

**Documenting and Managing
Service Oriented Software Architectures**

Marko Väyrynen

Thesis

Master of Business Administration

Business Information Technology

2011



<p>Tekijä tai tekijät Marko Väyrynen</p>	<p>Ryhmä YTI09K</p>
<p>Raportin nimi Documenting and Managing Service Oriented Software Architectures</p>	<p>Sivu- ja liitesivumäärä 95 + 135</p>
<p>Opettajat tai ohjaajat Lili Aunimo (24.5.2010 asti) Jouni Soitinaho (24.5.2010 jälkeen)</p>	
<p>Arek Oy rakentaa ja ylläpitää SaaS liiketoimintamalliin pohjaten tietojärjestelmäpalveluita Suomen työeläkesektorille. Arekin tietojärjestelmien taustalla oleva tekninen ympäristö on hyvin heterogeeninen koostuen monen eri aikakauden teknologioista. Käytännössä palveluiden tuottamiseen käytetty järjestelmäkanta koostuu sekä moderneista J2EE pohjaisista järjestelmistä että perinteisistä keskuskonejärjestelmistä. Vanhimmat järjestelmistä ovat peräisin 1980-luvulta ja uusimmat 2000-luvulta. Kokonaisuutena näiden tietojärjestelmien arkkitehtuuri oli dokumentoitu hyvin vaihtelevasti eri kuvaustavoilla sekä abstraktiotasoin.</p> <p>Vuonna 2010 Arek hyväksyi projektin jonka tehtävänä oli pohtia kuinka arkkitehtuuridokumentaatiota voitaisiin parantaa ja yhdenmukaistaa. Tämän lisäksi projektin tehtävänä oli kehittää arkkitehtuuridokumentaation ylläpitoprosesseja. Tässä opinnäytetyöraportissa kuvataan tuon projektin kulkua ja lopputuloksia.</p> <p>Opinnäytetyössä käytettiin toimintatutkimusta tutkimusstrategiana ja raportin teossa hyödynnettiin kertovan muutosselonteon menetelmää. Työn teoreettinen tausta muodostui liiketoimintaprosessien hallinnan, arkkitehtuurin hallintamallin, SaaS liiketoimintamallin, palvelukeskeisen arkkitehtuurin, yritysarkkitehtuurin ja järjestelmäarkkitehtuurin teorioista. Eri sidosryhmät olivat mukana työn eri vaiheissa työpajoissa sekä haastateltavina. Puolistrukturoituja haastatteluja tehtiin opinnäytetyön aikana 25 kappaletta.</p> <p>Projektin lopputuloksena Arekille syntyi uusi käytäntö arkkitehtuurikuvausten tekemiseksi ja ylläpitämiseksi. Vuoden 2011 aikana Arek käynnisti jalkautusprojektin jonka tehtävänä oli ottaa käyttöön tässä työssä ehdotetut kuvaustavat ja ylläpitoprosessit koko organisaatiossa.</p>	
<p>Asiasanat Järjestelmäarkkitehtuuri, Kokonaisarkkitehtuuri, Prosessit</p>	

Business Information Technology

<p>Authors Marko Väyrynen</p>	<p>Group YTI09K</p>
<p>The title of thesis Documenting and Managing Service Oriented Software Architectures</p>	<p>Number of pages and appendices 95 + 135</p>
<p>Supervisor(s) Lili Aunimo (until 24.5.2010) Jouni Soitinaho (after 24.5.2010)</p> <p>Arek Ltd. provides IT services to pension insurance sector of Finland. The earnings logic of Arek is based on SaaS business model. The technological environment in Arek is heterogeneous and technology landscape consists from both modern J2EE and legacy mainframe systems. Oldest of the systems were built in 1980's and the newest ones are built couple of years ago. During the years, the architecture of these systems was documented in great variety, with different notations and abstraction levels.</p> <p>In 2010 Arek approved a project to improve architecture documentation practices and maintenance processes. This document is a thesis report formed from that project.</p> <p>The research strategy used in this study was based on action research and this report was formed by using a narrative change accounting method. The theoretic background of this study consisted from theories of business process management, architecture governance, software as a service business model, service oriented architecture, enterprise architecture and software architecture. Different stakeholder groups took part in project by attending workshops and interviews. A total of 25 semi structured interviews were made during this thesis.</p> <p>As a result this project created a new praxis to document and maintain systems architecture in Arek. At 2011, Arek decided to start a project to rollout the new practices throughout the enterprise.</p>	
<p>Key words System Architecture, Enterprise Architecture, Processes</p>	

Table of contents

1	Introduction.....	1
1.1	Research background.....	1
1.2	Research question and goals.....	2
1.3	Research methodologies.....	3
1.4	Thesis project and report structure	4
2	Methodology.....	5
2.1	Action Research.....	5
2.1.1	Action Research and the role of the researcher.....	7
2.1.2	Validity and reliability in action research	8
2.2	The method of narrative change accounting	9
3	Theoretic Background.....	11
3.1	Business process management and processes.....	11
3.1.1	Process improvement and re-engineering	13
3.1.2	Process Modelling	14
3.2	Architecture Governance.....	17
3.3	Software as a Service Business Model.....	21
3.3.1	SaaS at Arek Ltd.	23
3.4	Service Oriented Architecture	24
3.5	Enterprise Architecture	25
3.6	Software Architecture Documentation.....	27
3.7	Available viewpoint frameworks.....	33
3.7.1	IEEE 1471-2000 / ISO/IEC 42010 Standard	33
3.7.2	RM-ODP	35
3.7.3	DoDAF.....	37
3.7.4	MDA	38
3.7.5	Rational Unified Process/Kruchten 4+1.....	39
3.7.6	TOGAF	40
3.7.7	Views and Beyond approach	42
3.7.8	Zachman framework.....	43

3.7.9	Archimate	44
3.8	Example, selected viewpoints in three telecom industry companies.....	45
3.9	Notations used in Architecture Descriptions	46
3.9.1	IDEF	47
3.9.2	UML.....	49
3.9.3	Archimate	51
3.10	Synthesis of the theories.....	59
4	Documenting and managing software architecture in practice.....	60
4.1	Case Arek Ltd.	60
4.2	Current state.....	60
4.2.1	Software Architecture Documentation	60
4.2.2	Software Architecture Documentation maintenance process	64
4.3	Future state.....	65
4.3.1	Software Architecture Documentation	65
4.3.2	Software Architecture Description maintenance process	75
4.4	Validation of future state	80
4.4.1	AD validation.....	80
4.4.2	Process validation.....	81
5	Results discussion	82
5.1	Contribution to the community and further research	83
	Bibliography	85
	Attachments	90

1 Introduction

1.1 Research background

A documented and up-to-date software architecture creates a good foundation for an organization to build its business. Arek Ltd. is an IT-service company in Finland, providing application services for Finnish pension insurance sector. Arek bases its business on Software as a Service business model, meaning that customers pay only for the application services they actually use.

History of Arek is relatively short. Arek was founded in 2003 to build a centralized earnings register to Finland. On that time all architecture efforts were concentrated in building a single system. In January 2008 Finnish Centre for Pensions outsourced several information systems to Arek Ltd. This event resulted in a situation where the systems at Arek had their architecture documented in great variety, with different notations and abstraction levels. Some of the systems didn't even have anything that might resemble an architecture description, as the oldest of the systems were built in 1980's and the newest ones only couple of years ago.

In 2010 Arek approved a project to improve architecture documentation practices and maintenance processes. It was also known that the project results would also benefit the existing Enterprise Architecture effort at Arek.

The writer of this thesis worked at Arek during this project as a Technology Manager, responsible for Architecture work in the role of Chief Architect.

1.2 Research question and goals

Eoin Woods (SEI 2011) said that “Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.” In Arek the importance of architecture is more than just the success of software projects, it is the core of business. The application services, some of them created as products, are provided to customers. Company income is based on these Service Oriented Architecture (SOA) style services and their on-demand usage. Under these services lies the realization of software architectures.

But what is software architecture? There are numerous definitions for software architecture and none of them actually gives an exact answer to the question. What is included in software architecture, hardware, software, processes, interfaces? Neither there is an exact answer on how software architecture should be documented and how the changes to software architecture are managed in a way that the documentation is still up-to-date. It is also said that the business and technical environment also gives requirements for architecture. For example some of these influencing factors are Software as a Service business model and architectural styles such as SOA.

From this mindset rises the research question of my thesis: how to document and maintain service oriented systems architecture in a form that stakeholders can understand it? This question can further be split into following questions:

- How to document service oriented systems architecture?
- How to maintain service oriented systems architecture?
- What kind of architecture description should be made in order for it to be understandable to the stakeholders?

A research hypothesis is that the architecture documentation practice is better after this project than it was before it. The goal is to define the future state architecture documentation model and maintenance processes of that documentation for Arek Ltd, a company that heavily utilizes the Software as a Service business model and SOA in its business.

1.3 Research methodologies

The research methodology or actually a research strategy used in this study was based on action research.

During the project several research methods were also used. Current architecture documentation was analyzed by reading the documentation through and making remarks on what information is missing in documentation. Opinions were collected about the existing documentation from different stakeholders by using personal semi-structured interviews and workshops. By using these methods it was ensured that all stakeholders were able to give their personal input to the project outcome.

The researcher worked actively with project issues during normal day-to-day work. During this time the researcher made small interventions in different projects at Arek. The result was that the ways of documenting and maintaining architecture were shaped towards the presented future state also during the project.

A critical success factor was that the to-be praxis was formed in a practical and realistic way. For this reason, the involvement of different stakeholders and the careful result validation were important.

In the end of the project the joint architecture group of the Finnish earnings related pension sector showed interest to the findings of this project, and gave their input to the end results from the perspective of extended enterprise architecture.

After the project ended, the managers at Arek decided to start a new project. The goal of that project was to implement the processes and documentation practices represented shortly in this document and more thoroughly in the attachments.

1.4 Thesis project and report structure

This thesis formed a project that improved the architecture work at Arek Ltd. All actions in this project were based on applied theories and the goal was to find an answer to research question. The project consisted of two phases: current state analysis and future state definition. The implementation of future state was not included in the project.

The action research method is explained more in more detail in chapter 2. The theoretic part of this project was based on theories of architecture frameworks, software as a service business model, process management, Service Oriented Architecture, enterprise architecture and architecture notations. These theories are described in chapter 3. The empiric part of this report is in chapter 4, explaining the current state of architecture documentation in the 2010 and how the future state was formed during the project. Chapter 5 includes a discussion about the project results and chapter 6 gives ideas on what might be the future subjects of research.

2 Methodology

This chapter represents the key methods used in this project, the action research as a research methodology and narrative accounting as a way of formulating this thesis report.

