of using Web service technologies are partially different for some of the implementations. In this short report it is only possible to give an overview of how Sandvik uses Web services. The focus of the report is to highlight the main issues raised at the presentation at Sandvik, as well as documenting design principles that might influence the Serviam project members and those organizations who are about to design and implement systems based on Web services. As usual, this report follows the Serviam architecture description template.

The next chapter describe some of the problems that the usages of Web services aim solve. The solutions applied by Sandvik are thereafter described in the form of a general integration solution, several examples of service use and a short description of how Sandvik administers the services. In addition to this, the chapter “Service design” gives an overview of the different service design principles utilised at Sandvik.

1.1 Problem Overview

As stated earlier, Sandvik was one of the early adopters of Web service technologies. Several issues emphasised the need to utilise Web services:

- An organisation that is distributed globally raised the need to integrate systems on the world-wide level.
- Numerous sub-suppliers and subsidiaries called for a standardised communication protocol.
- A wide set of different business systems required a structured approach to integration. (Take any well-known business system, and it is very likely that some department at Sandvik uses it.)
- A desire to distribute the responsibility of service provisioning made service oriented architectures interesting. The distribution of responsibility means that each system owner is responsible for the systems services.
- The need for external partners to use the same services/systems as internal systems.

In addition to the needs expressed above, the need to alleviate time consuming manual labour (for example exchanging information by fax) was one of the driving factors behind the usage of Web services.

1.2 Solution Overview

The need for integration prompted Sandvik to create its integration architecture. However, the simplicity of Web service technologies and Microsoft .net also made it reasonable to use Web services for creating “plain” distributed applications as well. A sign of the early adoption of Web services is that Sandvik has developed a structured way to manage an internal library of Web services. To elucidate some of the “best practices” applied at Sandvik the description is structured in three sections:

- *Integration hub architecture* – describes the main usage of web services: integration.
- *Examples of service usage* – Gives examples of how web services can be used to solve problems not necessarily related to integration.
- *Web service management* – Describes how Sandvik manages the large amount of services.

1.2.1 The Integration Hub Architecture

For integration purposes Sandvik created an architecture based on message delivery with IBM MQ and an integration hub based on Microsoft Biztalk. The purpose was to create a common "hub" that handles message routing, message translation and message delivery through different channels. Figure 1 gives a *very* simplified view of the architecture. Note that this view is similar to the architecture utilised by SEB Trygg-liv (Henkel, 2004).

