Dynamic Method for Service-Oriented Software Design

Karhunen, Harri
University of Kuopio, Department of Computer Science, P.O.B. 1627, FIN-70211 Kuopio, Finland
harri.karhunen@uku.fi

Abstract. Current software development is based on business vision, business processes, and software architecture. It is not possible to develop systems from the scratch. We strive to adaptive, cost-effective software systems, which are easy to develop and which offer all the services stakeholders need. In this paper we present Service-Oriented Software Engineering (SOSE) component framework, which helps to fulfil business visions. Our method is strongly use case driven and offers as a solution service and business components as design models for the system development.
The SOSE component framework is based on best practices: component-based approach, well proven development management methods, and a set of architectural decisions, which utilize architectural patterns, service-oriented architecture (SOA), and model-driven architecture (MDA). The goal of our research is to develop design methods and tools, which support software engineers to build reusable and adaptive systems with cost-efficient system development.

Introduction

Business decision-makers want adaptive, flexible and cost-effective solutions. Software projects compete with other projects inside a company when investments are made. Cost-benefit analysis is important when selecting development targets. The business domain and requirements of that domain need to be modelled in an appropriate way as well as existing systems.

Software engineers need an efficient developing method, and a path from business requirements to software implementation. Besides these methods need to be communicative so that business and software developers can understand and cooperate with each other. The process from business requirements to software is a transformation process where business requirements are an input and the software architecture is an output.

The architectural decisions in software development are always a compromise; a balance between quality requirements of the customers. Therefore the architectural evaluation and architectural transformations are needed (Bosch, 2000). It is easier to make improvements to early sketches than to implementations. Besides at the early phase of the development process the communication between business and software is easier to synchronize because the communication language is general enough for both participants to understand.

Problems between business and software engineering are based on their different focus and experience of environmental issues. First, business engineering has business case focus: business objectives, business processes and business communication between stakeholders and partners. Software engineering on the other hand is reflecting business requirements against software and technical issues. Forces in software development are driving in two directions: how to achieve business goals and how to implement these goals in practice so that non-functional requirements are satisfied. Second, the business approach has two different approaches a dynamic and static approach. The dynamic approach is concerned about business processes and modelling of those processes. The static approach considers business concerns and creates a business data model often referred as a business object model.

However, methods in software engineering have shortcomings. They take account too much on existing information system implementation while the focus should be more on the business vision. Because the information system is a set of old development and implementation decisions, current implementations, and future plans of software systems can constraint information system development and thus reject business visions to come true. Further the methods do not facilitate interoperability of distributed systems. They try too much on distributed objects, which make software and its development complex and unmanageable. Therefore, an efficient, flexible, practical, and sufficient simple methodology is needed.

We suggest that the methodology is based on state-of-art technology solutions: service-oriented approach (SOA) (Chappell, 2003) and best practices (Herzum et al., 2000).

The Service-Oriented Software Engineering (SOSE) framework offers two approaches to business development: business case analysis and specification, and service and component specification. The latter is emphasized in this article. The SOSE component framework referred later as the SOSE framework model is based on SOA and best practices and makes compromises with different business requirements with different ages in order to make the development and integration process easier offering means for co-operation in business network. It is adaptive because current and target systems are combined to appropriate extent. It is reusable because developed solutions act as a framework for new projects. Major architectural decisions are made at the requirement phase. Thus, development costs can be evaluated at a very early stage of the development.

The SOSE framework model is based on: functionality and information separating those lines and mapping them together in order to build a business service component. In practice this separation of functionality and information line is not tight instead lines are intertwined during the development process. However, the process and information separation offers flexibility in defining business entities and processes.

The goals of our research are to create a design framework, develop methods for software engineering, create documented template solutions for stakeholders, and create a flexible component model for distributed communication.

The rest of this paper is organized as follows. The SOSE analysis and design method is described in next section. First, the utilized theory is reviewed then the overall research methodology is presented. Second, the SOSE framework is presented. The research methodology is presented, and the actual framework is described. The process consists of defining business cases using use cases. A set of business service components constructs system level components. Finally we analyse and design those business service components and present them as a set of design components using 4-layer architecture, components, and adapters to legacy systems. The Experimental results section is a description of the business processes and a case study in the electricity market, in which SOSE framework was utilized. Discussion section emphasizes the nature of business activities and method evaluation. In Conclusion our results and future plans to validate the method in practise are summarized.

Methods: SOSE Analysis and Design Method Framework

This section describes methodologies of other researchers and SOSE framework with its methodological background. The research uses a constructive research methodology where the initial framework model is constructed and evaluated using a case example with a pilot design and implementation. The framework uses use case models as a tool in transition from business case to software implementation. The initial business service component is created by mapping use cases as services presented in a service map and adding business information models to the service map. Iterations are needed in order to create a proper component model with a right size of granularity and quality of service (QoS).

Methods of other researchers

Different kinds of stakeholders take part in the development project which has many phases guided in several methodologies. One of those uses a business use case model for composing business domain context (Maciaszek et al., 2005 p. 202). The requirements of the business domain are modelled with use case models (Jacobson et al., 1998 p.126). The use case-driven approach has two views: top-down and outside-in (Raistrick et al., 2004 p. 10). The outside-in approach means that the system with its environment operates in the same context and interacts with each other. The use case model specifies the functionality inside the problem domain. Each use case describes an interaction (manual or automatic) between stakeholders and a system. Use cases have dependencies with each other, such as use case implementations and presentations in the information system.