2.1 Action Research

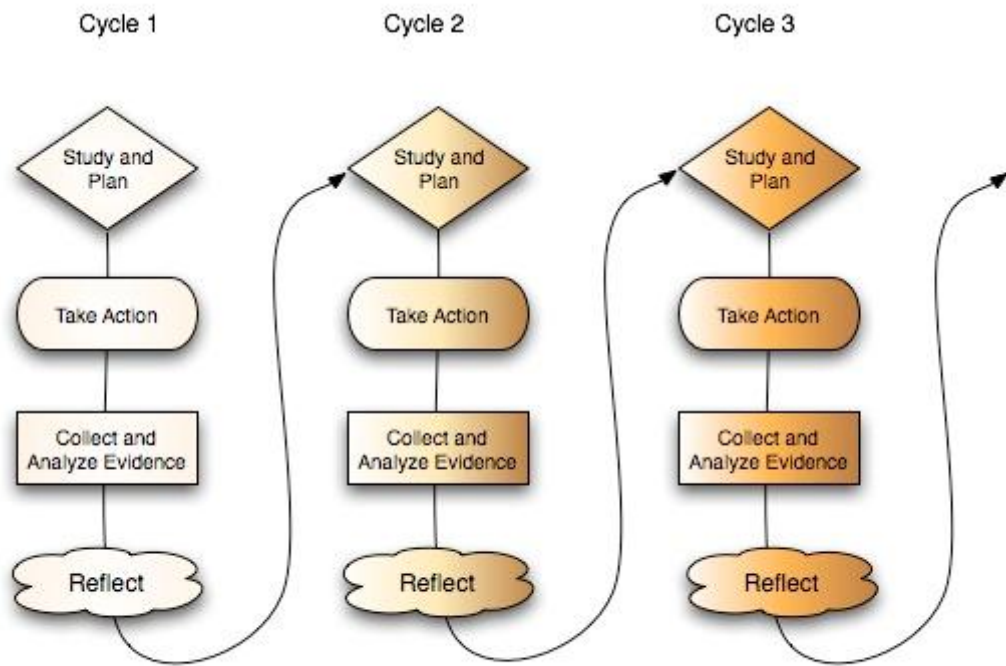
There is no exact definition for term action research. Action research cannot be differentiated by the research techniques used in it, because these techniques change from research to re-research. The action research is more like a research strategy belonging to qualitative methods than an actual method. (Kuula 1999, 218.) Metsämuuronen (2007, 222) states that action research is a situational, collaborative, participatory and self-evaluative process.

The concept of action research was first introduced at the beginning of 1950 by a social psychologist, Kurt Lewin, in his publication *Field theory in social science*. Lewin states that a social behavior tends to form routines and practices. This might help the work in an organization, but not all practices and routines work for the best of organization. If the surrounding conditions change, so must the routines and practices. (Heikkinen et al. 2007, 28.)

In action research the researcher works together with community in order to solve some existing problem (Kuusela 2005, 34 – 35). Action research does not limit the methods available for the research. The researcher is an active contributor in the research and the change process is driven together with the researcher and other people involved in the research project.

(Kananen 2009, 9 - 23.) The action research changes the reality by researching it and researches the reality by changing it. The new information is created in order to develop the praxis (Heikkinen et al. 2007, 15 - 16).

Riel M. (2010) has described action research as a progressive cycle of problem solving. The cycles go through phases of study and plan, take action, collect and analyze evidence, and reflect.



Progressive Problem Solving with Action Research

Picture 1: Progressive Problem Solving with Action Research (Riel M. 2010.)

In the Study and Plan phase a researcher defines what he or she wants to research and what are the main goals of research. The used methods and actions are defined and work is divided to project members. In the Take Action phase the defined actions are realized. Collect and Analyze Evidence phase is the most distinguishing phase when comparing action research to normal way of working in organizations. In this phase the researcher collects information for later reflection and analysis. For example the researcher might collect diary notes, run interviews or validate the achieved results. Even though the collect and analyze evidence is a phase of its own, it is also said that this action should take place in all phases of action research. The last phase is reflection, although the actions should also be continuously reflected within the research cycle. In the Reflection phase the researcher reflects how the research cycle succeeded. And based on this information a new research cycle is planned. (Suojanen 1992, 56 - 62.)

Pekka Kuusela (2005) writes that the action research has been criticized for its ability to fulfill the requirements of scientific research: the relationship between the theory, results and conclusions are often quite vague. On the other hand there are counterarguments that the academic research is often monopolized and it has very weak contact to overall society. Quite often the scientific research is also blamed to be constrained to university community. The goals of academic research and action research are not conflicting, because they both try to

solve real issues. Action research can be mainly a development activity and only partially a scientific research process. (Kuusela 2005, 73 – 74.)

This thesis consists two research iterations. The first iteration concentrated on the current state analysis while the other iteration aimed to form the future state proposition.

2.1.1 Action Research and the role of the researcher

The researchers' role in action research is not to primarily have a deep understanding of how the community or organization under research actually works. The researcher produces understanding on what is the connection between context, actions and results in current problem under research. Action research is a social activity, thus the researcher doesn't define the problem in isolation. The problem definition is done together in the community and the action research doesn't even have to be equally participatory towards all parties through all the process. (Kuusela 2005, 31 – 73.) Nevertheless the researcher is an active part of the community and the challenging part is to define the boundaries of researcher, when he or she is working in the role of a researcher and when he is just a participant in a change driven by the action research (Suojanen 1992, 20).

There are four roles the action researcher can take while he is working in the organization: a guide, consult, participator or organizer. While working in a guiding role, the researcher is just evaluating the results of a change process from a distance. As a consultant the researcher advises organization on how and what kind of changes should be made. When the researcher is actively organizing members of the community to solve their problems themselves, the researcher is working as a participator. As an organizer the researcher organizes democratic forums and assumes that through discussion the community will create new ways to do their chores. (Kuula 1999, 116 – 117.)

In this thesis the researcher worked in all of the four roles, depending always on the current situation at hand. The theory and practice was actually mixed in the process as it was a hard task to try to stay in one particular role. Nevertheless the project result was a success, which was the most important thing in the end.

2.1.2 Validity and reliability in action research

In scientific research validity means how well the research methodology and research subject are consistent with each other. In other words is the chosen method right for this research or not. Reliability means how reproducible the research is. In other words can someone else get the same end results if the research is made again (Heikkinen et al. 2007, 147 - 148).

In action research, both reliability and validity are hard to accomplish. When we look at validity the problem arises from the fact that in action research the researcher is building a social reality so basically it is impossible to reach the reality where the research assumptions can be compared. Regarding reliability the problem is that action research actively changes the reality, so it is very hard to change that reality back to the state where it was. (Heikkinen et al. 2007, 147 - 148.)

To cope with these facts Heikkinen suggests using alternative ways to prove that the research was valid and reliable. The research should follow the principles of historical continuity, reflexivity, dialectics, workability and evocativeness. (Heikkinen et al. 2007, 149 - 156.)

To fulfill these principles the research must indicate (Heikkinen et al. 2007, 149 - 160):

1. The moment in history where the research is conducted, how the research project advanced and what kind of world did it leave behind when it was over.
2. How the researcher had described his or her research material, methods and advancement in research in a way that a reader can validate the thoughts of the researcher.
3. How the researcher has built the resulting truth based on thesis and antithesis in a way that the result is synthesis.
4. How the researcher has described the used method.
5. How the researcher has described the pros and cons of his research in different indicators. In other words is the result actionable in the real world and in example how the ethical aspects are showing up in the research.
6. How the researcher has made his or her study lively and realistic.

In this research a method of narrative change accounting is used in order give the reader some grasp to the reader on how the research went trough.

The researcher has also tried to be objective when he searched previous literature and scientific knowledge. The information is gathered from both national and international sources. It is still possible that some relevant information might have left uncaught in the process. Nevertheless the researcher tried to read the material referenced in this research very carefully and he tried to be as objective as possible.

2.2 The method of narrative change accounting

Narratives enable the understanding of the touch of time. They reflect our position in the world and enable us to distribute experience and illustrate it in an understandable way. (Hyvärinen 2006, 1.)

In this research a method of narrative change accounting is used in order fulfill the requirements of validity and reliability explained in chapter 2.1.

The method of narrative change accounting is a research method used in action research. It is based on a narrative written from the action research. Narrative change accounting method is applied in example in the fields of historical research, storytelling and ethnography (Laitinen 1999, 205). A change narrative is written in six-phases beginning from the opening narrative. The next five phases are description of the research context, writing of the episodic progress narrative, collecting of the accounts, writing of the change narrative and negotiation of the change narrative with the interviewees (Laitinen 1999, 207 - 208).

Opening narrative describes the relationship between the researcher and the organization under research. The next phase is to describe the context of the research, basically where the research took place, what kind of organization was under research. Episodic progress narrative describes, in chronological order, what kind of changes took place in organization. Collecting the accounts phase is used to collect information about the people involved in the research. After this the change narrative is written, which is using the episode that are corrected and fulfilled with the information gathered in the collecting the accounts phase. The last phase is to give the change narrative to the people involved with research and let them comment on how the narrative is reflecting real episodes (Laitinen 1999, 207 – 208).

To cover the principles of Heikkinen et al. the following narrative phases are used:

- Principle 1: The opening narrative and the moment of history where the research took place are described in chapters 1, 4.1 and 4.2.

- Principles 2, 4 and 6: Episodic progress narrative describing the research material, methods and advancement can be found in chapter 4. Theoretic methods are explained in chapter 2.
- Principles 3 and 5: The actual synthesis discussion, based on narratives, is written in chapter 5.

3 Theoretic Background

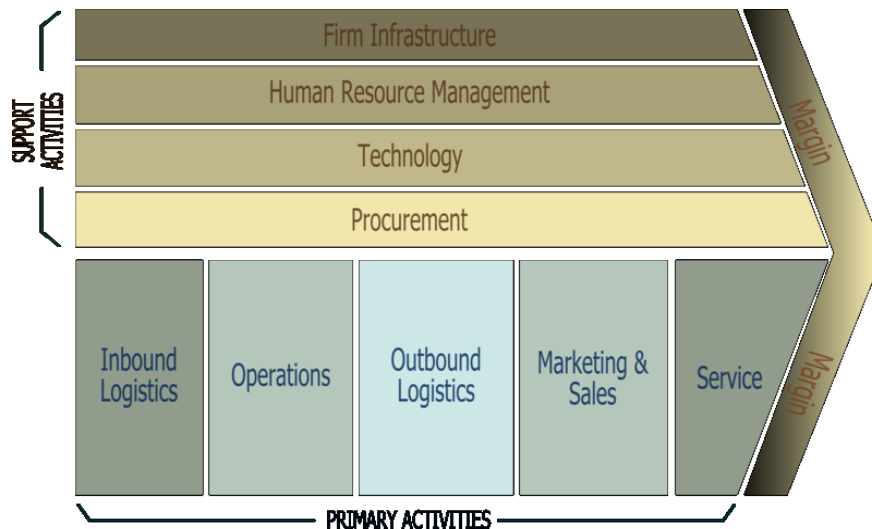
This chapter represents the theoretic background used in this thesis. The main theories included business process management, architecture governance, software as a service business model, service oriented architecture, enterprise architecture and software architecture with different viewpoint frameworks and notations.

