Role of Process Modeling in Software Service Design

Susanne Patig[1], Harald Wesenberg[2]

[1] University of Bern, IWI, Engehaldenstrasse 12, CH-3012 Bern, Switzerland
susanne.patig@iwi.unibe.ch
[2] StatoilHydro ASA, Arkitekt Ebbels veg 10, Rotvoll, NO-7005 Trondheim, Norway
hwes@statoilhydro.com

Abstract. Service-oriented architecture technically facilitates business process management as it enables software to evolve along with changing business processes by simply recomposing software services. From a theoretical point of view it is, thus, natural to take business processes as a starting point for software service design. However, deriving software services strictly top-down from business processes is awkward from a practical point of view: The resulting services are too fine-grained in scope and too vast in number, and particular process control flows become cemented in service orchestrations. In this paper, another approach of software service design is described that, though starting from process models, avoids these drawbacks. The approach is illustrated by a practical example. The presented service design approach has been successfully applied in industry for more than 14 years and enables agile service implementation.

1 Motivation

Building large, mission-critical enterprise systems has always been challenging. During the last decades, these software systems have grown into tightly coupled mastodons that call for extensive efforts to keep in sync with the mutable business.

Service-oriented architecture (SOA) promises to ameliorate this situation: By structuring large software systems into smaller components (*software services*), adapting software to changed business processes amounts to recomposing software services. Consequently, SOA provides the technical foundation for business process management, where software evolves along with continuous process improvements.

Implementing SOA requires the definition of what constitutes a software service. SOA design approaches as sketched in Section 2 suggest that it works best to take business processes as a starting point for strict top-down derivation of software services, which realize process activities. However, practical experience indicates another way of software service design that can be justified by its favorable outcomes and, simultaneously, changes the view on the role of process models in SOA design.

In this paper we outline and generalize the service design approach used by StatoilHydro, which has proven since the mid-90s to facilitate service identification independently of technology and to create highly reusable and stable software services. Especially stability is a prerequisite for agile service development.

This manuscript is published by Springer as part of Lecture Notes in Computer Science Volume 5900 , 2009, pp 420-428 Published version at http://link.springer.com/chapter/10.1007/978-3-642-10383-4_30

Section 3 explains the StatoilHydro approach of software service design abstractly and by means of a real-life example. Section 4 compares our approach with other practice-oriented ones, abstracts from the observations in the StatoilHydro case and draws some general conclusions on software service design for application systems.

2 Current Software Service Design Approaches

Basically, approaches to design software services for SOA fall into two groups: principles-driven approaches and hierarchical ones. *Principle-driven software service design approaches* (e.g., [3]) provide best practices that support SOA design principles such as abstraction, standardized contract, autonomy, statelessness, discoverability etc. Often these recommendations are bundled into patterns (e.g., [4]) whose realization and combination is left to the user.

In contrast, *hierarchical software service design approaches* prescribe steps from some level of abstraction to a set of software services. They end either at the design stage (e.g., [7], [13], [16]) or include further stages of the service life cycle (e.g., [11], [1]). It can be distinguished between *top-down approaches,* which proceed from abstract information at the business level to the technical level of service implementation, and *bottom-up approaches* that increase the level of abstraction during design. *Hybrid approaches* combine bottom-up and top-down strategies (see Section 4).

Starting points for *top-down software service design approaches* are business goals [5], [9], functional business areas [13] or business processes [12], [7], [13], and [6]. *Goals* describe what should be achieved by a software service, *functional areas* are sets of related tasks referring to, e.g., departments or products, and *business processes* additionally consider the roles that perform these tasks as well as the order of tasks (*control flow*). Some of the top-down approaches rely on several types of business information [1]. The common idea of top-down approaches is a strict decomposition of business information into particular and usually fine-grained functions, which constitute service candidates. Process-oriented approaches are often favored as they enable the design of composite software services by orchestrating (atomic) software services according to the control flow between process activities [1], [11].

Current *bottom-up software service design approaches* (e.g., [16]) try to achieve service-orientation by wrapping existing application systems. They use reverse engineering techniques such as clustering to identify cohesive components. The functions these components provide form service candidates.

The direction top-down vs. bottom-up mainly refers to the *identification of service candidates*. Often the initial candidates are *refined* before they are specified: (1) Fine-grained services that have some logical affinity (in terms of, e.g., functions or communication [11]) are *grouped* into coarse-grained services; (2) v*erification* and (3) *validation* check whether or not the candidate software services are conform to the SOA design principles [7], [2] and the stakeholders' needs [1], respectively.