The software system consists of different items such as architectural specifications, entities, and blocks of code. A component-centric approach is based on the use of components working together (Herzum et al., 2000). The *Business Component Factory* methodology defines, designs, and constructs granularity components which satisfy business requirements. A top-down approach is used when the business domain is analysed. The component architecture is a set of hierarchical components. The *Catalysis* approach methodology for component-based development offers means, how to define platform independent component kit architecture with component connectors and a path from business goals to the program code (D'Souza, 1998). The *Service-oriented architecture (SOA)* expands this point of view.

The diversity of information systems is reality. The *goal-driven* approach for enterprise component identification and specification (Goal Framework) analyzes and structures the business model to the enterprise components (Levi et al., 2002).

This goal-driven approach is both component-based and service-oriented. The business processes are accomplished as services. The enterprise components offer high level services.

Services offer a way to build cooperation inside an enterprise and between enterprises. *Enterprise Application Integration (EAI)* is used inside an enterprise and *Enterprise Service Architecture (ESA)* between enterprises (Woods, 2003). EAI covers all the means which are necessary to build a cohesive system by combining a set of functions using middle tier programs. EAI covers both the system and process integration (Ruh et al., 2001). ESA emphasizes actions needed for value add of the business transactions. As a tool with business partners cooperation ESA uses a service-oriented approach.

The service-oriented approach in a business chain between enterprises comprises an *Enterprise Service Bus (ESB)*, which is a connector between request and response actions in business processes (Bond, 2001). The Bus consists of a set of business level service components, which all together provide the needed media for different business processes. Standards-based integration platform combines messaging, web services, data transformation, and intelligent routing in a highly distributed, event-driven *Service Oriented Architecture (SOA)* (Chappell, 2003). SOA is an architectural style for building software applications which promote loose coupling between components. The services are components with published interfaces. The consumers can dynamically discover these interfaces. The services are interoperable. Thus SOA is a design principle (Panda, 2005).

SOSE research objectives

In this research we develop and improve the former methods according to following goals. The first goal is to build a *design framework* which is dynamic, flexible, cost-effective, and applicable for several heterogeneous design problems. It will form a path from business requirements to the implementation of software covering the whole lifecycle of software development projects.

The second goal is to develop methods for software engineering and software business with ICT companies and their customers. In our research, we use two approaches for service oriented software engineering: the *SOSE business framework* and *SOSE component framework*. The business framework is concerned with a path from business requirements to business cases and the component framework with the path from the business cases to the information system definition. In this paper we concentrate on the component framework.

The third goal is to create template solutions with documentation for stakeholders. The purpose of this is to help the system development through information. The documentation includes business model, system requirements based on the business model, use case model, test cases based on requirements, service model, and component model using *Model-Driven Architecture (MDA)*. MDA has two models: *platform independent model (PIM)* and *platform specific model*

(PSM). The PIM model captures business requirements and software specification. The PSM model uses the PIM model and adds platform specific features to the model (Raistrick et.al., 2004). The fourth goal is to create a component model for distributed communication supporting interoperability and software integration

Our modelling techniques are based on several researchers' methodologies (Jacobson, Herzum, Arsanjani, Levi and D'Souza). The *SOSE component framework* describes business requirements and main processes using use cases, which form services. Services consist of a set of processes, sub-services or legacy systems. Each service is modelled as a service level design component. We refer to this design component as a *business service component*. The software system is constructed out of these business service components. Each service orchestrates processes and uses other services when needed. The SOSE framework offers a very flexible component model, which can be computational or manual. It is designed for dynamic business processes where flexibility is needed between partners. For instance, a supply chain with a changing partnership needs flexibility.

SOSE research methodology

Our research approach obeys both a professional work practice approach by giving a recipe to developers (Iivari et al., 1998) and object-oriented approaches (Jacobson et al. 1998). The constructive research based on existing state-of-art research and technology is used here. Based on problem domain formulation the initial model was built. The model is tested using pilot studies and improvements are made using iterations. Finally the source business domain is implemented as software code by applying the SOSE framework. The evaluation of this model is presented using a theory testing case-research where first a research is presented, second a single implementation is presented, and third in a discussion and conclusion sections the SOSE model is evaluated (Järvinen, 1999).

A business operation is a combination of procedure and data, which can be classified in five categories: business processes, task-optimized user interfaces, use-case driven methodologies, Model-View-Controller pattern and component-based development (Pawson et al., 2002). We use business processes when we create a value chain for main processes, task-optimizer user interfaces when we have actor-centric approach creating a process map based on main processes (described as a value chain), use-case driven methodologies when specifying processes and making transformations for processes in order to create business services. Because our objective is to create implementations we use components-based software development in order to create design components for various implementations.

The SOSE framework applies Kruchten's (Kruchten 1995) ideas of 4+1. The SOSE method is use-case centric and also process-driven while business processes are described using use-cases. The use case constructs a business service

component, which is a set of service process, entity and auxiliary components. Business data is considered as a set of entity components and all other resources as components of their own type. The logical view is a set of requirements.

The non-functional requirements such as security, performance, scalability and throughput are emphasized too. While many of these requirements have crossover features they are applied as aspects in process specifications. In order to satisfy non-functional requirements architectural transformations are needed. Thus, the architecture transformation process goes side by side of the development process from initial architectural decisions to the mature final architecture. During transformation process each quality attribute is examined and the effects to the architecture are taken into account. The component-oriented approach manages the physical view and deployment view. The architecture is a set of different views. Together these views the architectural framework is composed. SOSE makes possible to reuse well-proven practices.

SOSE Framework implementation

The framework design and implementation include: identification of the domain specific key use cases, set of actors interacting with framework, patterns and proven solutions, key interfaces and components, default implementation providing extension points, documenting, and final test case creation work (Larsen, 1999). The SOSE framework uses methodology based on a top-down approach where the business domain is divided into business processes and business data concepts.