3.1 Business process management and processes

When we try to improve organizations processes that deal with the maintenance of architecture descriptions it is important to understand what is business process management.

Business processes management (BPM) has existed in the literature for quite a long time. In 1911 Frederick Winslow Taylor published a book called *Principles of Scientific Management* that introduced new ideas on how to improve and manage business processes. (Harmon 2007, 2.) Today, BPMN, a widely used standard to describe processes, refers BPM as services and tools that support process management (OMG 2011, 1). Nevertheless the need to manage and improve processes is an ancient practice. Archeologists have studied how potters used to create their wares. Archeologists discovered that that the potters refined their pot-making process in order to create better products, faster and cheaper. (Harmon 2007, 1.)

After Taylor, the next major improvement in the world of BPM took place in the 1985 when Michael Porter wrote a book called *Competitive Advantage: Creating and Sustaining Superior Performance* and its predecessor *Competitive Strategy* (1980). Porter introduced value chain, a concept that is still widely used. In this model a company is divided to five primary activities and to a four supporting ones. By using this model a company can discover what products are costing and what is going to be the margin when company sells a product. (Harmon 2007, 3-4.)



Picture 2: Michael Porter's generic value chain (Wikipedia 2012d).

When you compare Porter's model with the concept of core business process and supporting business process one can clearly see that primary and supporting activities are actually processes. According to Harmon, Porter defined the concepts of core process and supporting process. Core processes are something that generate products or services while supporting processes do not add any value (something that a customer is willing to pay for), but assure that the core processes continue to function properly. (Harmon 2007, 41 - 86.)

But what is a business process? BPMN standard defines business process as "set of business activities that represent the steps required to achieve a business objective. It includes the flow and use of information and resources." (OMG 2011, 499.) Process also has a start with input and ending with output. It always has a customer and an owner or manager within an organization. (Harmon 2007, 114 -115, 198.)

In this thesis the processes around architecture descriptions are clearly support processes for the Arek. The core processes concentrating on service development and delivery. But how should these support processes be modeled and improved? The next subchapters deal with these questions.

3.1.1 Process improvement and re-engineering

Today, a lot of companies have interest on something called business process reengineering (BPR). This movement began after few interesting papers were published in the early 1990's. Michael Hammer published a paper called "Reengineering Work: Don't automate, Obliterate" and Thomas Davenport with James Short published their paper in Sloan Management Review: "The New Industrial Engineering: Information Technology and Business Process Design." (Harmon 2007, 9.)

The idea of BPR was that companies should think on improving their value chains. In short the idea of BPR was to redesign processes in order to utilize information technology to gain competitive advantage. Some companies succeeded while others failed in their BPR projects. (Harmon 2007, 8 - 11.) As BPR projects are hard it's usually a very good idea to use some method in process improvement.

One of people oriented process development methods is Six Sigma, founded in the early 1980's by W. Edwards Deming and Joseph M. Juran. Six Sigma was seen as quality oriented method and toolset that emphasized company culture, dedicated into employee training, to support process change in the organization (Harmon 2007, 8 - 11).

Harmon (2007, 353 – 382) presents a BPTrends methodology, aimed to help in process redesign. The method basically goes through the cycle of starting a project (understand), defining as-is processes (analyze), defining to-be processes (redesign), implementing processes and rolling those processes to organization. And then again, look if the processes need any improvement.

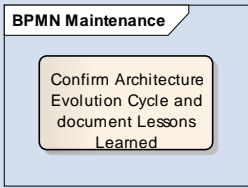
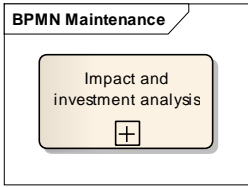
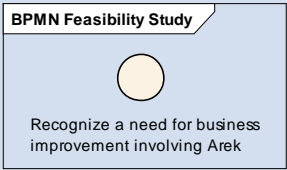
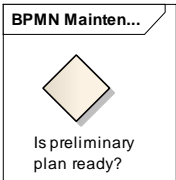
BPTrends is a very similar way of thinking as in action research and in TOGAF ADM cycle (described in chapter 0). With this similar method processes supporting Architecture governance were designed for Arek. The as-is processes were documented first and then the to-be processes were designed. The processes were modeled by using the BPMN standard.

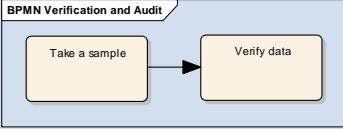
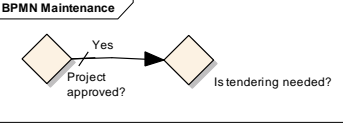
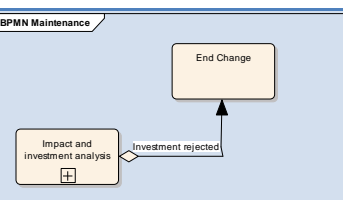
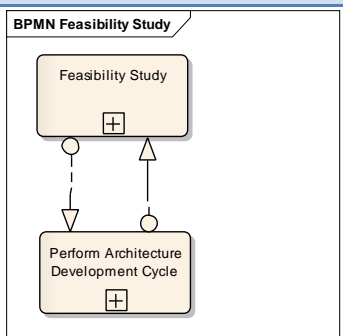
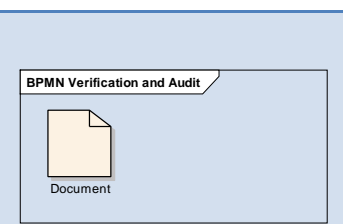
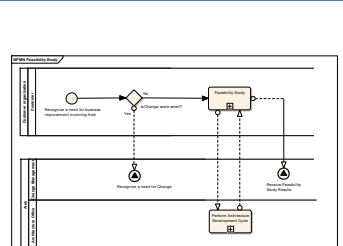
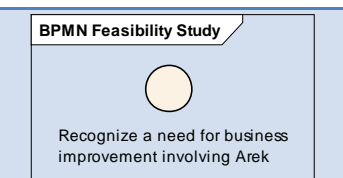
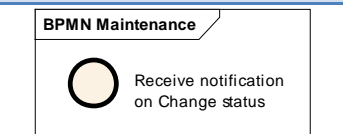
3.1.2 Process Modelling

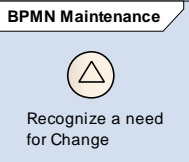
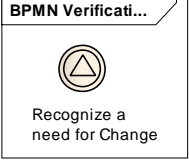
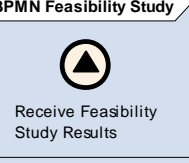
When we talk about process modeling, a consensus on how to do it seems to have been reached. The alignment of process modeling notations begun on 1990 when Geary Rummler and Alan Brache wrote their book, *Improving Performance*. The notation shown in the book is usually called as Rummler-Brache notation. Those concepts were then used in Object Management Groups’s (OMG) UML notation in the form of Activity Diagrams. The Business Process Management Initiative (BPMI) created BPMN notation in 2004 and in 2005 BPMI was merged into OMG. BPMN has number of different symbols for describing what happens in process, but there is also a basic set which is enough in most cases. (Harmon 2007, 232).

In this thesis I decided to use BPMN as it is easier to approach from business point of view than UML; although the basic set of symbol is exactly the same. UML is generally considered somewhat “technical” than BPMN. Table 2 summarizes the core symbols of BPMN.

Table 1: Core BPMN symbols (OMG 2011, 29 - 41).

Symbol (with case example)	Explanation
	<p>An activity is a generic concept for work that a company performs. Activities inside a process can be sub-processes or tasks.</p>
	<p>An activity with more detail is an activity symbol with an + inside it. This symbol means that there is another diagram that explains the activity in a more detail.</p>
	<p>An event is something that happens during the course of a business process. Events usually have a cause (trigger) or an impact (result). There are three types of events based on when they happen: Start, Intermediate and End.</p>
	<p>A gateway is used to control the divergence or convergence of a sequence flows in a process. It will indicate branching, forking, merging and joining of paths.</p>

	<p>A sequence flow shows the order in which activities are performed in a process. Labels can be associated with arrows to give more information on what is happening in the process.</p>
	<p>Default sequence flow shows the default behavior in a process flow. Default flow is shown with a crossing line in the normal sequence arrow.</p>
	<p>Conditional sequence flow shows a condition that the process must meet in order to advance to the direction of the conditional sequence flow. A conditional sequence is shown with a small gateway in the beginning of an arrow.</p>
	<p>A message flow indicates that information is transmitted between two organizations or individuals (usually shown in different pools with swim lanes).</p>
	<p>A data object is an artifact that is moved from one step of the process to another step of the process.</p>
	<p>A pool with swim lanes provides a context for activities. Organization departments, roles or other participants are described in small boxes and activities and flows are drawn inside bigger boxes.</p>
	<p>Start event starts a process.</p>
	<p>End event ends a process.</p>

	<p>Signal start event indicates that start of process is triggered by some pre-defined event.</p>
	<p>Signal intermediate event indicates that the process continues in some other process that is started with an event.</p>
	<p>Signal end event indicates that a process ends in making of a predefined event.</p>

Laamanen (2003) states that a modeled process description should include the critical steps, show dependencies and to help everyone to understand the big picture and the roles of process participants. It is also important to support the co-operation of process participants and to provide flexibility to the daily operations. The documentation of a process should be short (maximum of four to five pages long document), have same the same notation as other processes. Naturally the process should be understandable (over 15 to 20 tasks or sub-processes makes a process description hard to understand) and to have a name, unique identification, name of the modeler and creation date. (Laamanen 2003, 59 -81.)

The processes modeled during the thesis project are described in chapter 0.

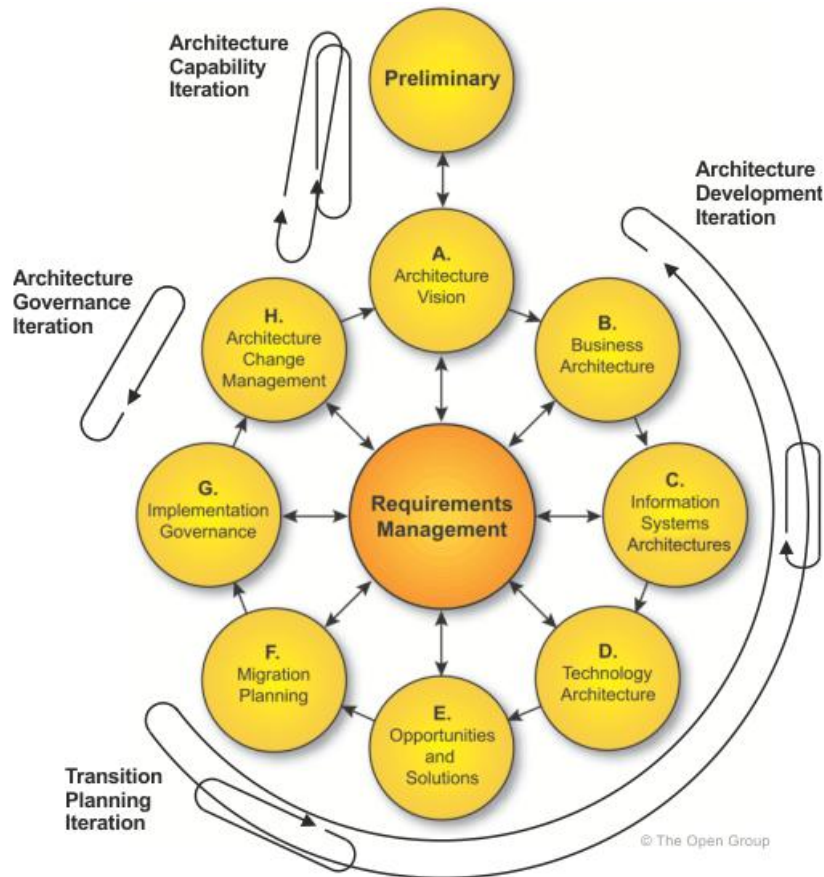
3.2 Architecture Governance

Architecture Governance is the practice that manages and controls architecture at an enterprise-wide level. It includes the implementation of controls of the creation and monitoring of all architectural outcomes and activities, compliance checks against enterprise standards and accountability to stakeholders. In addition, Architecture Governance establishes processes that support the successful execution of the mentioned activities. (The Open Group 2011.)