Grouping and refinement can be found in top-down and bottom-up approaches. Only top-down approaches require *asset analysis* to map the identified and refined services either to existing application systems or to service implementation projects [2], [1], [7], [11]. Bottom-up approaches, on the other hand, need an analysis of business requirements and a corresponding *alignment* between IT and business [8].

The final step of *service specification* is always necessary. It defines the service interface (operations and their signatures, given by message types for inbound and outbound messages) and the conversations between services [1], [2], [13], [11].

3 A Practical Case of Software Service Design

3.1 SOA Development Context

StatoilHydro is an integrated oil and gas company with about 30,000 employees in 40 countries and more than 30 years domain experience. Part of the StatoilHydro value chain is the global sales and distribution of crude oil, an area that has been supported by a set of custom-made systems over the last 15 years. During the continued development of the application portfolio, the functional core of these systems was re-engineered as a set of services in the mid-90s, first as PL/SQL interfaces, then as web services. When developing these services (both PL/SQL and web), it was paramount to enable reuse and reduce duplication of code. Since the mid-90s, the services have been expanded, but the initially identified core has remained stable.

3.2 Service Design Process

There was no prescribed method for the design of the initial set of application services (*service inventory* [3]) at StatoilHydro when the project started. An academic ex-post analysis (by interviews, document and system analysis) of the service design process revealed recurring steps, which are depicted as BPMN activities [10] in Fig. 1 below. Section 3.3 illustrates the service design process by an example.

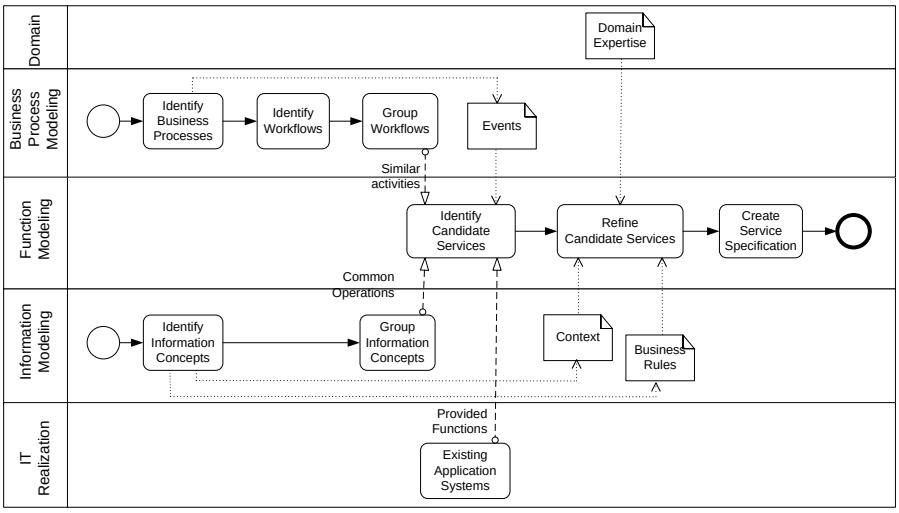

Fig. 1. Software service design process (in BPMN 1.1 [10]).

Application services express the contribution of a software system to a business process on a logical level. They represent functions with high interdependencies in

This manuscript is published by Springer as part of Lecture Notes in Computer Science Volume 5900 , 2009, pp 420-428 Published version at http://link.springer.com/chapter/10.1007/978-3-642-10383-4_30

terms of data usage, user interaction or business rules. The (application) service design process followed a two-pronged approach focusing on business processes/ workflows and information concepts: Workflows were arranged in groups and analyzed to identify candidate services in a top-down way. Simultaneous bottom-up analysis of information concepts helped in generalizing these candidate software services to increase their stability and encourage reuse.

The identified candidate application services were *refined* (grouped or split) by using the following heuristics (for more examples see Section 3.3):

- An application service must refer to the same information concept in the same semantic *context*. For example, the information concept 'Cargo' has distinct interpretations depending on whether it is related to terminal operations, e.g., storing at the port, or to supply operations, e.g., lifting [15]. Hence, separate services are needed.
- An application service must stick to the same *business rules*.
- *Domain expertise* beyond the models must be used to check the candidate application services or to discover new ones.