The Every business process or use case is fulfilled using a service or services where services are designed and implemented using a business service component. The information architecture is designed using business data concepts which are used by services. Thus, the SOSE framework has a use case-, component-, and service-oriented approach (see Figure 1).

At beginning of the analysis and design process the business domain area is considered as one problem domain. The work list consists of:

- Move from the business requirements to the system requirements.
- Divide the business domain into subsets. Specify main processes and find crossover properties among these.
- Model refinement using several iterations.
- Create the system's use case maps: the current system, the goal system, and the mediating path. A use case map is a hierarchical collection use cases of different granularities. The map is used to describe system level preferred as *system map*, business level preferred as *service map* or even smaller use case collections.

– Divide the system into dynamic and static subsystems making the division on process and data centric way like, and take the time aspect is into account which processes or data are going to change during considered period of time.

The SOSE framework process defines as its main objective design components which satisfy both business and information system requirements as a goal to describe the information system for the future needs. First, the business process was defined using the *functionality line* specifying the service structure. Secondly, the *information line* is followed where business concepts are used when business entities are created. Both of these lines are accomplished intertwined. Thirdly, the *final design phase* consists of mapping the business entities to design components. And last, in the refinement phase the communication, interfaces, and dependencies between components are defined.

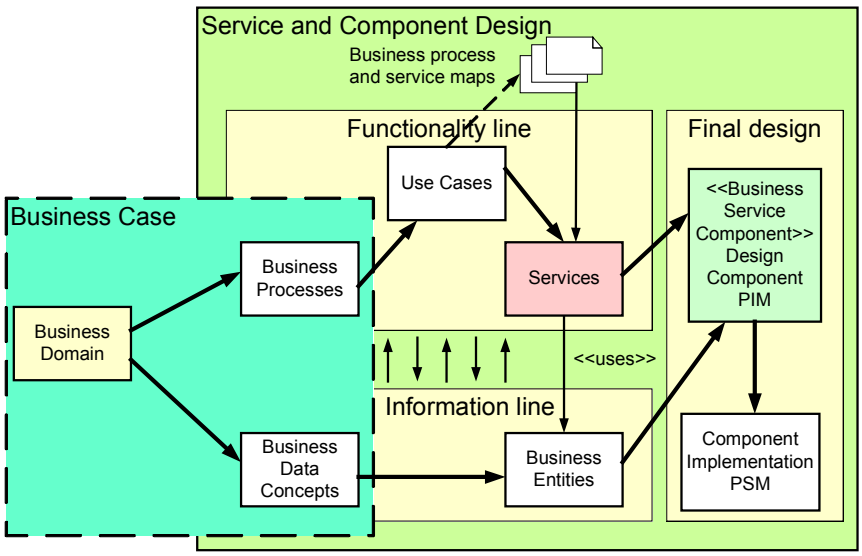

Figure 1: SOSE Framework Developing Diagram.

Functionality line

The problem area is divided up into smaller business processes, where the whole system is considered as a set of sub-systems, which consists of smaller business component level units - use cases, which describe human to machine, machine to machine, and human to human interaction and are the basis for service definitions.

The SOSE framework phases for modelling use cases are:

– The requirements of the business domain are described as business processes.
– Every business process is described using a use case, which acts like a process and can have several sub-use cases and the business processes of a business domain constitute a system map.

- The business process is a set of business level use cases, which are described in a service map.
- Every use case needs at least one service, which fulfils the requirements of the use case.
- All the use cases utilize by themselves or together with other use cases services (Services).
- Services use the information model of the company (Business entities).

The business processes is modelled using activity diagrams concentrating on most important features in a purpose to get an overall picture of the business processes. When the system requirements are more familiar then the use cases and their relationships are modelled on a more precise way using collaboration diagrams.

Different granularity of use cases are designed depending on the point of view of analysis starting with the system level use cases. After that the big picture of communication between participants is defined and the system level use case is divided into smaller business level use cases which encapsulate business processes. This is the critical point and iteration is needed here so that the right size of business level use cases is found and the interfaces between business level use cases can be modelled. Finally, dependencies between use cases are derived so that the reuse of the use cases is possible.

The use case model is depending on the business model and reflects the business vision, which changes in the time. Thus, we have the current business model and the target business model. Because software development is costly and needs a lot of work, we need a balance between today and tomorrow and a path from today to tomorrow. Therefore, the use cases of current and target must be fitted together so that we have a track from the current architecture to the target architecture. This is done using composition either *stringing* or *overlapping*. The stringing composition means that we add target use cases besides the current use cases. The overlapping composition means that we construct target use cases using the today use cases. Business service components are derived so that use cases can be accomplished efficiently. However, use cases are designed again and again in order to get the right granularity of business components.

Information line

Business data has a static nature and business entities exist physically somewhere. Besides, they have locations, views, and targets. This whole is referred as information architecture, which is valuable enterprise assets and is modelled using business data entities. When developing the business software architecture the architect must design what information is needed in the future, where it is and who is using this data. SOSE provides as its contribution a path from the existing information architecture to the future architecture.

The *Enterprise Object Model (EOM)* and the business process model are the fundamental building blocks of business applications (Herzum et al., 2000). We refer to this model as business information model (BIM). Because the business data has a long lasting nature, it is described as early as possible. BIM has an influence to the whole system and therefore SOSE defines the business component system model after main business processes and the BIM model are created. Thus, SOSE sees services as an answer for the needed processes.