Architecture Governance is usually integrated to other management practices. Typical frameworks that are integrated with Architecture Governance are Balanced Scorecard, European Foundation for Quality Management (EFQM), ISO 9001:2000 standard, Control Objectives for Information Related Technology (COBIT), IT Infrastructure Library (ITIL) and Capability Maturity Model Integration (CMMI). (Lankhorst et al. 2009, 14 - 22; The Open Group 2011.)

In this thesis I will concentrate on the processes around making and maintaining the architecture documentation. TOGAF9 Architecture Development Method (ADM) gives a good foundation on both of these activities.

ADM is an iterative process which involves four main iterations: architecture development, transition planning, architecture governance and architecture capability (The Open Group 2011).



Picture 3: "Applying iteration to the ADM" (The Open Group 2011.)

In Architecture Development Iteration, architecture is developed through three main viewpoints: business, information systems and technology. In these viewpoints the current (Baseline) and future (Target) architecture is modeled. After modeling the current and future state architecture, it is possible to discover realistic ways to move from the current state to the future state (phase E) and later on plan the steps in which the future architecture is to be implemented (phase F). Phases E and F form the Transition Planning iteration. (The Open Group 2011.)

In Architecture Governance iteration the implementation projects are supported in order to reach the future state architecture as specified in the roadmap defined in Transition Planning. The function of Architecture Capability iteration is to mobilize the needed activities to establish or adjust the approach, principles, scopes, vision and governance of architecture. (The Open Group 2011.)

As Architecture Governance is the most interesting activity in this thesis I will open it in more detail. The main functions of architecture governance are the change management and imple-

mentation governance. In change management the upcoming changes are evaluated and categorized to three categories:

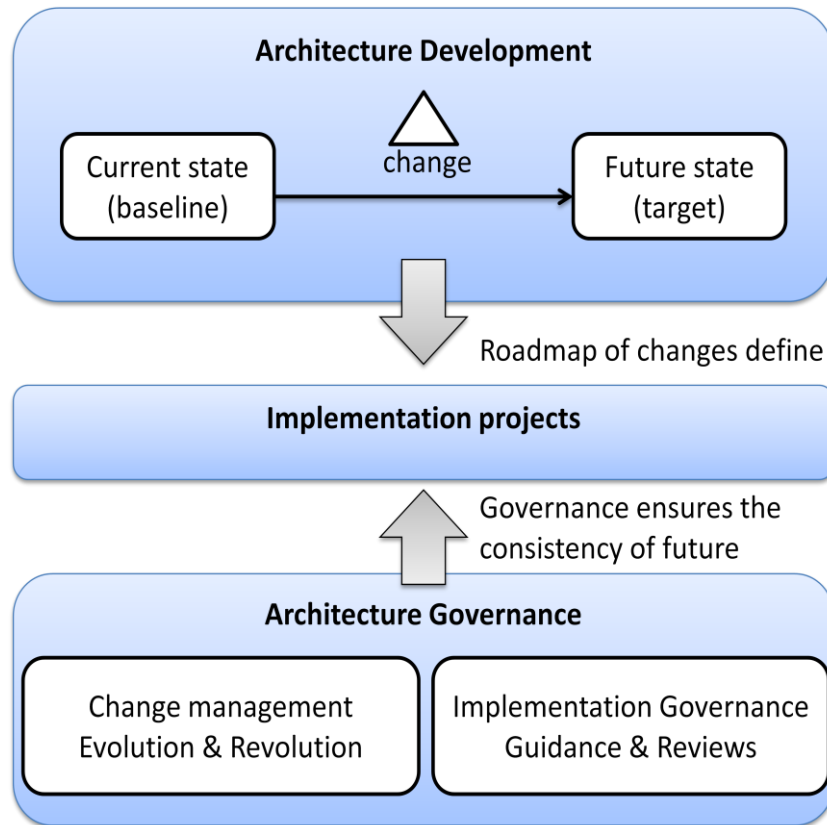
- Changes that simplify existing architecture.
- Changes that create an increment to a existing architecture.
- Changes that create a new architecture or significantly change the existing ones.

(The Open Group 2011).

Personally I call the first two ones as an evolution. It is something that you can manage with change management procedures and architecture maintenance activities. The last one is revolution; it is something that needs the use of Architecture Development procedures.

Implementation Governance is all about giving guidance to implementation projects and reviewing their outcomes against the architecture definitions. The depth of involvement depends always on the project in hand. Some projects need more governance than others. (The Open Group 2011.)

In Picture 4: From architecture to implementation, I have summarized the above. When we create something new or re-architect an existing solution we develop the future architecture and decide how we are going to get there through one or more implementation projects. On the other hand we have changes that evolve the architecture and might not necessary be a result of a large architecture development effort. Nevertheless we want to govern all implementation projects somehow in order to ensure the consistency of future.



Picture 4: From architecture to implementation

The literature over Architecture Governance involves also other sources of information than TOGAF9. Moser et al. presented a list of useful patterns for Enterprise Architecture Management (EAM). They have discovered processes that seem to reoccur in the real life. As a summary they have presented the following six process patterns:

- Centralized Manual Data Acquisition/Maintenance pattern.
- Decentralized Manual Data Acquisition/Maintenance pattern.
- Automatic Data Acquisition/Maintenance pattern.
- Architecture Control by Applying a Release Workflow pattern.
- Lifecycle Management pattern.
- Verification and Audit pattern.

(Moser et al. 2009.)

These patterns dealt with the problem space represented in this thesis. Especially the patterns helped in finding solutions to the efficiency of architecture description maintenance. Clements et al. (2011, 18 - 19) state that the work effort saved from using good architecture documentation should exceed the effort of creating and maintaining it. Thus the effective processes and governance is a must when we are dealing with architecture descriptions.

When we look outside of processes, Capilla et al. (2007) writes about the importance of documenting design decisions while making the architecture documentation in order for the maintenance to be successful. Clements et al. also highlight the importance of documenting decisions in architecture descriptions. One of the most important things is that documented decisions save work effort and result is usually a good quality documentation, which leads to effective maintenance of architectures. (Clements et al. 2011, 240 - 250.)

All in all, from Arek perspective, both effective processes and the importance of documenting decisions in architecture descriptions seem to play key role in successful maintenance of architecture descriptions. Both TOGAF9 and the work from Moser et al. were used in future state processes along with quality aspects of architecture descriptions, such as documenting decisions. The future state of both architecture documentation and maintenance processes is described in section 0 and its sub-chapters.

3.3 Software as a Service Business Model

We have discussed processes and architecture governance, but it is also important to understand the environment in which architecture governance is to be executed. Arek utilizes the SaaS business model and this model plays a key role also in the architecture supporting that kind of business. So what is SaaS?

Software as a Service business model is all about providing software to the customers without giving the actual ownership of the system to the customer. The software is provided to the customer as a service by using information technologies as a medium. As a concept Software as a Service is e.g. found in a document made by SIIA (2001), from where the concept and the actual business model have evolved.

Sääksjärvi et al. (2005) defines the term of Software as a Service (SaaS) as: “Software as a Service is time and location independent online access to a remotely managed server application, that permits concurrent utilization of the same application installation by a large number of independent users (customers), offers an attractive payment logic compared to the customer value received, and makes a continuous flow of new and innovative software possible.”

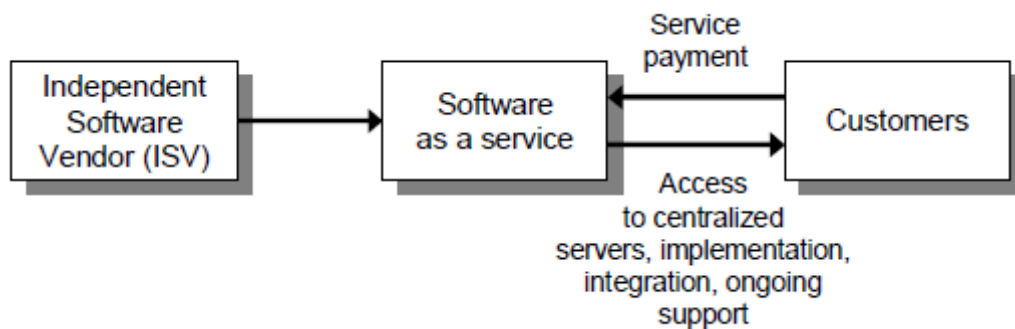
But what is the difference between SaaS and traditional ASP (Application Service Provider) models?

According to Lassila the differences are following:

- 1) SaaS bases itself more or less to e-commerce, where the ASP is based more or less on outsourcing.
- 2) SaaS makes a promise that mass customization of application services is possible.
- 3) SaaS is a concrete business model while the ASP is a more technically oriented concept.

(Lassila 2006.)

Lassila defines the generic SaaS model (Picture 5) that describes the basic nature of SaaS between independent software vendor (ISV) and the customer. Software vendors can deliver the software for a company that runs the software as a service business model. The ISV can also itself be the SaaS provider. Customers pay for the service that they receive. SaaS provider then gives access to centralized servers running the software. SaaS provider also takes care of implementation, integration and ongoing support of the provided services. (Lassila 2006.)



Picture 5: Generic SaaS model (Lassila 2006).

For the customers the SaaS model provides the possibility to focus on core business and innovation. It also gives easier access to professional software services and continuous updates to software services without extra cost in addition to default service payment. In addition to above the SaaS model promises to provide access to business critical services from anywhere, any time. (Hoch F. et al. 2001; SIIA 2001; Sääksjärvi et al. 2005; TripleTree 2004.)

For the service provider the benefits include economics of scale in service production and delivery and easiness of customer base expansion. The revenue estimation is also easier when compared to classical software business along with shorter sales cycle. There are also smaller costs for version control and maintenance. And one of the most interesting things is that suc-

cessfully designed services make it more difficult for competitors to enter the same service market. (Sääksjärvi et al. 2005.)