As for *specification*, StatoilHydro decided to build small service interfaces containing only stable operations. In all, identification and refinement as described brought about three types of software services: (1) *entity services* (mainly CRUD – create, retrieve, update, delete - on information concepts), (2) *task services* (execution of operations more complex than CRUD and strongly guided by business rules) and (3) *technology services* that are not related to business, but needed for the systems to operate (e.g., services that provide a system with data from a data base). Currently, specification guidelines for these service types are prepared in the form of patterns.

3.3 Service Design Example

This section illustrates the service design process described in Section 3.2 by an excerpt from the current business process, workflow and service model of StatoilHydro. All pictures are real-life snapshots of the company's model repository. The models address human readers, contain both manual and IT supported activities and their execution is not automated, but relies on human interaction (*human-centric processes*). All models were already available due to governance requirements.

StatoilHydro has a supply business focusing on the delivery of crude oils and refinery products to customers all over the world. The high-level business process *Supply Operations* consists of the four sub-processes shown in Fig. 2 below.

Fig. 2. Business Process 'Supply Operations'.

Each sub-process is detailed by workflows that are modeled with BPMN [10]. For example, Fig. 3 depicts the workflow of the sub-process 'Schedule Delivery'. The BPMN diagrams and candidate application services of the other sub-processes of Fig. 2 can be found in [14].

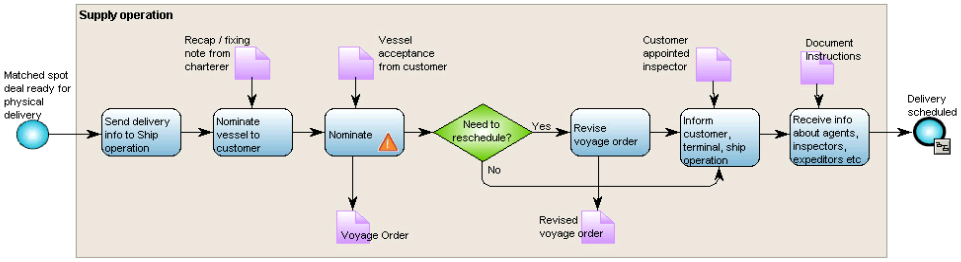

Fig. 3. Sub-process 'Schedule Delivery'.

Top-down and bottom-up software service design were conducted simultaneously: The four sub-processes of the business process 'Supply Operations' form a natural group (*functional area*) to look for similar activities and information concepts. The candidate application services needed in a workflow were gathered by domain experience; Table 1 contains the results for the sub-process 'Schedule Delivery'. From workflow activities, both task and entity services (see Section 3.2) were derived, whereas information concepts initially brought about only entity services. The initial classification of the service types may change during refinement (see below).

Table 1. Identified candidate application services.

Activity / Information Concept	Candidate Application Service	Service Type
Sub-process 'Schedule delivery'		
Send delivery info (information) to ship operations (A = activity)	Send Delivery info	Entity (Delivery)
Nominate vessel to customer (A)	Nominate Vessel, Update Vessel	Task (Nominate), Entity (Vessel)
Revise voyage order (A)	Reschedule Voyage	Task (Reschedule)
Inform customer, terminal... (A)	Send Voyage info	Entity (Voyage)
Receive info about agents, inspectors, expeditors etc. (A)	Receive Agent info, Receive Inspector info, Receive Expeditor info	Entity (Agent/Inspector/Expeditor information[1])
Vessel acceptance (IC = information concept)	Receive Vessel acceptance, Update Voyage	Entity (Vessel acceptance, Voyage)
(Revise) Voyage order (IC)	Send Voyage order	Entity (Voyage order)
Customer appointed inspector (IC)	Receive Inspector info, Update Inspector info	Entity (Inspector information)
Document instructions (IC)	Issue Document instructions	Entity (Document instructions)

After the initial identification of candidate application services for the workflow of each sub-process (see [14]), *refinement* was conducted in workshops with domain experts, software architects and software developers (see Table 2). There are three categories of refinement of candidate application services:

1. *Service type changes*: When more knowledge is gathered and a CRUD operation turns out to involve business rules, then the type of the candidate service is

[1] Information concepts whose names include the term 'information' represent relations between information concepts. Here, the relation exists between 'Cargo' and 'Agent', 'Inspector' etc.