Final design

The presumptions for final design are:

- Every service is seen as a *business service component (BSC)*, which is a design component used in platform independent models (PIM).
- The business model is built using these components so that integrity of data remains and a needless replication is avoided.
- The implementation model is built using design components (Component Implementation PSM).

The SOSE component model has three levels of granularities: *system*, *business service* and *component level*. Both the SOSE model and the Herzum component model (Herzum et al., 2000) have three levels of the granularity, but the SOSE model is service based whereas the Herzum model is component-based (see Figure 2). The SOSE system level component is a set of services, which all publish services to the customers using a business service component which has only one network addressable service component. Thus, other components are used only inside the BSC. As a result, the system is more manageable because distributed objects are not used.

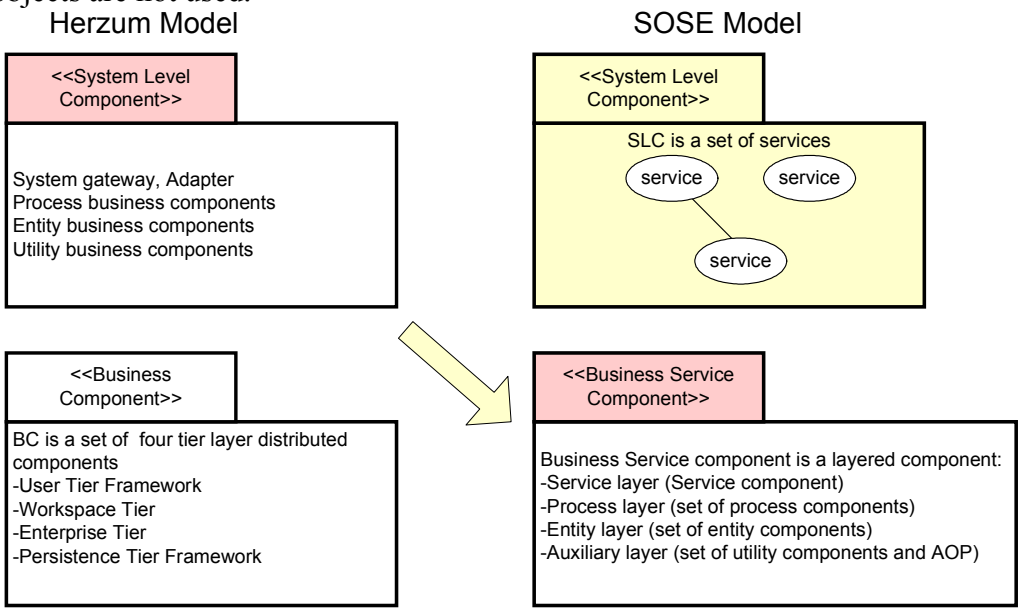

Figure 2. Component models of Herzum and SOSE.

The business service component has 4-layer architecture (Buschmann el al., 1996) where each tier is separated of each other and the communication occur using well-defined interfaces. Components construct hierarchical layers, where components depend on another component forming components dependency graph (see figure 3).

The upper most is the service layer with one component referred as *service component (SC)*, which is a stateless session component. The service component's concern is to conduct, communicate, and coordinate other components in order to create the service using a workflow of a set of process components. The service component offers service through an external interface to other participants in a bus using other business service components, utility components, legacy systems through adapters, and distributed systems.

Second layer is the process layer consisting of *process components (PC),* which fulfil the actual service workflow. The next layer downwards is the entity layer with business entities and other resources like legacy systems' adapters referred as *entity components (EC),* which are conducted by the service component.

The lowest level is the utility layer with supporting components. For instance user authentication, distributed transaction management components, and distributed service agent component locate here. The service agent component helps SC and parses the process service request instructions which are usually described using for instance a *Business Process Execution Language (BPEL)* (Alonso et al. 2004). Of course the component mix depends on the component model and the platform in use. For instance the implementations of the design component like transaction management, service agent, and authentication can vary. When Lightweigth Container -model in Java (Johnson 2004) is used the transaction management, service agent and authentication are implemented using Aspect-oriented programming model (AOP).

The BSC uses coordinated interaction mode between layers (Herzum et al., 2000). This coordination style is recommended, if persistence resources are provided by other service or resource outside the business component. It is possible that there is a persistence data provider, which offers information as a service used in many processes. In this case it is suitable, that there is as few dependencies as possible between components and this service. Thus, coordination gives simplicity in this case.

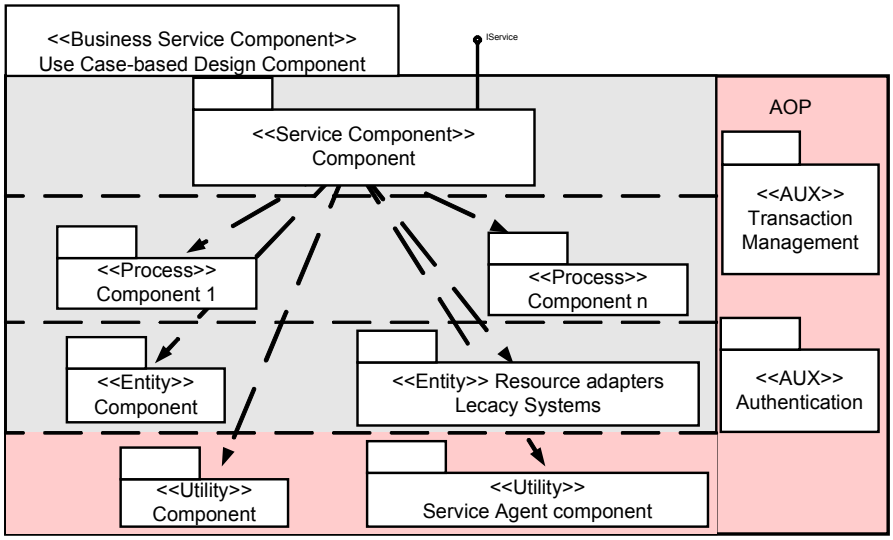

Figure 3. Business Service Design Component.