There are also some cons and risks associated for the software vendor utilizing the SaaS business model. Sääksjärvi et al. state that controlling a large subcontractor network might prove difficult and company revenue might decrease when moving to SaaS model, as you cannot any more collect license costs from customers using your products. From technical side the services might face scalability and performance issues and moving to SaaS model usually involves a large upfront investment to infrastructure and 3rd party licenses in order to be successful. The vendor is also required to comply with software update cycles. (Sääksjärvi et al. 2005.)

According to SIIA the three reasons why a software vendor should use SaaS business model are:

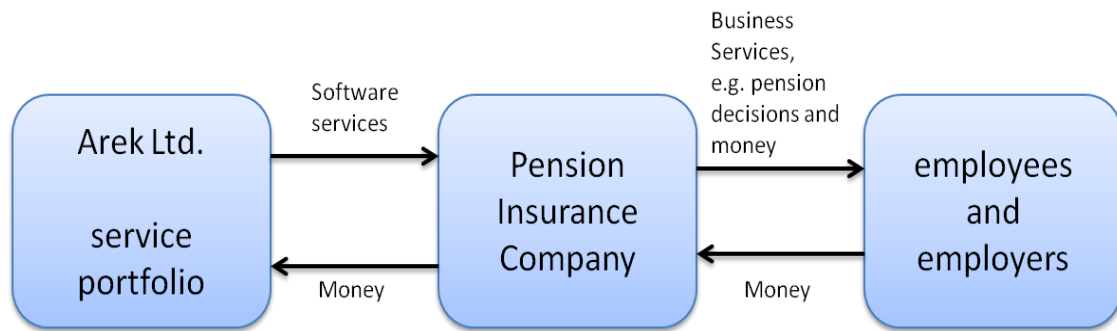
- 1) SaaS lowers the cost of bringing the software to customer.
- 2) Model is very tempting to software developers.
- 3) Potential expansion of customer base.

(SIIA 2001.)

3.3.1 SaaS at Arek Ltd.

Arek Ltd. provides over two hundred services to its customers by using SaaS business model. The services are used by the pension insurance companies and Finnish Centre for Pensions, all working in Finnish earnings-related pension sector. These services are core business services that these companies need in their every day work. The services are provided centrally from Arek servers. Arek customers are free to choose the services they need from the service portfolio provided by Arek. Usually customers enhance these services to provide additional value to their own customers through innovative solutions.

Earnings logic of Arek business is quite simple, service usage is charged based on actual usage. In addition to this each customer must pay fixed cost that allows customers to use any service they want from the service portfolio. Customers are also entitled to volume discounts: higher use of services gives you cheaper transaction prizes.



Picture 6: Arek SaaS model

Because of SaaS the concepts of service and product are important in Arek architecture descriptions. It is crucial to understand the services as they are bringing money to the company.

3.4 Service Oriented Architecture

The services provided by Arek are created by using the service oriented architecture style. SOA has been hype in the IT industry during recent years. The downside of hype is that there are misconceptions about what SOA is (Lankhorst et al. 2009, 43).

SOA is an architecture style. In SOA, there are distributed components that provide and consume services. The basic idea is that service providers and consumers can use different implementation languages and platforms. Services are usually deployed independently and often belong to other systems or even organizations. The key is to think in terms of services, utilize a service-based development style and understand the outcomes of services. The service thinking allows IT people to talk with business by using a mutual concept. The great promise is also the re-use of services in business processes. By thinking this way the business processes become the exploiters of lower-level business services provided by information systems (Clements et al. 2011, 169 – 172; Lankhorst et al. 2009, 46 - 47; The Open Group 2012.)

Service providers and consumers are not the only parts of SOA. Usually there are specialized components and related architecture patterns such as ESB, service registry and orchestration engine. All these provide more flexibility to SOA. SOA is especially good architecture style when you want to have high interoperability. This is achieved by allowing services to be built and accessed on different technologies and platforms (Clements et al. 2011, 169 – 172).

Lankhorst et al. note that the concept of service has risen also in the manufacturing industry and public service providers, SOA is not pure technology, it's a way of thinking. Marketing

and management is increasingly also focusing on the concept of service. (Lankhorst et. al. 2009, 44.) In the IT service management, ITIL also emphasizes services and service-level-agreements (itSMF 2007).

According to Open Group the service is a logical representation of a repeatable business activity that has a specified outcome. Service is also self-contained, it may be composed of other services and it is a “black box” to its consumers. (The Open Group 2012.)

In Arek the concept of SOA has already been successfully combined with the Software as a Service business model. In a nutshell the whole business of Arek runs on business services provided by using SOA ideology and profits collected by using SaaS business model.

3.5 Enterprise Architecture

While this thesis focuses more on software architecture than enterprise architecture, it’s very important to define the term enterprise architecture as the software architectures must be compatible with the architecture of the enterprise. Or more further, the extended enterprise, meaning also the stakeholder organizations.

IEEE 1471-2000 (IEEE 2000) defines architecture as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."

A widely used enterprise architecture framework, TOGAF, defines architecture with two different meanings; depending upon the context the word is used:

1. "A formal description of a system, or a detailed plan of the system at component level to guide its implementation."
2. "The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time."

(The Open Group 2011.)

TOGAF9 defines Enterprise as “The highest level (typically) of description of an organization and typically covers all missions and functions. An enterprise will often span multiple organizations.” and word Organization as “A self-contained unit of resources with line management

responsibility, goals, objectives, and measures. Organizations may include external parties and business partner organizations.” (The Open Group 2011.)

Lankhorst et al. state that Enterprise Architecture is architecture at the level of an entire organization and uses the following explanation for enterprise architecture: “a coherent whole of principles, methods, and models that are used in the design and realization of an enterprise’s organizational structure, business processes, information systems and infrastructure. (Lankhorst et al. 2009, 3.)

In the book *Enterprise Architecture as a Strategy*, the enterprise architecture is defined as “the organizing logic for business processes and IT infrastructure, reflecting the integration and standardization requirements of the company’s operating model.” And the operating model is defined as: “The operating model is the necessary level of business process integration and standardization for delivering goods and services to customers.” (Ross et al. 2006, 8 - 9.)

As the terminology hints, EA frameworks usually divide enterprise architecture into four smaller pieces of architecture: business, information, applications and technology. For example, TOGAF9 has business architecture, information systems architecture (consisting of data architecture and information applications architecture) and technology architecture (The Open Group 2011). This thesis focuses itself on information systems architecture. As the terminology is somewhat still developing in the industry, I will use the established explanation of Enterprise Architecture at Arek: “A documented description of an Enterprise and its goals, business processes required to meet the goals, information systems supporting business processes, technologies and methods used to support development of information systems. EA is the glue between other Architecture concepts.” This explanation is based on the fact that company must have goals together with integrated and standardized ways to meet those goals.

As there are many definitions for enterprise architecture, there are also many frameworks designed to support it, in this thesis I will present in brief TOGAF (see chapter 0) and Zachman framework (see chapter 3.7.8) as these are probably the most used ones. I think these frameworks should be used also in the systems architecture work; otherwise you cannot reach the integrated, strategic view of the company. The systems and their architecture must be able to integrate into each other and also to the business they are supporting. As such, it is also important to know the business model in order to create successful architecture descriptions that

meet the requirements of the stakeholders. For these reasons the business model of Arek is revealed in chapter 3.3.

Lankhorst et al. state that in the future the role of architects and architecture descriptions increasingly includes the business network outside of traditional enterprise. This is called the extended enterprise. In his case study with some government organizations Lankhorst et al. discovered that there is an increasing need for cross organizational architectures. (Lankhorst et al. 2009, 322 - 325.) TOGAF9 states that nowadays extended enterprise frequently includes, partners, suppliers and customers (The Open Group 2011). When we look at our case organization (chapter 4.1) with its business model (chapter 3.3) and the trend mentioned above it was very important to involve stakeholders from extended enterprise context in the definition of future state architecture descriptions as presented in chapter 0.

3.6 Software Architecture Documentation

As there is no single definition for enterprise architecture, neither there is any single definition for software architecture. Actually, according to SEI (2011) there are lots of them and SEI is currently collecting new definitions for the term software architecture.

One of the classical definitions is that the software architecture is "the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time." This definition was created by David Garlan and Dewayne Perry in 1995. (SEI 2011.)

Thus I decided not to use any classic definition in this part of my thesis as Arek already had one definition also for software architecture. Although this term is called system architecture, actually it is a synonym for software architecture. The system architecture is defined as a documented description of a system embodied in its components, their relations to each other, and to the environment, and the principles guiding its design, development and operation. The definition is actually a derived combination of the existing definitions, but it suits Arek better than the numerous other definitions.

The basic need for software architecture rises from the fact that the complexity of software-intensive systems outpaces our collective ability to deal with that complexity (Dashofy 2007, 8). The earliest efforts to create some sort of architecture appeared in the McIlroy's seminar paper at 1968. McIlroy explained that the software should be viewed above the code lines and

simple modules. (McIlroy 1968.) In 1974 David Parnas found out that software consists of many structures and relations between them (Clements et al. 2011, 24). Later on, in the 1976, DeRemer and Kron made distinction between what they called “programming in the large” versus “programming in the small”. Their idea of abstracting the software as called “programming in the large” introduced abstraction and even an architecture description language, MIL, Module Interconnection Language. (DeRemer & Kron 1976.) The modern conception of software architecture is based on DeWayne Perry's and Alexander Wolf's seminar paper that introduced concepts such as element (processing, data, and connections), form (how elements may be arranged and configured) and rationale (the reason behind the design). They also recognized that the structures originally found by Parnas had similarity to architecture of real world buildings. Such structures could then be visualized by using views tailored for different stakeholders. (Perry & Wolf 1992.) In 1994 Philippe Kruchten developed a “4+1” approach to architecture, which was claimed to be the foundation for the Rational Unified Process. Since then several viewpoint models have emerged and in the year 2000 IEEE published a standard that defined the concept of view and viewpoint in coherent way. (Clements et al. 2011, 23 - 25; Dashofy 2007,8.)

However there are no standard structures in computer systems. This fact was found out by Smolander et al. (2002) during a research in Finland that aimed to find out the relevant documented structures of a computer system. There is no globally agreed structure and this fact is admitted also by the IEEE, thus their recommended practice (IEEE 2000) makes no commitments on what viewpoints or structures should be documented from a computer system.

Nevertheless there are some structural elements that are more or less agreed on. Usually a information system is built from smaller pieces, called e.g. modules, that make up the actual system. These modules can be aggregated to concepts called as subsystem. As the terms are vague in higher abstraction level, so are they at this level also. There is no actual agreement on what a subsystem is, but the subsystem often has three features: it carries a subset of functionality from the overall system, it can be executed independently from the overall system and it can be deployed and developed incrementally. For example a planet exploration robot can have subsystems of communication, motion, power management, navigation and monitoring. The robot might also have a math utility library that is used in the overall system, but it is not a subsystem; it cannot be operated independently to do something that is recognizably part of the actual robots purpose. (Clements et al. 2011, 73 – 74.)