This manuscript is published by Springer as part of Lecture Notes in Computer Science Volume 5900 , 2009, pp 420-428 Published version at http://link.springer.com/chapter/10.1007/978-3-642-10383-4_30

changed from ‘entity’ to ‘task’. For example, ‘Update Vessel’ verifies that some selected vessel meets all legal requirements, which no longer is a CRUD operation.

2. *Grouping*: Candidate application services having the same names or working on similar information concepts in the same context are grouped. For example, initial services such as ‘Update Vessel’ and ‘Update Voyage’ form the service ‘Maintain Cargo’ as they work on the same, more general information concept ‘Cargo’.
3. *Service Discovery*: The refined services ‘Archive electronic documents’ and ‘Receive external documents’ gathered from domain experts demonstrate that some services cannot be derived from workflow activities. Another example is the task service ‘Calculation engine‘ that calculates transport costs, prices and volumes.

Altogether, service refinement has reduced the number of application services handed over to software development teams from initially 33 candidates for the business process ‘Supply Operations’ to 21 [14]. Especially grouping increases reuse: The service ‘Maintain Cargo’ is used seven times in the workflows related to the business process ‘Supply Operations’; further reuse occurs in other functional areas.

Table 2. Refined candidate application services.

Refined Candidate Service	**Identified Candidate Application Services**[2]	**Justification for Refinement**	**Service Type**
Service Type Changes			
Update Vessel	Update Vessel	*Business Rule*: Before updating it must be checked, e.g., whether or not the vessel fulfils all legal requirements.	Task
Issue Document instructions	Issue Document instructions	*Business Rules:* A rules engine determines which document must be sent to which business partner.	Task
Grouping			
Maintain Cargo	Update {Cargo | Delivery | Volume | Transport costs | Gain/loss | Arrival info | ETA | Inspector info}	*Context*: All candidate services relate to attributes of a cargo. The refined service both creates and updates cargoes.	Entity (Cargo)
Receive external documents	Receive {Transport costs | Discharge documents | Cargo documents | ETA | Arrival info | Vessel nomination | Vessel acceptance}	*Domain expertise*: All candidate services relate to the reception of paper documents (by surface mail or fax). The scanned documents must be automatically processed and distributed electronically.	Task
Service Discovery			
Calculation engine	Calculate Transport costs	*Domain expertise:* Prices and volumes must be calculated in several activities.	Task
Archive electronic documents	Not applicable	*Domain expertise*: All legal documents related to a *cargo* must be stored in the corporate electronic archives. Archiving also adds necessary meta data.	Task

[2] To save space, terms common to the names of several service candidates are given outside the brackets ‘{}’ and distinctive parts of the names inside, separated by ‘|’.

4 Generalization and Conclusions

The described approach to design application services has been successfully applied over 14 years in a complex industrial setting. In essence, candidate application services are functions related to (a) data handling (CRUD) of an information concept in the same context (*entity services*), (b) business rules (*task services*) or (c) IT (*technology services*). These service types are gathered both top-down (from activities common to a set of workflows) and bottom-up (from – potentially generalized – information concepts and domain experience). So, a service design process should be *hybrid*. If available, process models can be used as a source of domain knowledge; otherwise, a list of application (business) functions to be supported by IT is sufficient for service design. For human-centric processes, the control flow in the process models should be ignored to not artificially restrict process execution (see [14] for an example).

Grouping of similar activities is essential in application service design to (1) keep the number of designed services small and (2) increase reuse. Grouping requires domain expertise and (preferably object-oriented) information models to guide *generalization*; thus, process modeling must be supplemented by information modeling. Finally, sometimes services must be *split* based on semantic *context* to enable reuse.

Table 3 compares the generalized StatoilHydro and other hybrid approaches of software service design. Shaded activities do not lead to software services.

Table 3. Comparison of hybrid SOA design approaches.