The SOSE framework includes business case definition, specifying functionality line, information line and final design phase. The functionality line includes business process definition, transition from business domain to software domain using use case maps, mapping use cases to services, and connecting information architecture to services.

The information line includes business concept transformation to business information (BIM) model offering loose coupling between business concepts and BIM. This feature offers flexibility in software development specially, when legacy systems are used.

Finally, in final design phase services are described as design components referred as business design components. The whole development process is iterative in order to create right size granularity of components. Every business service component with its internal component is ready to be implemented on platform. And at last the implementation phase with module and integration testing constructs and delivers the business process as a service to the customer application.

Experimental results: example scenario

Introduction

In this section we present experimental results got in our case example in the Electricity Market domain. With this example we evaluated our framework in practise with SOSE participants acting in electricity domain. We used interviews to collect information and develop the business vision presentation together with our participant.

We consider business processes and their effects on software design. In our example, we specified first main processes and system level use cases. We described the current system and designed the goal, i.e. the target system, according to the business goals and processes referred as functionality line. Secondly we created business information model by applying the information line. Both lines were accomplished intertwined, but here we present the results separately to increase readability. Finally the business service component was designed to fulfil required services and business entities referred as final design phase.

Business Processes in the Electricity Market

Our example case is from power market. It is a description of electricity supplier business activities. The electricity supply business had a "closed market model" with no competition in the past. The company, who owned the electric wires, could operate in a monopoly position. Thus, there was no need to be open and have co-operative systems, but this all changed rapidly. The companies needed co-operation with other electricity operators, but software solutions were monolithic. They started to integrate software using one-to-one interfaces. This was costly and the software architecture looks now like spaghetti with many dependencies between business software programs and systems.

In this example the main focus is on strategic goals. Questions to be answered:

- What services are needed for five years?
- What is the business evolution?
- Is the business data valid?
- How the existing software solutions map to new architecture?

Because of the closed market model information systems are not co-operative. Business process planning using an existing information application context is very inflexible, expensive and has a lot of point-to-point integrations between applications.

Functionality line

When applying SOSE framework the functionality line was first followed. In our case study we specified main processes. The electricity business main process consists of following sub-processes (see Figure 4):

- Marketing including CRM, Sales, Invoicing, and Metering,
- Network Monitoring Process,
- Network Service and Maintenance Process,
- Network Planning Process,
- Electricity Marketplace Brokering Process.

The sub-processes have crossover dependencies such as security, authentication, authorization, transaction management, and reliable messaging. These proc-

esses are defined as aspects and thus not described in use case diagrams (Jacobson, 2003).

The marketing is responsible for customers' marketing activities (custom-sales-chain) like offer, order, order confirmation, invoicing. The Network Monitoring process monitors the state of local electricity network and balances with other networks. Furthermore, it offers information about electricity consumption (kWh) in the network, what is the state of the global electricity network, where and what kind of alarms exist in the network. The Network Service and Maintenance Process uses the information produced by the network monitoring process and offers the failure service, which consists of the network guidance and repair work. In the network planning process the network structure is designed, evaluated and stored in a digitalized form. This process offers e.g. graphical presentations of the network and makes calculations of the network load. The Electricity Marketplace Brokering Process buys and sells the electricity in the Nordic electricity exchange.

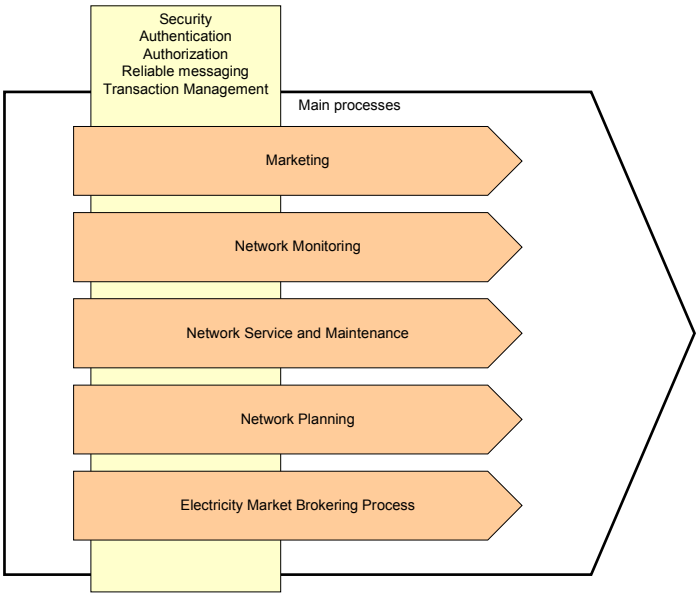

Figure 4. Main processes.

The next phase of the functionality line is to define a process map. Here the use case diagram of UML was utilized. All the main processes are intertwined and have dependencies with each other. The first presumption was that the processes are supported by system level use cases. We described these system level use cases in a process map. The system map definition tries to answer questions: what kind of dependencies these processes have with each other and which roles do the interest groups have. Further, we should know how the customers utilize the system level component at the high abstraction level. The dependencies between use cases were modelled using uses- or extends-stereotypes. Before the suitable granularity of use cases were mapped, the processes were redefined using sub-

processes. This stage is important, because the goal is to model the system as system level business components using the system map (see Figure 5).