As Software Architecture documentation deals with components and their relations, a important concepts is needed. This concept is interface. Interface is a boundary across which two elements meet and interact or communicate with each other. Interfaces are highly important concept in architecture. Without them you cannot build systems or perform analyses that are based on architecture descriptions. An interface describes and defines how and what other elements can do with the element that provided the interface. The following principles are related to interfaces and element that provide them:

- No element exists without an interface.
- Interface is a separate thing from its implementation.
- An element can have several interfaces.
- Elements both provide, but can also require interfaces.
- Interface can be used by multiple users at the same time.
- Interfaces are extensible.

(Clements et al. 2011, 262 – 267)

When we look SOA style of architecture, a service is usually treated like an interface. To my opinion the service can be provided through an interface as long as it complies with the nature of service as described in chapter 3.4.

While there are no standards for system structures, there are modeling frameworks that help in creating system architectures. The concept of implicit and explicit viewpoint models helps in the classification on existing architecture modeling frameworks. Explicit viewpoint models can be found from standards like RM-ODP and from literature, like the Kruchten's 4+1 model. The implicit viewpoint model does not give explicit viewpoints, but instead the viewpoint model can be inferred by analyzing diagram types and modeling elements used in modeling. An example of this type of viewpoint model is the UML standard. In order to create views the architect needs to have an either explicit or implicit framework that defines the relevant considerable aspects of software architecture. (Smolander et al. 2002.)

Viewpoints are basically agreements between people and architecture consists of views created from different viewpoints. When we implement a computer system from requirements to actual implementation we basically make agreements between people participating in the project. Likewise when we choose viewpoints we make agreements on what viewpoints are intuitively relevant to the stakeholders. We must also make agreement on how these viewpoints

should be modeled. A model is also an abstraction of an actual world and thus it's another agreement between people. (Smolander et al. 2002.)

One typical abstract viewpoint is describing software architecture structure as components and connectors. In this viewpoint people agree on relevant components of software, connections between these components and notations associated to the components and connectors. (Clements et al. 2011, 123.)

Because all viewpoints are agreements between people, no one can state that one set of viewpoints is better than the other. Instead the viewpoints relate heavily to the environment in which they are used. Most of these viewpoints are general to the whole organization, but some might be relevant only to a specific development project. But while the viewpoints vary by organization or even project, there are some factors that must be considered when architecture descriptions are made. First of all the descriptions are used in communication with different stakeholders. For this reason a description must be readable and it must answer the needs of the stakeholders. (Smolander et al. 2002)

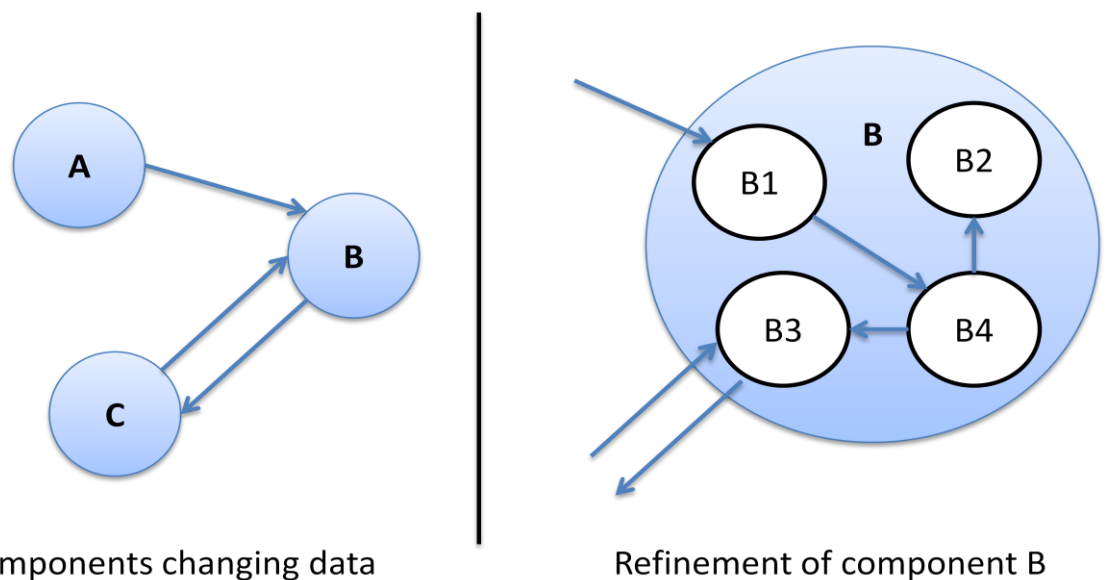
In order to ensure a good readability the architect has to find out who will read the description. All insider jargons and acronyms must be avoided, unless these are explained in glossary. The description structure should be planned beforehand and all kinds of repetition should be kept in minimum, otherwise the description becomes hard to maintain. On the other hand readers don't want to flip pages or click too much hyperlinks either. A good way to ensure that the documentation is written from the user point is that it must give answers to the most common questions raised by the readers of the document. It is also advised to create models iteratively and take the feedback of users into account. While writing the description it is important to show only stable intermediate versions to the readers, architect should not model something that is incorrect. (Clements et al. 2011, 36 – 45; Lankhorst et al. 2009, 131 - 141.)

All drawings should have an explanation of the used notation, the reader cannot be assumed to know the used notation. Number of drawings and models should also be kept decent, the more architect models the more bloated description becomes. While making views and drawing images, architects should use layers and group drawn elements e.g. by service, information and/or physical distribution in order to make the pictures more readable. The abstraction levels of the models should always be kept consistent and separate clearly internal and external behavior of the elements. The models are less complex if the number of elements, types of

elements and relations are reduced. The reader should see only what he or she needs; there is no need to model too much detailed information if it is not needed. (Clements et al. 2011, 36 – 45; Lankhorst et al 2009, 131 - 152.)

But how can an architect know what is small enough number of elements?

Horton (1991) studied that humans can work effectively with models that do not have more than thirty elements. This is a good rule of thumb to remember when making models. If there are more than thirty elements, it might be an indication that something should be abstracted in the model. Miller (1956) stated also an important fact about short-term capacity of human memory. Humans can generally process seven plus or minus two elements at time in short-term memory. This is a limitation that should be kept in mind when explaining complex systems in a model. Layering and generalization help in making the models simpler. Koning et al. (2002) has thus stated that overview of the model faints if there are more than three levels of abstraction. (Lankhorst et al. 2009, 144 – 152.) Clements et al. (2011, 218) presents a process called decomposition refinement, which deals with this abstraction issue. The idea is to hide details. In the left side of Picture 7 three components A, B and C send data to each other. A refinement (“zoom”) of component B is shown in the right side of the picture. This kind of refinement is a good way to create abstraction views.



Picture 7: Decomposition Refinement

There are also some other tricks that architects can use in modeling. The Gestalt theory of human perception can be used to deceive the limitations of mental capacity of humans. This theory states that humans tend to make some assumptions when they see pictures. Humans

treat same colored or otherwise similar objects relate to each other or belong to a same group. Also objects near each other are associated to have some kind of relation between them. A line establishes a directions and crossing lines are bisecting each other. Bigger objects are treated more important than smaller ones. The direction, facing or movement of objects tends also to create associations in the human mind. While modeling tools have no understanding of these things, a careful use of these tricks might help architect in communication with the stakeholders. (Lankhorst et al. 2009, 144 - 152.)

Readers usually also want to know why the architecture is the way it is. For this reason, a rationale must be written in the document. And in the end, description should always be reviewed by the stakeholders to ensure its usefulness. When description is ready, it must be kept current by using a process and release schedule; otherwise it will lose its meaning. (Clements et al. 2011, 36 – 45; Lankhorst et al 2009, 131 - 141.)

Besides of readability, organizational factors like the need to divide work, manage complexity, guide the developers or use of certain development methods such as Rational Unified Process or Scrum should be taken into account when making architecture descriptions. (Smolander et al. 2002.)

Technical environment and quality attributes like security are also factors that greatly influence architecture descriptions. Especially quality aspects have a lot of influence to architecture as each element in architecture description also has some quality aspects, such as performance or security properties. The used architectural style also has some affect, a style such as SOA or client-server architecture has different effects to quality. It should also be noted that quality attributes usually come from requirements that must be met. As such the importance of different quality attributes is depending of the organization and the situation at hand. (Clements et al. 2011, 17-18; Smolander et al. 2002.)

3.7 Available viewpoint frameworks

As the reasons for forming different viewpoints vary a lot, there are numerous available viewpoint models in the industry and literature (Smolander et al., 2002). It should be noted that there is increasing pressure to form a group of de jure and de facto standards for guiding the architecture modeling. Often the developers of these standards call their creations as frameworks. These frameworks contain viewpoints or perspectives, from which software architectures should be described. (Dashofy 2007, 40).

As described earlier, UML could be also seen as viewpoint framework, because it also describes several perspectives, diagrams, from which one can model information systems. There is one primary distinction why UML is not an architecture framework, it defines individual notations for each viewpoint and does not give guidance on what to capture in architecture descriptions. (Dashofy 2007, 41.) Probably for this reason, Smolander et al. (2002) called UML as implicit viewpoint model, compared to explicit ones described in the following sub-chapters.

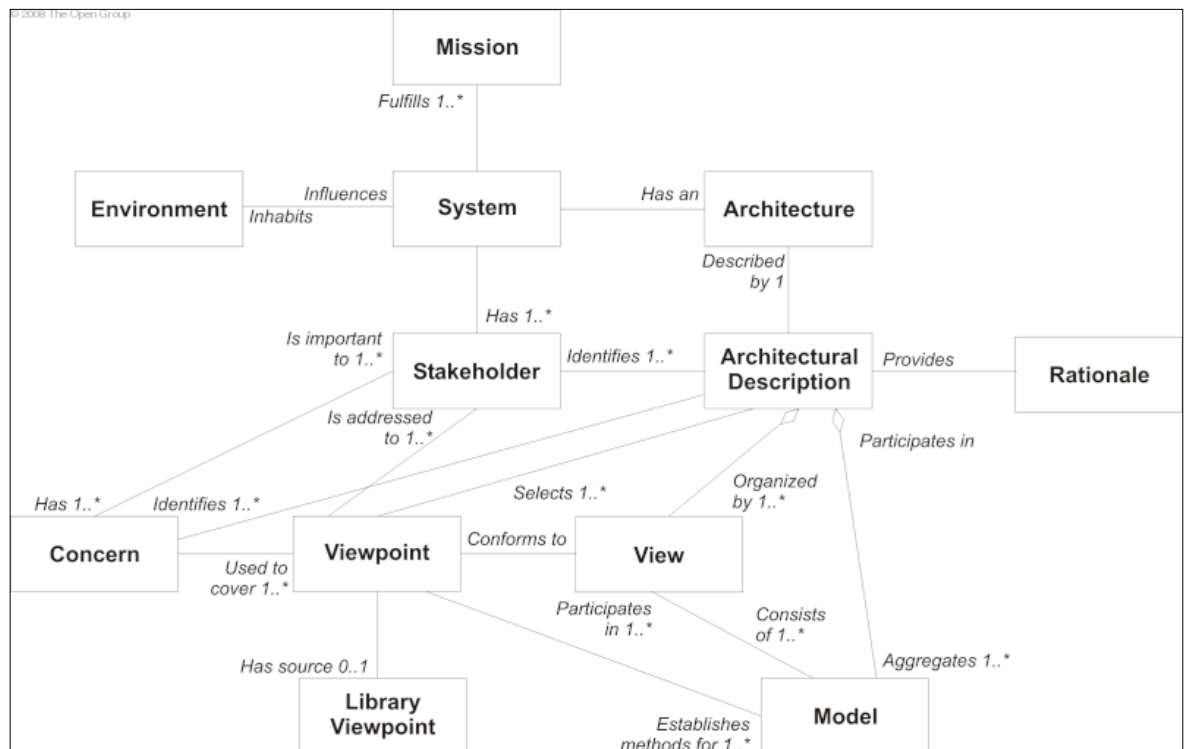
The frameworks listed here guide architects on what to capture, but some of them even give guidance on how to do it, while others use UML as notation or strongly imply to that direction. In the following sub-chapters the most used frameworks are described. It should be noted that the list is not all-inclusive there are numerous other viewpoint frameworks existing. The ones listed here are based on work of Clements et al. (2011) and Smolander et al. (2002). TOGAF and Archimate are included as they are essential when combining the enterprise architecture and software architecture disciplines.