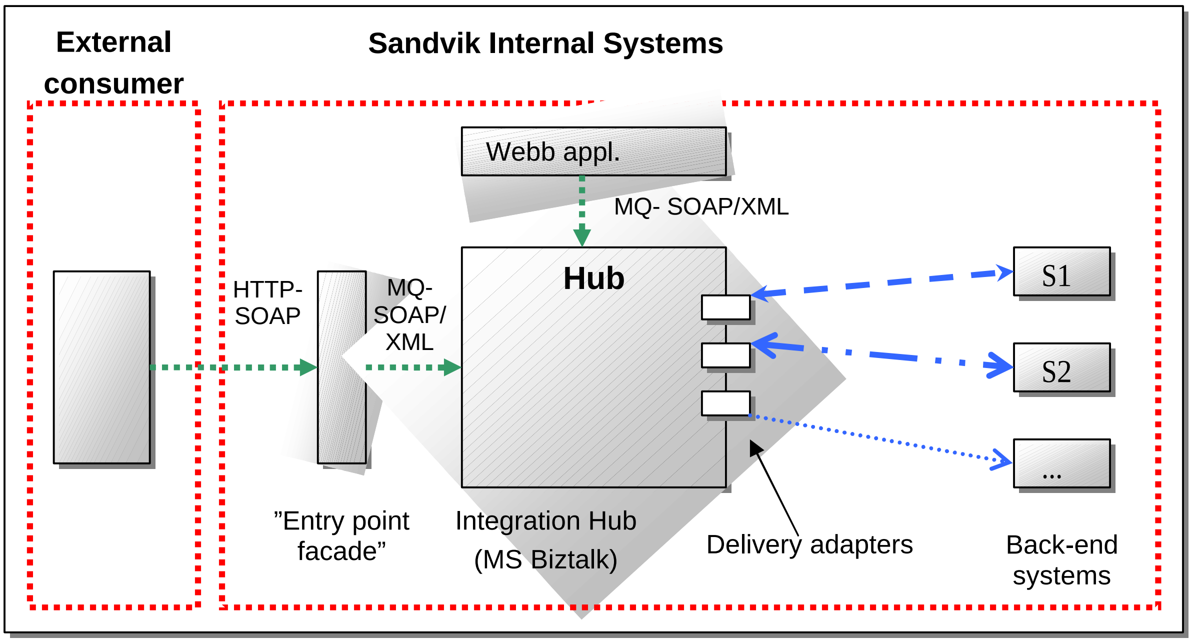

Figure 1, Overview of the integration solution

The integration hub (Biztalk) has the following functionality:

- Message transformation
- Message routing, with the possibility to use content based routing.
- The possibility to use different delivery channels (such as http, EDI, flatfile).
- Message filtering

For an overview of Biztalk functionality, please refer to the Microsoft document "Understanding Biztalk Server 2004" (Chappell, 2004).

Among the contributions to the success of the integration architecture where three things; in-house developed adapters, extensions to the Biztalk admin utilities and a very well defined and managed set of interfaces/messages.

Adapters was needed to integrate the wealth of existing systems, it was necessarily to either buy or develop adapters for transport protocols such as IBM MQ, SAP iDoc and Web services.

Sandvik took the decision to develop most of the adapters. This decision was taken due to the lack of quality of the third-party adapters and the high price of existing adapters. An argument was that for the amount of money spent for buying an adapter it is cheaper, and gives tailored functionality to develop it in-house. In addition to developing transport adapters, special components that handle message routing was developed.

The administrative utilities of Biztalk 2002 lacked a lot of functionality, so Sandvik created its own administrative applications. These applications include functionality for packaging and installing Biztalk components, monitoring and configuration of message queues, and a web-application for gathering statistics about the throughput of the system.

Management and standardisation of messages was early seen as a prerequisite for handling an ever increasing amount of services. Based on the existing ERP systems, a set of request message structures were created and documented in a repository. When integrating new ERP systems into the hub, the systems must implement at least a set of the required interfaces. The creation and management of the common message repository demanded new routines for handling updates to the interface specifications. The management of the common repository is described in further detail in the chapter “Web Service Management”.

1.2.2 Examples of Web Service Usage

Beside its use as an integration technology, Sandvik uses Web service technologies for purposes other than integration in several applications. This chapter describes three cases developed at Sandvik, one regular application (the transport case), one globally accessible service (the quotation system) and one general service (the authorization service).

The transport case: Web services as a transport protocol

Business need: Trucks delivering goods from the warehouse to the assembly line needed to get information on where to pick up the goods, what to pick up and where to deliver the goods.

Technical context: The trucks are equipped with small windows stations, with a simple control pad and a 10 inch screen.

Solution: Two applications were created: one for the coordinator and one for the truck drivers. The coordinator can issue delivery orders to the trucks by using a web based interface. By equipping the trucks with GPRS phones and a small windows application, it was possible to relay the request to the trucks, and display the delivery order to the truck driver. The truck driver can pick a suitable order and perform the delivery. Web service technology (SOAP) were used as a transport protocol between the trucks and the central server.

Purpose of using Web service technologies: In this case SOAP over HTTP was chosen as a communication protocol because it was simpler than to develop a custom TCP/IP based protocol.

Special web service feature: Since GPRS is slow, and charged per delivered byte, every message is sent in a compressed format. This gave a dramatic decrease in (XML) message size. Since the transfer rate is slow, a special “GetWhenSend” design was used. This design enabled every response sent from the server to carry updated delivery order information. For example, when a truck driver picks an order to perform, a notification is sent to the server. The server then replies with an acceptance of the order as well as with a updated list of available delivery orders.

The quotation system: Using and providing services

Business need: Create a common system that supports the requisition and quotation process.

Technical context: There were 15 more or less outdated Quotation systems that the new system needed to replace.