The process map describes the business main processes and their external dependencies and stakeholders. The Marketing use case offers services to the customers and customer services and extends the services of the Network Planning, e.g. adding new customer information, listing and updating the existing customer information. The service and maintenance uses the services offered by Network planning and Monitoring, which gives information of net structure, state of the network, and situation information from the geographic information system (GIS). The Exchange uses Monitoring to make business transactions, e.g. what is the consumption of electricity during different time of the day, what is the shortfall or overspill. The balance information about the energy between suppliers is also important.

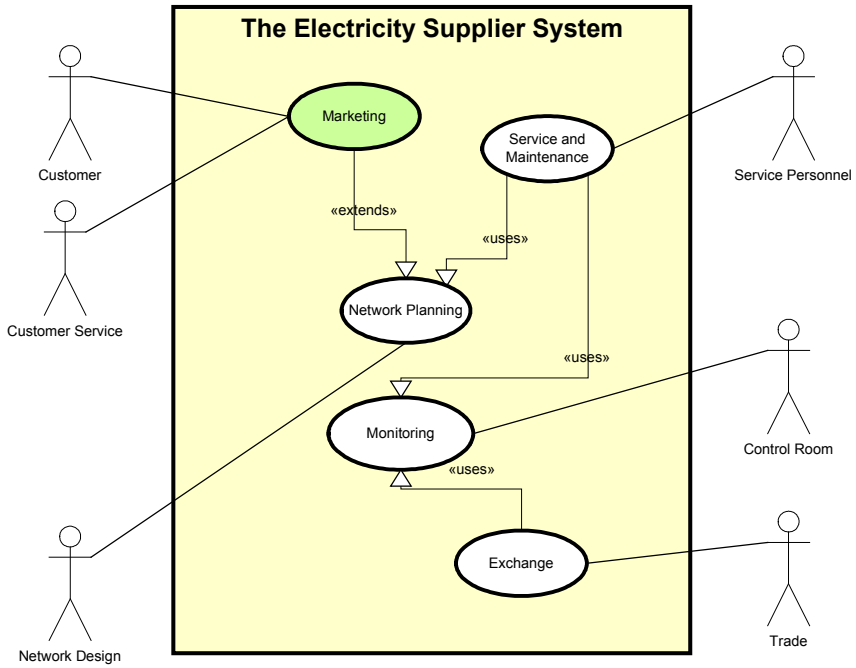

Figure 5. Process map: Main processes as use cases.

In the next phase in the functionality line the system level was divided into business service level. The business context was emphasized here. Use cases were described in a service map transformed as services. Both existing business use cases and the future use cases were defined and finally after service transformation services were mapped to business service components, which have their own islands of data which can have dependencies with other business components. As an example the Marketing service map is presented (see Figure 6). The upper diagram describes the existing situation and the lower diagram describes the strategic target and the services needed for customer service. Both diagrams have the basic elements but the target one has some extra items. The existing system offers tools

for Marketing like contract handling of customers and basic invoicing services but the target system has a database of all potential customers in addition to the current customer and network data.

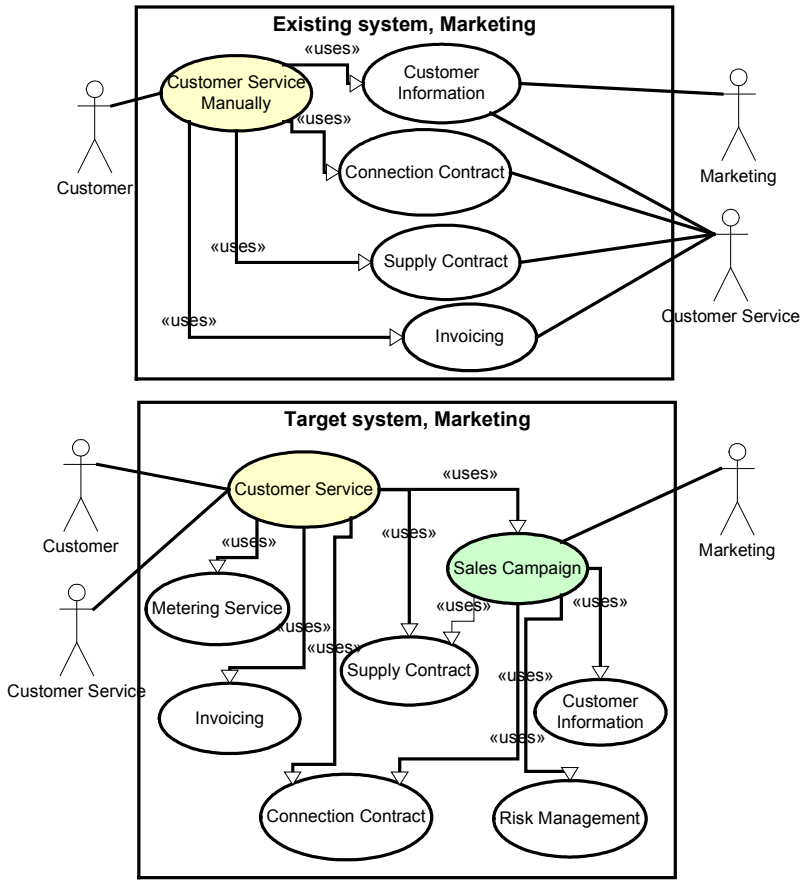

Figure 6. Marketing service map of the existing and future system.

Next the services and their behaviour were studied further by defining contracts between services using collaboration diagrams. The connection contract use cases were mapped to a business service component. The behaviour of this component was described using a sequence diagram (see Figure 7). The main assignment was to define processes, parameters, concepts, and choreography. First, a new connection was created and basic information was collected. The output of this operation is the basic information: customer number, name, address, and other invoicing information. Secondly, the building service with the design information for the connection building was delivered. The output of this service is an offer for connection building and detailed technical information. Thirdly, a contract was made. After this last phase the actual building phase of connection can be started.