3.7.1 IEEE 1471-2000 / ISO/IEC 42010 Standard

In 2000 IEEE Computer Society approved IEEE Standard 1471-2000, which introduced a theoretical base for the definition, analysis and description of software architectures. Later on this standard was adopted by ISO in the year 2007 as ISO/IEC 42010. (IEEE 2000; Lankhorst et al. 2009, 23 - 25.)

IEEE standard 1471-2000 claims to define content for architectural descriptions, but actually what it gives is the definition that the architecture description includes several views that comply with the defined viewpoints. The standard also helps in giving advice on how an architect should define these viewpoints. The core of the standard is a conceptual model that states that

the viewpoints must be selected based on the needs and concerns of stakeholders. (IEEE 2000.)



Picture 8: IEEE 1471-2000 conceptual model of architecture description (The Open Group 2011).

IEEE 1471-2000 does not try to standardize architecture processes, yet it does not recommend any modeling languages or standards either. The standard does, however introduce some valuable concepts and terms to be used in architecture descriptions. These concepts are explained in Table 2.

Table 2: IEEE 1471-2000 concept definitions (IEEE 2000).

Concept	Definition
System	Individual applications, systems, subsystems, systems of systems, whole enterprises or other interesting aggregations
Environment (or context)	Settings and circumstances that influence the system of interest. The environment can also include other systems that interact with the system of interest. Environment defines the boundaries that define the scope of the system of interest relative to other existing systems.
Stakeholder	A stakeholder that is somehow involved with the system.
Concern	Pertaining interests to the systems development that are important to stakeholders. Includes system considerations such as performance, reliability and security.
Mission	Explains why how the system is intended to be used by one or more stakeholders.
Architecture	"The fundamental organization of a system embodied in its components,

	their relationships to each other, and to the environment, and the principles guiding its design and evolution."
Architectural Description	A recorded architecture. Concrete product or artifact.
View	Addresses one or more concerns of the system stakeholders.
Viewpoint	Establishes the conventions by which a view is created, depicted and analyzed. Viewpoint defines descriptive languages, methods and analysis techniques for a view.
Library Viewpoint	A viewpoint defined outside of architecture description.
Model (or architectural model)	A architectural model is developed by using the methods explained in the viewpoint.
Rationale	Architecture description must provide rationale for the system.

The concepts used in IEEE 1471-2000 / ISO/IEC 42010 are very useful and widespread in the industry. Because of this the future state architecture description formed for Arek in this thesis was compliant with the standard and its concepts.

3.7.2 RM-ODP

Reference model of open distributed processing (RM-ODP) was developed in order to provide a framework to support open distributed processing, especially when it comes to large distributed systems (Lankhorst et al 2009, 32).

RM-ODP provides five viewpoints: enterprise, information, computational, engineering and technology. RM-ODP defines a description language for each viewpoint and it also offers a way to organize the models to the viewpoints. (ISO, 1998.)

The viewpoints contain the following information as described in standard:

- "The enterprise viewpoint: A viewpoint on the system and its environment that focuses on the purpose, scope and policies for the system."
- "The information viewpoint: A viewpoint on the system and its environment that focuses on the semantics of the information and information processing performed."
- "The computational viewpoint: A viewpoint on the system and its environment that enables distribution through functional decomposition of the system into objects which interact at interfaces."
- "The engineering viewpoint: A viewpoint on the system and its environment that focuses on the mechanisms and functions required to support distributed interaction between objects in the system."

- "The technology viewpoint: A viewpoint on the system and its environment that focuses on the choice of technology in that system."

(ISO, 1998.)

According to Dashofy the viewpoints of RM-ODP vary in level of concreteness. Computational and engineering viewpoints are quite concrete and detailed, whereas the enterprise viewpoint is abstract. RM-ODP does not give any concrete notation that could be used in making the viewpoints; instead it gives requirements, in the form of characteristics, to a notation and the viewpoint content. (Dashofy 2007, 43.)

Lankhorst et al. add that the RM-ODP viewpoints also lack the association of viewpoints to specific stakeholders. By not showing this the RM-ODP does not reveal the concerns which the viewpoint aims to address. (Lankhorst et al. 2009, 158.)

When we look Arek as a company and how RM-ODP might suit to Arek's needs the actual contribution might be the definitions of viewpoints and concepts. But these kinds of definitions exist also in the other frameworks, and because RM-ODP does not give any ready notation to build solutions, its value as a framework declines. From Enterprise Architecture point of view the framework is also too detailed in computational and engineering viewpoint level. These kinds of details do not work well at architecture level from the maintenance point of view, or they might create a high maintenance burden. And Arek already has well grounded ways of describing detailed information from a system. The lack of stakeholder association can also be considered a con, as the IEEE Standard 1471-2000 has the concerns of stakeholders in great value.

3.7.3 DoDAF

DoDAF is a system architecture documentation standard used by Department of Defense in the United States. Its previous version was the C4ISR framework. DoDAF has three viewpoints. There is an interesting detail in DoDAF, it calls viewpoints as views. In original terms the views (viewpoints) are:

- "Operational view (OV), which defines the objectives needed to be accomplished and defines who should accomplish those objectives. The key components are processes, activities, operational elements used in activities and information change between elements."
- "System view (SV), which defines systems that provide or support operational activity. Systems in SV are interconnected to other systems and associated with the elements in OV."
- "Technical standards view (TV), which defines standard engineering guidelines, rules, conventions and other information intended to ensure that the system requirements are met."

(Dashofy 2007, 45.)

DoDAF has also some cross-cutting concerns that affect all views (AV). DoDAF does not give any rules on how and what notation should be used in each view. Actually users can choose any subset of viewpoints they want and document them by any way they might feel right. DoDAF does, however, indicate that the use of UML diagrams might be a good choice. (Dashofy 2007, 45 - 48.)

As DoDAF is intended for a military organization, its use in peace loving company like Arek would be a very interesting decision. Although, Lankhorst et al. (2009, 31) state that DoDAF can be extended to a more general purpose, but still it seemed quite a big and time consuming task to try to extend DoDAF for Arek. As such it was actually not a solution to challenges faced by Arek, but it does play significant role in the field of viewpoint frameworks. It should be noted, that solutions from military industry have somewhat appeared in our everyday life, for example the ARPANET was the roots of modern Internet in the end of 1960's. For this reason, I think that DoDAF will also continue have some affect in the field of systems architecture.

3.7.4 MDA

The goal of Model Driven Architecture (MDA) is to provide an open vendor-neutral approach to interoperability. It is built upon Open Management Group's (OMG) standards: Unified Modeling Language (UML), Meta Object Facility (MOF) and the Common Warehouse Meta-model (CWM). The application descriptions created with these standards can be implemented with various open or proprietary technologies, such as CORBA, Java, .NET, XML and Web Services. The goal of MDA is to raise abstraction level on which software is designed and promising that code can be generated from MDA models and vice-versa.

(Lankhorst et al. 2009, 29 - 31.)

MDA has three levels of abstraction with mapping between them:

- Computation-Independent Model (CIM), which includes a domain model, a business model and business requirements.
- Platform-Independent Model (PIM), which includes system specifications independent from technical platform, these are basically UML models of the system and business processes modeled with BPMN.
- Platform-Specific Model (PSM), which included platform specific specifications, such as UML models used in Java-platform and WS-BPEL created from BPMN.

(Lankhorst et al. 2009, 29 - 31.)

One of the key features of MDA is the notion of mapping between abstraction levels. The mapping is basically the rules on which, for example a CIM can be translated to PIM and vice-versa. It should be noted though that these mappings are still partially under research.

(Lankhorst et al. 2009, 30 – 31.)

When we look critically at MDA, yes it's promising, but it seems too idealistic and incomplete to be used in Arek. The mappings are incomplete and it seems quite hard, in common sense way of thinking, to create 1:1 mappings between CIM, PIM and PSM. For these reasons a MDA is not an option in Arek.

3.7.5 Rational Unified Process/Kruchten 4+1

Rational Unified Process as a development method contains a 4+1 view model originally created by Philip Kruchten (1995). 4+1 view model defines a general viewpoint framework for architecting computer systems using five viewpoints.

The viewpoints of the 4+1 view model are:

- Logical Viewpoint that contains functional requirements for the architecture.
- Process Viewpoint that address issues such as concurrency, distribution, integrity and fault tolerance.
- Development Viewpoint that explains how the system software handled in software development environment.
- Physical Viewpoint that describes how the software elements are deployed on hardware devices / nodes.
- Use Case Viewpoint that captures the use cases of the system.

(Dashofy 2007, 41 - 42.)

The 4+1 view model actually contains its own notation, but as Dashofy (2007, 42) says the notation is incompletely described, with only a paragraph or two. So the actual contribution of the model is the views and their content. In my own experience I have seen mostly UML notation used with the 4+1 view model.

Smolander et al. (2002) states that the 4+1 view model is usually used in environments where general viewpoint model is useful. After thinking, there seemed to be no reason why Arek should change its own already existing viewpoint model to a more generic one. This is because it lacks viewpoint that are useful for an IT service provider that bases its business on a SaaS business model.