Solution: Create a new quotation application, this application needed to integrate with several back-end systems, as well as provide services to other applications. Thus a service oriented architecture was applied, where the applications server-side logic partitioned into services. This enables future use of the services from other systems.

Purpose of using Web service technologies: The new system needed to use existing Web services, as well as providing its own functionality to other systems.

Special web service feature: The web service operations where designed using Microsoft Diffgrams as carrier of information, this design is further descried in the “Service Design” chapter.

The authorization service: A generic service

Business need: Access to subsidiaries ERP systems is limited to users having a certain role. Thus, many applications need to identify the user, decide which system the user can access, and route the requests to the right systems. Since this functionality is desired in many applications, the need for a centralised handling of authorization and management of a catalogue with user roles and subsidiaries ERP systems was called for.

Solution: Create a central service that stores the lookup table, make this service accessible from all applications. An administrative application was created as well, to enable updates of the lookup table. As an effect of this, the administration of user roles and ERP system was moved out from each application into the central web-based administrative application.

Purpose of using Web service technologies: Provide a generic “horizontal” service that is accessible regardless of platform and programming language.

1.2.3 Web Service Management

As stated earlier, Sandvik uses a common repository (“Generic System Interface”, GSI) to document its generic ERP service interfaces. The repository contains generic interfaces that are meant to be implemented by several back-end ERP systems. Any subsidiary that implements the GSI can hook-up their ERP systems to the integration hub. It is very advantageous for a subsidiary to implement the GSI, since this gives the following benefits:

- Access to Sandviks sales sites, the subsidiaries customers can thus utilise Sandvik web-applications to place orders against the subsidiary that are connected to the integration hub.
- Easy B2B integration, since B2B partners of the subsidiary can connect to the subsidiaries ERP system through the integration hub.

The GSI consists of message definitions specified by using XML schemas. The term “interface” is thus not really applicable in this context, since the messages do not belong to any interface. So, basically the repository can be described as a set of generic messages definitions.

Each of the three business areas at Sandvik have its own repository; two of those have the repository accessible via a web interface. In total, the three repositories contains about 70 message definitions. The repository contains at least the following information for each message:

- A textual description of the message
- An XML schema definition of the request/response message structures.
- Information about the current versions, and prior versions.
- Identification of the systems that can receive the message.

The common repository was created by analysing existing systems, as well as business needs. Experience from this process revealed several "success factors" for building such a repository:

- Involve the business people in schema definition.
- Do not try to create the "perfect" schema, to have a schema that works is more important.
- Design the schema by using the existing concepts/terms, do not start with "a blank paper".
- Avoid to reverse-engineer schemas from databases, commonly the databases contain to much information that are bound to the implementation.
- Create modular schemas to enable reuse.

An important aspect of the repository that Sandvik has dealt with is its maintenance. In order to put the repository in focus for all system development efforts Sandvik have defined the following roles:

Role	**Responsibility**
Process Owner	Responsible for a business process. The process owner can initiate change request if the business process requires extended IT support.
Project Leader	Responsible for development projects. The project leader will have to synchronize all changes that affect the services with the librarian.
Librarian	Responsible for the repository belonging to a business area. All change requests of the schema definitions must go through the librarian. The librarian plans the changes so that they are synchronised with updates to the back-end systems. The librarian also maintains the structure of the repository, to keep every schema well-designed by avoiding overlapping functionality etc. Currently there are three repositories, one for each business area. In the future there will probably be a need for one full-time employee working as a librarian for each business area.

Table 1, Defined roles

2 Service Design

The main design approach applied by Sandvik when creating services is a message-based approach, where the XML schemas that describe the messages are designed first. However, some applications at Sandvik are designed using a procedure-based design approach; other services are designed by employing a constrained interface.

2.1 Message Based Design

A message based design approach is used for all of the interfaces that are put into the common repository. This design has a clear focus on the request and response messages of the services. In fact, the term "interface" is not prominent at all in this design, since the messages that are designed are not grouped into interfaces. The created messages structures convey both the desired operation “CustomerRequest” as well as the operations parameters (“Customer” in this case). Even if the concept of interface is not important in this design, WSDL files can be generated from the XML schemas.