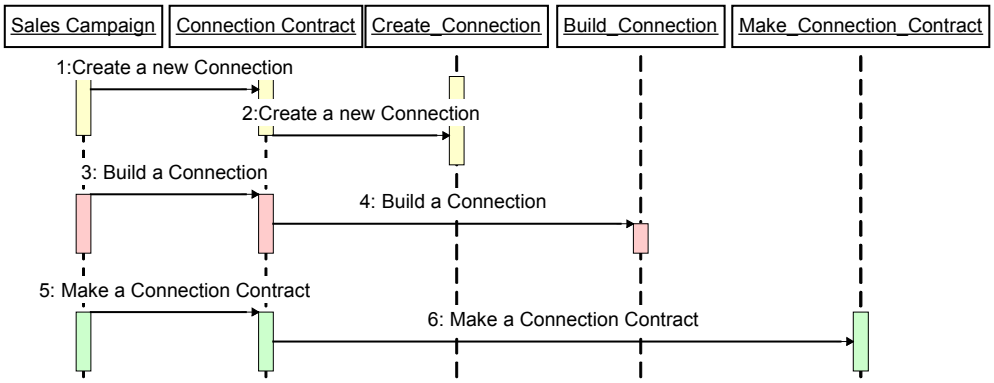

Figure 7. Sequence diagram.

Information line

The business information model (BIM) of marketing was modelled using the information line of the SOSE framework. BIM consists of customer data, the facility data, the network planning data, metering data of the electric network and risk management data. The marketing system component uses other system components in order to get metering data and network planning data (see figure 8).

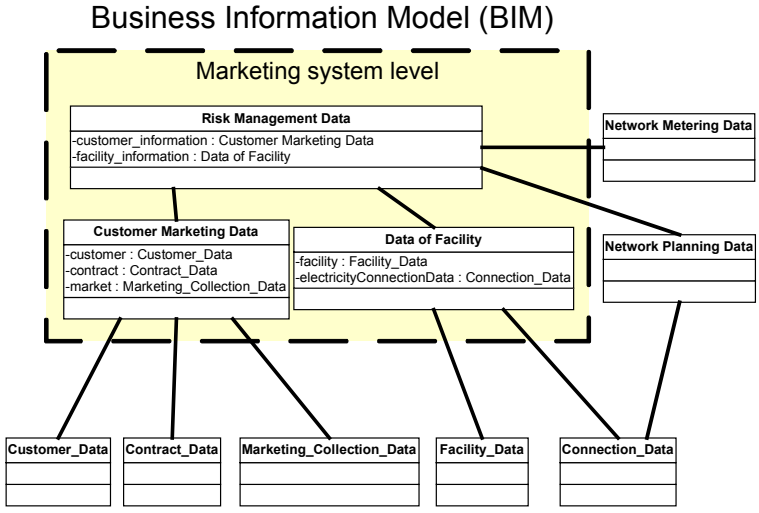

Figure 8. Business information model (BIM) of Marketing

Final design

The last phase in defining the design business service component (BSC) consists of the analysis and design phase, which maps the requirements of the business

services to business components. The offered interfaces of the BCS are defined including contract information (pre/post conditions), which define prising, user rights, and the quality requirements of service. BSC offers through endpoints (for entry and exit) interfaces for external communication (Chappell, 2004). The development process is iterative and cycled. This means that at any stage it is worth to study, how decisions influence on other parts of the system.

As an example, the connection contract business service component is presented in figure 9. The cooperation style here is coordinated cooperation, where the service component acts as a coordinator. However, it is possible that each process component has its own data islands. The persistence of data is guarantied by entity components, which use appropriate resources like databases, other services, application interfaces, and legacy systems.

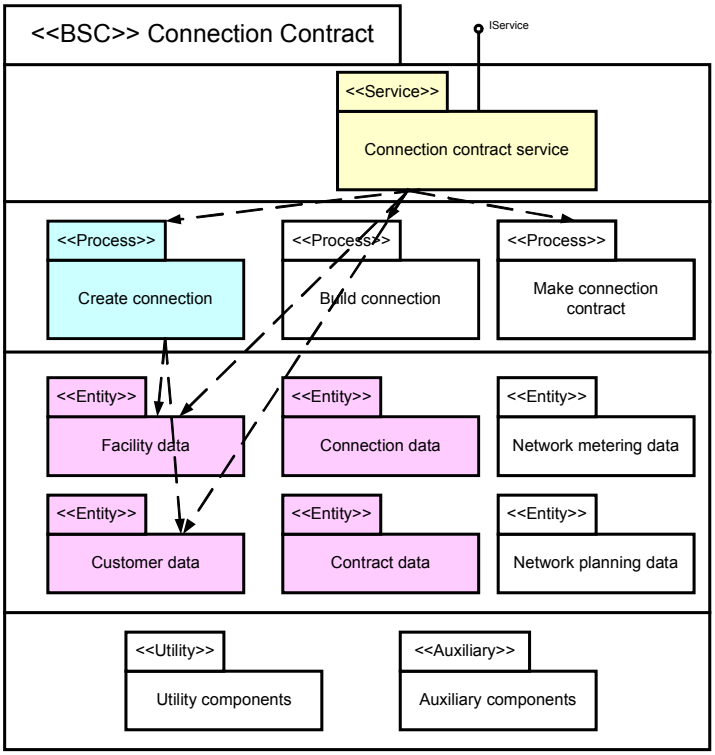

Figure 9. Connection Contract Business Service Component (BSC).