Approach		**SOMA [1]**	**SOAF [2]**	**[6]**	**[7]**	**Statoil**
Candidate Software Services						
Top-down	Goals	Decomposition	—	—	—	—
	Functional Areas		Decomposition	(SOA scope)	—	Similar activities in functional areas
	Business Processes	Activities, processes, CF	Activities of a use case	Activities, CF	Activities	
	Other	BR, variations	—	Stakeholder	Roles	Events
Bottom-up	Existing application	Available functions	Assessed functions	(Implementation)	(Service list)	Available functions
	IC	(CRUD)	—	Only to group	(State changes)	CRUD
	Other	—	—	—	—	IT
Additional design rules				Low data transfer, not time-critical, reuse	SOA principles, service context/ layer, laws, reusability	SOA principles
Refinement	Grouping	Logical affinity	Shared data/ code, scope, reuse	Entity / Task services		Same (generalized) IC or BR
	Splitting	—	—	—	—	IC context
	Checks	VA	—	—	VE, VA, overlap, feasibility	—
Specification		X	X	X	Specification schema	Small interface, stable operations

BR: Business rule, CF: Control flow, IC: Information concept, VA: validation, VF: Verification

This manuscript is published by Springer as part of Lecture Notes in Computer Science Volume 5900 , 2009, pp 420-428 Published version at http://link.springer.com/chapter/10.1007/978-3-642-10383-4_30

The design of the StatoilHydro application services has proven to be stable for more than 14 years. As we look forward, agile software development is seeing wide-spread adoption across the software industry. Agile software development places less emphasis on upfront analysis and more emphasis on deferring decisions until more knowledge is available. In this setting, identifying stable application services at the right granularity before software service development starts is even more important, as identifying the wrong services can lead to extensive rework. We believe that the software service design process outlined in this paper has shown to facilitate service identification while, at the same time, significantly reducing the need for upfront analysis, making it immensely suitable for agile development projects.

References

1. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: A method for developing service-oriented solutions. IBM Systems Journal, 47 (2008) 377 -396
2. Erradi, A., Anand, S., Kulkarni, N.: SOAF: An Architectural Framework for Service Definition and Realization. In: Proc. SCC 2006. IEEE, Los Alamitos (2006)
3. Erl, T.: SOA Principles of Service Design. Prentice Hall, Upper Saddle River et al. (2008)
4. Erl, T.: SOA Design Patterns. Prentice Hall, Upper Saddle River et al. (2008)
5. Kaabi, R.S., Souveyet, C., Rolland, C.: Eliciting service composition in a goal driven manner. In: Aiello, M. et al. (eds.): Proc. ICSOC 2004. ACM Press, New York (2004) 305-308
6. Klose, K., Knackstedt, R., Beverungen, D.: Identification of Services - A Stakeholder-based Approach to SOA development and its application in the area of production planning. In: Österle, H. et al. (eds.): Proc. ECIS 2007. St. Gallen (2007) 1802-1814
7. Kohlmann, F.: Service identification and design - A Hybrid approach in decomposed financial value chains. In: Reichert, M. et al.: Proc. EMISA '07. Koellen, Bonn (2007) 205-218
8. Lämmer, A., Eggert, S., Gronau, N.: A Procedure Model for SOA-Based Integration of Enterprise Systems. Int. Journal of Enterprise Information Systems, 4 (2008) 1-12
9. Levi, K., Arsanjani, A.: A Goal-driven Approach to Enterprise Component Identification and Specification. Communications of the ACM 45 (2002) 45-52
10. Object Management Group (OMG): Business Process Modeling Notation, V1.1. OMG Document Number: formal/20012-01-17, http://www.omg.org/docs/formal/012-01-17.pdf
11. Papazoglou, M.P., van den Heuvel, W.-J.: Service-oriented design and development methodology. Int. Journal of Web Engineering and Technology 2 (2006) 412-442
12. Papazoglou, M.P., Yang, J.: Design Methodology for Web Services and Business Processes. In: Buchmann, A. et al. (eds.): Proc. TES 2002. LNCS, Vol. 2444, Springer, Berlin et al. (2002) 175-233
13. Quartel, D., Dijkman, R., van Sinderen, M.: Methodological support for service-oriented design with ISDL. In: Aiello, M. et al. (eds.): Proc. ICSOC 2004. ACM Press, New York (2004) 1-10
14 Patig, S., Wesenberg, H.: Role of Process Modeling in Software Service Design. Preprint No. 219, University of Bern, May 2009.
15. Wesenberg, H., Landre, E., Rønneberg, H.: Using domain-driven design to evaluate commercial off-the-shelf software. In: Proc. Companion OOPSLA 2006. ACM Press, New York (2006) 824-829
16. Zhang, Z., Liu, R., Yang, H.: Service Identification and Packaging in Service Oriented Reengineering. In: Chu, W.C. et al. (eds.): Proc. SEKE '2005. Skokie (2005) 620-625