This message based design is used for all services that are a part of the GSI repository.

2.2 Procedure Based Design

Another approach to creating Web service interfaces is to let Microsoft .net generate WSDL and schema files based on regular classes written in for example C#. This approach is applied in the “ScanCert” application at Sandvik. In this case component based design principles can be applied. This means that the operations and the parameter/message structures are designed separately, and that the operations are grouped together according to for example central business concepts.

To let Microsoft .net generate WSDL and Schema documents is very simple, and a rapid way to build Web services. One drawback is that the developer might loose control over the WSDL definitions, since WSDL is generated rather than written manually.

2.3 Constrained Data- Based Interface Design

A third way to build Web service interfaces where utilised to build the Quotation system. In this case a single fixed interface was designed, this interface contains a modify and a get operation. Each of the operations takes a message structure as a parameter. The message structure contains all the information needed to process a request.

This design is very similar to the REST architecture as described by Roy Fielding (Fielding, 2000). The idea is that the operation shows what to do, and the parameter identifies the “resource” that the operation should be applied to. For instance, calling the modify operation with a customer structure as parameter indicates that the customer identified in the parameter should be updated (with the supplied information).

To handle incoming messages to the get and modify operations Sandvik has implemented a generic dispatch routine. When a message arrives at the serverside, the dispatch logic (the “traverser”) breaks down the structure and relays the call to classes implementing the actual business logic. For example, a call to get “modify” operation with a Customer message structure will cause the dispatcher to extract the Customer structure from the message and call the modify operation on the Customer class. Note that each incoming call can cause the dispatch logic to call several classes that implements the business logic.

It is also interesting to note that Sandvik in this case uses Microsoft “Datasets” and “Diffgrams” as operation parameters. Datasets are carriers of structured information. For example, a resultset from a SQL query can easily be converted to a DataSet object by using the Microsoft .net framework. The DataSet object can then be converted (“Serialised”) into XML. Diffgrams are a convenient way of storing changes made to a DataSet. By using the functionality of the Microsoft .net Diffgram, it possible to keep track of the changes made to a DataSet structure. Just like DataSets, Diffgrams can be serialized into XML.

3 Communication

See next chapter for a summary of the technologies used at Sandvik.

4 Web Service Technology

Layer	**Standards**	**Usage in Sandvik**
Service composition/ Process	BPEL4WS	Sandvik is currently not using process coordination/orchestration. However, Sandvik recognises the need to use such facilities in the future.
Composable service assurance	WS-Transaction, WS-Coordination, WS-Reliable Messaging, WS- Security	Currently IBM MQ is used to handle reliable message delivery.
Description	WSDL, XSD, UDDI	XSD is the basis of interface definitions, however, WSDL ca be generated for the interfaces stored in the common repository. UDDI is not used, Sandvik has its own schema repository for internal use.
Messaging	SOAP, XML	SOAP, XML
Transports	HTTP, HTTPS, SMTP	HTTP, IBM MQ

Table 2, Use of the web service stack

5 Success Factors

- Skilled developers enabled Sandvik to create its own solutions when essential features were missing from products.
- A pragmatic approach to schema definition enabled the successful creation of XML schemas that could be utilised by several systems. This pragmatic approach meant that schema definitions were created based on existing system, using a “bottom-up” approach.
- The creation of a central repository with service/message definitions makes it easier for developers to find and use existing services.
- Explicit role definitions enable the repository of services to evolve in a controlled way.
- Recognition that Web services can be used not just for integration made is possible to apply services in “non-integration” applications.
- The possibility to get access to Sandviks “global” business systems (e.g. sales systems) created a big incentive for the subsidiaries to implement standardised Web service interfaces to their ERP systems.

6 References

Henkel, M., “Serviam Architectural case: SEB Trygg-liv”, Document “SERVIAM-ARC-04, 2004-04-02”, Accessible from www.serviam.se.

Chappell, D., “Understanding Biztalk Server 2004”, http://go.microsoft.com/fwlink/?LinkId=21313, February 2004, Accessed 13 May 2004.

Fielding, R. T., “Architectural Styles and the Design of Network-based Software Architectures”, Doctoral dissertation, University of California, Irvine, 2000.