In conclusion the business domain was specified and the platform independent models were built. The SOSE component model conforms to business model as precise as possible and considers from several points of views the problem domain using several description models: use case model, collaboration models, static and dynamic models. From design components (PIM) are implementation models (PSM) transformed. Thus, design components must meet the technical

issues so that the granularity of distributed components is taken account to: components have right size and the system structure has no extra complexity. Therefore the simplicity is an objective, but not in cost of business goals.

Discussion

SOSE framework uses state-of-art technology. Its main target usages are enterprise integration and software integration inside the enterprise when the interoperability between software is increased. With the SOA as a backbone it offers uniform platform for business development. As a mediating tool it uses use case notation as process description language. The crosscutting dependencies in use cases are specified using aspects which often have security tasks. Also extension stereotype dependencies in use case diagrams are aspects defining extensions and extension points where extension is used in main use case.

The SOA technology adds a new layer on top the software architecture adding complexity. At the same time it makes integration easier, adds reusability, and is a common language between software vendors and customers. Drawbacks are performance, complexity, and security issues which must be taken carefully account on. Besides the business language and software language differ quite much of each other, thus graphical description tools are needed.

Using the MDA-models the early design late binding goal is achieved where the platform independent model is transformed to platform specific model. The later this transformation is made the better portability is achieved. The SOSE platform independent model is a collection of design components, which are building blocks when the platform specific models are created. Thus, SOSE is both an abstract design framework and an implementation tool when arranging services as code packages. Together with the loose-coupled component system of SOSE framework the business and system developers can accomplish very cost and quality competitive, reusable, adaptive, and portable system development.

The SOSE Framework helps enterprises to combine business vision, processes and existing artefacts. Thus, it is a mediator between the business and information system developers. It is informal enough with graphical descriptions for discussions and new ideas. It separates the actual, existing information system implementation from the business vision offering tools to design a path to fulfil business visions.

Conclusion and future work

SOSE framework has a clear structure a path from business case to the implementation. It offers step by step instructions how business and software are combined in a way that business requirements are achieved. The simplicity of the model

adds its usability among non-professional members of the development team. The flexibility of the model makes it more efficient because the developer can pick needed parts of the model and apply them in the software development.

SOSE offers as its contribution a solid path from the current software system of the enterprise to the target system through a mediating system model. As a framework it offers reusability. We are going to make a manual for this method and its architectural features, apply it into practice with several business companies and test it more completely.

We are going to validate and further develop the method in practice during years 2005-2006 by building an open source implementation and studying how our principles meet the needs of the practical implementations in software industry. This requires that we need to consider how use cases are formed, which kind of dependencies are between use cases, how use cases are combined or divided, and how business service components from business components are grouped.

Acknowledgements

This paper is based on research in the SOSE project (2004-2006), funded by TEKES (the National Technology Agency), European Regional Development Fund (ERDF), ICT and customer companies in electricity domain. I wish to thank professor Anne Eerola for her comments, researcher Marko Jäntti for his help in data collection and companies for participating in interviews and discussions.

References

Alonso, G., Casati, F., Kuno, H., Machiraju, V.(2004). *Web Services Concepts, Architectures and Applications*. Springer.

Bond, A. (2001). ODSI: Enterprise Service Co-ordination. *DOA'01*.

Bosch J. (2000). Design and Use of Software Architectures, Adopting and Evolving a Product-Line Approach. Addison-Wesley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.(1996). *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley & Sons Ltd.

Chappell, D. (2004). *Enterprise Service Bus*. O'Reilly.

D'Souza, D., Wills, A. (1998). *Objects, components, and frameworks with UML : the catalysis approach*. Addison-Wesley.

Herzum, P., Sims, O. (2000). *Busines Component Factory*. OMG Press.

Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1998). *Object-oriented Software Engineering*. Addison-Wesley.

Jacobson, I. (2003). The Case for Aspects. *Software Development*, October 2003.

Johnson, R. (2004). *J2EE Development without EJB*. Wiley.

Järvinen, P. (1999). *On Research Methods*. Opinpaja Oy.

Iivari, J., Lyytinen K. (1998). Research on Information Systems Development in Scandinavia - Unity in Plurality, *Scandinavian Journal of Information Systems*, 10 (1&2):135-186, 1998.

Kructen, P. (1995). The 4+1 View Model of Architecture. *IEEE Software*, November 1995.

Larsen, G.(1999). Design Component-based Frameworks using patterns in the UML. *Communications of ACM*, vol. 42 no.10.

Levi, K., Arsanjani, A. (2002). A Goal-Driven Approach to Enterprise Component Identification and Specification. *Communication of the ACM* , vol 45, no.10, 45-52.

Maciaszek, L., Liong, B. (2005). *Practical Software Engineering A case Study Approach*. Addison-Wesley.

MDA Specification. *http://www.omg.org/mda/specs.htm#MDAGuide*. OMG. (visited 31.10.2004).

Panda, D. (2005). An Introduction to Service-Oriented Architecture from a Java Perspective, *http://www.onjava.com/pub/a/onjava/2005/01/26/soa-intro.html*. (vis. 9.2.2005).

Raistrick, C., Francis, P., Wright, J., Carter, C., Wilkie, I. (2004) *Model Driven Architecture with Executable UML.* Cambridge University Press.

Pawson, R., Matthew R. (2002) *Naked Objects.* Wiley.

Ruh, W., Maginnis, F., Brown, W.(2001) *Enterprise Application Integration*. Wiley.

Woods, D. (2003) *Enterprise Service Architecture*. O' Reilly.