Designing Software Services for Business Agility

Harald Wesenberg
Leading Advisor Enterprise Architecture
StatoilHydro Corporate Staff IM/IT
E-mail:hews@statoilhydro.com

Introduction

One of the primary concerns when designing software services is identifying the right services to develop. A service at the right granularity with the right functionality can become a crucial asset in the enterprise software domain, while getting the granularity and functionality wrong may lead into a quagmire of interdependent services that are difficult to maintain and further develop whenever new business needs arise.

Within the oil trading domain in Statoil we have worked with services for more than 10 years. The first services where identified and developed in the mid-nineties, and have been reused throughout the domain in numerous places since then.

In this paper I take a look at the service design process we follow and explain how this approach has given us the services we have today.

The concept of a service

The concept of a service varies from company to company, and there is no common definition of a service applicable throughout neither the software industry nor any of the industries the software industry supports. In StatoilHydro, there is no common definition of a service either, and we often talk of services at several abstraction levels and granularities. In the context of this paper, I will use the term service to describe a cohesive grouping of functionality and information that can function independently. A good service is a service that has high cohesiveness and low coupling which can be part of our service based application portfolio where we want to easily recompose services to support business changes in an ever faster cycle.

Find the right services

A key success factor for any service based application is that it consists of the right services. We have used our business processes as basis for finding services, as they often represent valuable knowledge about the functionality the applications must provide. We apply the principles of domain-driven design [1] with focus on strategic domain design in a two-step process. The first step is to identify the bounded contexts of the domain and look at what services are needed for supporting the context relationships.

Figure 1 An example of an order-to-cash process from a generic shopping domain

Dagstuhl Seminar Proceedings 09021
Software Service Engineering
http://drops.dagstuhl.de/opus/volltexte/2009/2044

Figure 1 is an example of an order-to-cash process from a generic shopping domain, supporting both online and physical shopping. Based on such processes, it is natural to look at each sub-process as its own domain, and look at the different services needed to support the relationship with the neighboring domains.

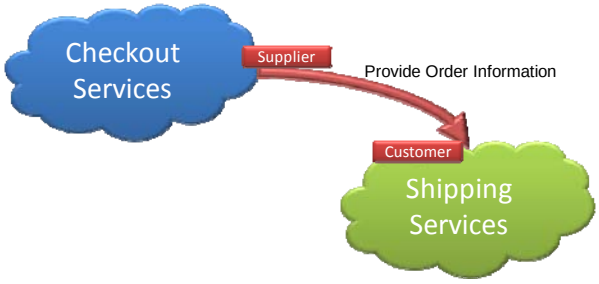

Figure 2 Example of an inter-context service

Figure 2 shows an example of such an inter-context service. The Provide Order Information service supplies the Shipping Services context with order information from the Checkout Services context.

Next it is time to start looking for services within the context. StatoilHydro uses processes and workflows as the main form of communicating operational requirements and best practices throughout the company and thus all contexts have the most important functionality of the context documented in the form of business processes. An example of such a process is shown in Figure 3:

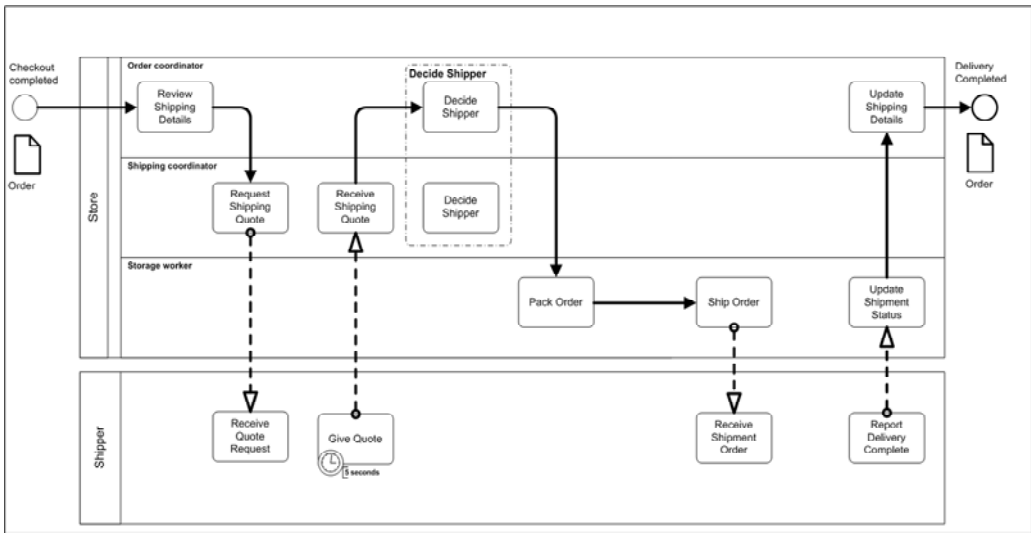

Figure 3 The workflow for the Shipping Services context

Based on the activities in this workflow, a list of potential services can be identified. We aim to reuse as many services as possible, and thus we group potential services into more generic services. Such a grouping is shown in Table 1:

Table 1 A list of inter-context services based on workflow activities

Service Name	Supports Activity
Receive Order	Receive Shipping Details

Service Name	Supports Activity
Request Quote	Request Shipping Quote
Receive Quote Request	Receive Quote Request
Receive Quote	Receive Shipping Quote
Update Order	Decide Shipper, Update Shipping Details
Receive Shipping Updates	Update Shipment Status

Even though we want to have atomic services, we do not want to implement each service as a separate application. Thus we try to group the potential list of services into applications based on how they work on a common set of information concepts. This gives us a balance between the need for atomic services and the need for minimized development overhead in developing and maintaining functionality multiple places. Based on the list of services in Table 1 below, we group the services into two main applications around the information concepts *Order* and *Shipment*. The mapping of services to applications are shown below in Table 2

Table 2 The services grouped into applications

Application	Services
Order Management	Receive Order, Update Order
Shipment Management	Request Quote, Receive Quote, Receive Shipping Updates

It has been customary to use e.g. one order management application to support order management throughout the entire end-to-end process. To achieve the necessary business agility it is necessary to question this practice as it greatly increases the degree of coupling between processes. It is important to keep in mind that we want to be able to develop the different processes independently of each other, and this can be difficult when all the processes are supported by the same application.

The next step

The next step after identifying these service proposals is to add them to a product backlog for development. Usually no more up-front analysis is necessary before the product owner starts prioritizing services and handing them out to the development teams.

Conclusion

In this paper I have shown how we approach service design to get find the right services to support our business processes. This approach is based on the "just enough, just in time" deferring analysis and decisions to as late in the development process as possible where the most amount of knowledge of service requirements is available.

References

[1] Eric Evans, *Domain Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, 2003.