Architectural Case: SEB

2004-04-02

Martin Henkel

SERVIAM-ARC-04

Version 1.5

Table of Contents

1 INTRODUCTION 1

1.1 PROBLEM OVERVIEW 1
1.2 SOLUTION OVERVIEW 1
1.2.1 Development Method 2
1.2.2 The SSEK Protocol 2
1.2.3 The EXA Message Server 2

2 SERVICE DESIGN 3

3 COMMUNICATION 4

4 WEB SERVICE TECHNOLOGY 5

5 SUCCESS FACTORS 5

6 REFERENCES 6

1 Introduction

This document gives an overview of the architecture that SEB Trygg-Liv utilises to provide Web Services to its insurance brokers. The description is based on a presentation by Peder Nateus, Lars Patala and Johan Lidö. The purpose of this document is to give a broad overview of the architecture, so that Serviam project members and others can get new ideas on how to construct their own architecture for Web services. The description follows the Serviam architecture description template.

The rest of this introduction will describe the problem that the architecture aims to solve, and give a brief overview of the solution. The following sections will describe certain aspects of the solution, such as design principles for services and the use of Web service technologies.

1.1 Problem Overview

SEB is, among other things, provider of insurances that are sold by numerous insurance brokers. All these brokers need access to SEB systems in order to market, sell and change insurances for the end customers. In this case the end customers are commonly organisations that provide the insurances as benefits for their employees. From the beginning SEB, the insurance brokers and the end customer where communicating using ordinary mail and fax. Two changes in the business required increased (IT) system support for these tasks. Firstly, the amount of manual transactions where gradually increasing due to changes customer behaviour (frequent updates in the insurances became common). Secondly, the biggest insurance brokers and their biggest end customers required direct connection to SEB systems in order to be able to manage their insurances in an efficient way.

Another aspect that required a new look at the integration possibilities were that the largest insurance brokers needed access to not only SEB systems, but to other providers of insurances as well (such as Skandia). As usual when dealing with insurance information, a communication solution would need to the secure.

All these aspects can be summarized into two key needs:

- The existing IT assets in the form of systems needed to be externally useable.
- An access channel was needed to enable secure access for insurance brokers.

Basically, the first need required an internal architecture that allowed access to existing systems without rewriting or performing extensive changes to the systems. The second need required a communication protocol that was platform and language neutral.

1.2 Solution Overview

As mentioned above SEB was in need of a secure communication protocol and an internal architecture that enabled structured access to their internal systems. Since there are many organisations interested in having a secure connection to their insurance brokers the secure communication protocol was developed together with key players in the insurance domain, the companies Skandia Liv, Länsförsäkringar, Alecta, Aspispronia, Danica, Folksam and SPP. The created protocol is dubbed “Specifikation av säker elektronisk kommunikation mellan aktörer i försäkringsbranschen”, SSEK.

The architectural solution to enable external access to internal systems was to create a centralised message-handling system, the EXA system.

1.2.1 Development Method

The architecture has been developed in a tight cooperation between business experts and IT-architects. The solution has gradually been refined as a result from discussions and experiments. This way of working closely resembles methods and ideas for system development proposed by Agile Methodologies (such as XP, eXtreme Programming) as articulated in the Agile Manifesto (Beck, 2001).

The SSEK protocol is also a result of the work of a small group. This only reinforces the argument that architecture work is best done in small groups with a strong motivation and clear goal.

1.2.2 The SSEK Protocol

The SSEK is an open specification for secure communication based on Web service and XML standards. The fundament of the protocol is to use XML SOAP messages sent over a SSL secured channel.

SSEK is very well described at www.ssek.org (se reference SSEK). So SSEK will not be described in detail in this document. However, to give a hint of the SSEK scope the following SSEK features can be mentioned:

- Identification of how to handle unique transaction identifiers.
- Specification of how to apply XML signatures to digitally sign documents.
- Identifies common error codes (SOAP Faults) that both parties need to handle.
- Definition of four security levels and their mandated use of signatures and SSL client and server certificates.

It should be noted that the above features are not specified in the current Web service standards, thus the SSEK specification is very valuable as a basis for establishing secure Web service connections.

1.2.3 The EXA Message Server

The SSEK specification enables secure message exchange between the insurance broker and SEB. When the messages arrive at SEB they need to be routed to the back-end system. These back-end systems uses different communication protocols, thus before routing the SOAP messages to the appropriate system they need to be transformed into the respective system native communication format. Another problem that needed to be dealt with was that one incoming message might result in numerous messages sent to the back-end systems. Thus, there was a need for a “middle-man” that handled message transformations, message splitting and message joining (this functionality roughly corresponds to the integration patterns “Channel adapter”, “Conten-based router” and “Splitter” (Hohpe, 2004)). Figure 1 shows an overview of the solution.

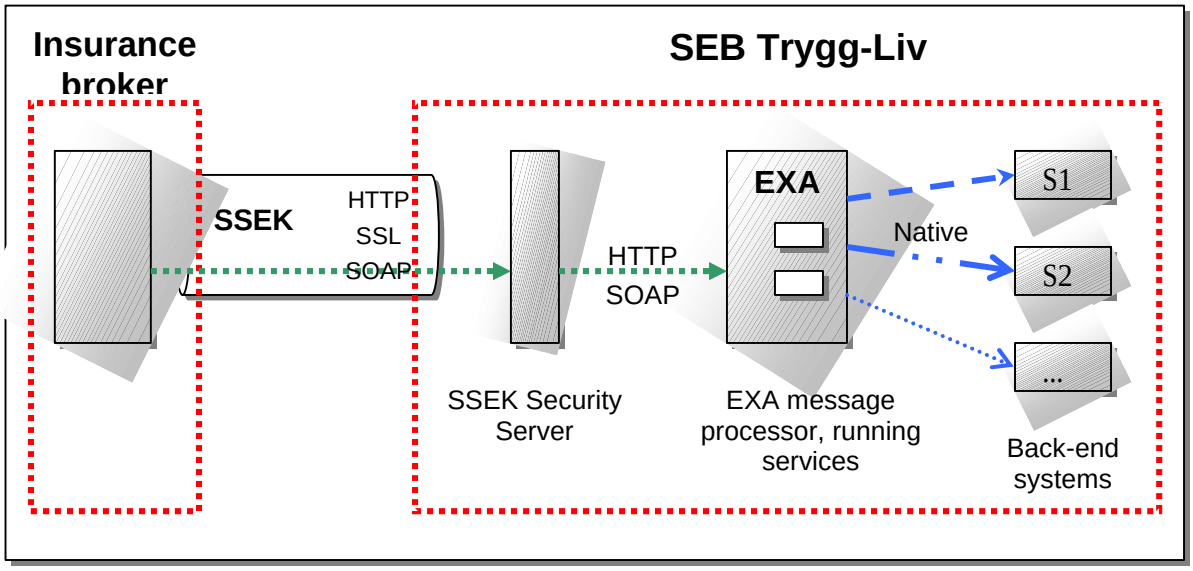

Figure 1, Overview of the solution

Besides acting as an advanced router for external (SOAP) messages the EXA server also acts as an internal communication bridge between internal systems. To summarize, the EXA server has the following main functions:

- Façade for handling incoming SOAP request.
- Communication adapter, the server has functionality to handle the various internal communications protocols.
- Router for requests, passes the request to the appropriate back-end system
- Provides a platform for deploying services that split and merge requests.

The EXA server is a custom ISAPI extension running inside the Microsoft Internet Information Services (IIS) web server. The services that run inside EXA are modules built in C++.

The next chapter takes a closer look at the design of the EXA server and the services running inside it.

2 Service Design

The actual services provided by SEB are executed by the back-end servers. However, the EXA server has the capability of combining the functionality of several back-end systems and provide the combined functionality as a “composite service”. These services can then be called by using SOAP messages.

The scope of the (composite) services provided by EXA is to a large extent dictated by the back-end systems. Thus the services *granularity* is very similar to that provided by the back-end systems. The services running inside EXA have constrained interfaces (Orchard, 2003)(Henkel, 2004). This means that all services have the same methods in their interfaces, se table 1 for a brief an somewhat simplified description of the methods.

Method Name	Basic Functionality
In(XMLMsg) Return: TransactionList	When called, this method prepares the messages that are to be sent to the back-end systems. These messages are put in a transaction list (the return value TransactionList). The method has access to the incoming XML message (via the parameter XMLMsg). Note that the In method does not call the back-end systems directly, instead the EXA server will use the information in TransactionList to call the systems when the In method has finished.
Out() Return: Response	This method is called after the transactions listed in the TransactionList are (successfully) executed. This gives the service an opportunity to modify the returned response.
Sec(SecurityInfo)	This method performs an individual security check for the service. This method gives the service a possibility to check that the security level is high enough for the request. This method is called before the In method. Note that the EXA server performs a generic security check. Thus, the Sec method is only implemented in those services that require special security features.

Table 1, The constrained service interface used by all services.

This interface design requires that the incoming message contains details describing the requested function (i.e. CreateInsurance), along with the information required to perform the function (i.e. CustomerNumber).

Having the same interface design on all service interfaces enables the server to handle the services and request in a generic way. For example, the server can perform authentication on incoming requests prior to passing the request to the “in” method of the service.

3 Communication

The services hosted in the EXA server are synchronous. However, by allowing call-backs to the requestor, the system can provide the same functionality as an asynchronous call. The following sequence describes how asynchronous calls are “simulated” by the server:

1) A request is received.
2) After a security check, The EXA server relays the request to a service via the service In method.
3) The service sends an asynchronous message to a back-end system (via the TransactionList).
4) The service returns a simple “request received” response to the requestor (via the service Out method, called by the EXA server).
5) When finished, the backend system calls the EXA server to deliver the response.
6) The response is sent to the requestor.

This “asynchronous” sequence is used for services that need to call slow back-end systems.

4 Web Service Technology

As mentioned earlier, the SSEK specification relies on Web service standards such as SOAP. The EXA server also uses SOAP for the incoming request.

Currently the architecture does not include a UDDI registry. The reason for this is that there are not enough services to justify the use of UDDI, currently there are about 10 Web services running in the EXA server.

Another Web Service standard that is not used is WSDL. The services all have the same interface, so there little need of WSDL. Instead the services "interface definition" consist of the message structures that each service can handle.

Table 2, below, gives an overview of which parts of the web service technology stack (Donaldsson, 2003) the described solution utilises.

Layer	**Standards**	**The SEB Tryg-liv solution**
Service composition / Process	BPEL4WS	The service implementation running inside the EXA server contains the process flow for a request, implemented in C++.
Composable service assurance	WS-Transaction, WS-Coordination, WS-Reliable Messaging, WS- Security	The SSEK protocol, based on XML-Signatures and SSL is used to create a secure channel. Parts of WS-Security are used in SSEK. Transactions are not used.
Description	WSDL, XSD, UDDI	Not used
Messaging	SOAP, XML	SOAP is used
Transports	HTTP, HTTPS, SMTP	HTTPS is used in SSEK

Table 2, Use of the web service stack

5 Success Factors

The following points summarise why the SEB Trygg-liv solution was successful:

- A small team of business and technical expertise enabled rapid development and refinement of the architecture.
- Focus on technical protocol issues, rather than business aspects, made it possible for competing organisations to jointly define a protocol specification, SSEK.
- A single entry point to all back-end systems enabled a structured approach to exposing existing systems as services.
- Separating generic functionality such as authentication, logging and protocol translations from the service implementation makes it easy to add more services later on. This generic functionally was put into a server framework (the EXA server).

6 References

Beck, K., et al., “The Agile Manifesto”, http://www.agilemanifesto.org/, 2001, accessed 2 April 2004.

Donaldsson, F., Storey, T., Lovering, B., Shewchuck, B., Secure, Reliable, Transacted Web Services: Architecture and Composition, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsOverView.asp, September 2003, Accessed 13 April 2004.

Henkel, M., “Serviam Literature Survey Part III, Web Service Design”, Document “SERVIAM-LIT-03, 2004-02-12”, Accessible from www.serviam.se.

Hophe, G., Woolf, B., “Enterprise Integration Patterens”, Addison-Wesley, 2004.

Orchard, D., “The four Major Constraints to Loosely Coupled Web Services”, Webservices.org 2003, http://www.webservices.org/index.php/article/articleprint/1246/-1/24/, Accessed 15 January 2004.

SSEK Specification, http://www.ssek.org/docs/Spec_av_saker_ekommunikation_i_forsakringsbranschen.pdf (also available in English), 2003, Accessed 2 April 2004.