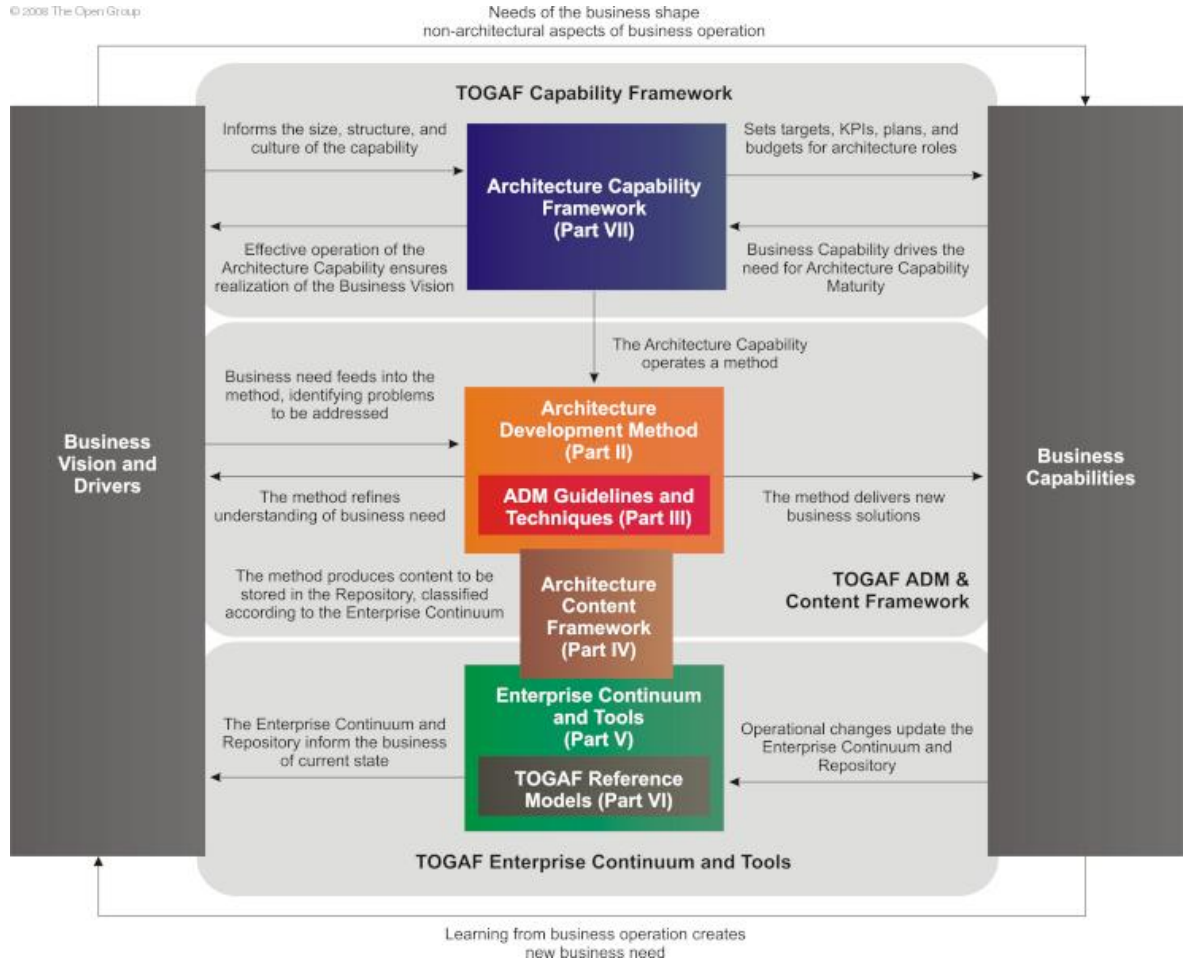
3.7.6 TOGAF

The Open Group Architecture Framework (TOGAF) is an enterprise architecture framework and method, yet it also suits and supports the architecture descriptions of systems.

TOGAF builds itself on four main components:

- Architecture Capability Framework, which specifies the organization, processes, skills, roles and responsibilities required to operate architecture function within an enterprise.
- Architecture Development Method (ADM), which is the core of TOGAF, it provides a cyclic method that architects can use in their work.
- Architecture Content Framework, which defines a way to split architecture to four closely interrelated architectures: Business Architecture, Data Architecture, Application Architecture and Technology Architecture.
- Enterprise Continuum, which provides reference models and shows how one can categorize solutions for later use, by dividing them from common foundation architectures to enterprise specific architectures.

(The Open Group 2011.)



Picture 9: TOGAF 9 (The Open Group 2011).

As part of Architecture Content Framework, TOGAF provides viewpoints in four different categories: business, information systems, technology and composite. Within these categories, the viewpoints are classified as

- Catalogs, which are lists of architecture building blocks.
- Matrices, which explain the relationships between building blocks.
- Diagrams, which give graphical presentations of building blocks suitable for stakeholder communication.

(The Open Group 2011.)

The content of TOGAF view categories is the following:

- Business Architecture viewpoints concentrates on the concerns of people, how the business information is used in the business processes that people take part of. And how people use information systems.
- Information Systems Architecture viewpoints divide into two categories, Data Architecture and Application Architecture. The main groups targeted for these views are

system engineers, database designers and administrators. The main concern is how a system is to be implemented and maintained.

- Technology Architecture viewpoints take care of concerns of system managers, operators and acquirers. Technology architecture concerns focus around standards, networks, servers and technology portfolio.
- Composite viewpoints that combine the viewpoints mentioned above. An example of composite viewpoints are the manageability and security viewpoints. Thus these composite viewpoints are mainly targeted for administrators, managers and operators.

(The Open Group 2011.)

For Arek, TOGAF gives good viewpoint candidates. Especially the Information Systems Architecture viewpoints have details that are very useful to Arek. These details have been more or less taken into account when the content of future state architecture description example was developed (look section 4.3.1 for the making of future state description).

3.7.7 Views and Beyond approach

Views and Beyond approach is presented in a book Documenting Software Architectures. The book tells about architectural styles and views associated in those views. These views are presented in three categories: module styles, component-and-connector styles and allocation styles. (Clements et al. 2011, 49.)

The module styles represent modules, which are units of software providing a coherent set of responsibilities and their interrelationships. The book represents six sub-styles which can be used e.g. to create a data model for information system. These styles are used to create a high level picture of the system and to provide some input for project work assignments. (Clements et al. 2011, 65-67)

Component-and-connector styles represent components which are principal processing units and data stores. Components have ports and interact with other components by using connectors. These styles are used to represent how the system actually works and how quality attributes, like performance and reliability, are intended to be implemented in the system. (Clements et al. 2011, 126-127.)

Allocation styles map software to its environment, like servers. These styles help to analyze quality attributes and cost estimates of the system. The styles can also be used to discover runtime dependencies of the system. (Clements et al. 2011, 189 – 194.)

It is sometimes useful to combine different views in order to make the system under documentation easier to understand. Creating combined views also has a risk if you are not sure what you are doing you might create too complex and confusing views that do not serve the stakeholders of architecture documentation. (Clements et al. 2011, 252-254.)

When we look what Arek might get from Views and Beyond approach we can discover that the styles represented are very general and used in wide variety of existing viewpoint frameworks that are represented in this thesis. However the Views and Beyond is a very useful approach that should be looked after when you want to keep the architecture descriptions as simple as possible even if you might use combined views based on other viewpoint models.

3.7.8 Zachman framework

Zachman framework is a logical structure for classifying and organizing descriptions of an enterprise that are meaningful to the management and development of the enterprise. It was the first enterprise architecture framework and it was named after its creator John Zachman (1987).

The Framework is actually a matrix that answers six questions on different levels of detail. The framework is quite easy to understand and it addresses the enterprise as a whole. It's also independent from technology, tools and methodologies. Cons are that the framework has many cells and there is no exact specification on how these cells actually interact with each other (Lankhorst et al. 2009, 25-26).

	Why	How	What	Who	Where	When
Contextual	Goal List	Process List	Material List	Organizational Unit & Role List	Geographical Locations List	Event List
Conceptual	Goal Relationship	Process Model	Entity Relationship Model	Organizational Unit & Role Rel. Model	Locations Model	Event Model
Logical	Rules Diagram	Process Diagram	Data Model Diagram	Role relationship Diagram	Locations Diagram	Event Diagram
Physical	Rules Specification	Process Function Specification	Data Entity Specification	Role Specification	Location Specification	Event Specification
Detailed	Rules Details	Process Details	Data Details	Role Details	Location details	Event Details

Picture 10: Zachman framework (Wikipedia 2012a).

Zachman Framework is useful for Arek, when architects deal with issues they can map these issues against the matrix and discover where these issues stand in organization. The matrix can also be used discover useful viewpoints for the stakeholders.

3.7.9 Archimate

Archimate is a modeling language for architecture, but it also provides some viewpoints that are mainly meant for describing Enterprise Architecture. Some of those views are also suitable for modeling systems architecture.

Archimate provides the following sixteen viewpoints with different meanings of use: introductory, actor co-operation, business process co-operation, application co-operation, organization, business function, business process, information structure, application behavior, application structure, infrastructure, service realization, implementation and deployment, product, application usage, and infrastructure usage (Lankhorst et al. 2009, 178).

When we compare these available viewpoints to the goal of this thesis, we can find some useful viewpoints that can be adapted for use in Arek as described in section 4.3.1. Probably the most suitable for system architecture level descriptions are introductory, product, application co-operation, application structure, application usage, infrastructure usage, implementation & deployment and information structure viewpoints. Other viewpoints are mostly interesting from Enterprise Architecture point of view. It should be noted that the provided viewpoints

are most of the time modified to suit the needs of an organization. It is most unwise to take viewpoints as provided as they might not answer the relevant concerns of stakeholders.

3.8 Example, selected viewpoints in three telecom industry companies

To get an understanding on what kind of viewpoints are used in other companies a study of Smolander et al. was analyzed. Smolander et al. (2002) studied three organizations in telecom industry in order to find out what kind of viewpoints do organization really use. The study was a qualitative research and the results are summarized in Table 3.

Table 3: Comparison of viewpoints in three companies (Smolander et. al. 2002).

	Company A	Company B	Company C
Primary business area	System integrator	Mobile software company	Telecom Service Provider
Selected viewpoints	1 Logical viewpoint (subsystems and their responsibilities, dependencies, interfaces) 2 Physical viewpoint (hardware topology) 3 Process viewpoint (concurrent processes and their responsibilities, communication between processes, process distribution) 4 Technical viewpoint (software components, detailed interfaces, deployment to physical topology, product specific information, layering, used libraries, 3 rd party components)	1 Static structure viewpoint (components, dependencies, interfaces, used resources, system services) 2 Information viewpoint (users/actors, data structures, databases, files, persistence) 3 Dynamic structure viewpoint (information flows, instantiation, resources, threads, processes, data communication) 4 Business viewpoint (cost efficiency, development speed, partnerships, subcontractors, COTS components, IPR, patents) 5 Engineering viewpoint (quality, complexity, reusability, tools, methods) 6 Technology viewpoint (competence, product life cycle and lifetime estimate, technology complexity and stability, licensor and licensee relations, standards and regulations, RAM/ROM budget and measurements.	1 4+1 viewpoint model 2 Enterprise viewpoint (system context, policies, earning logic, purpose) 3 Development organization viewpoint (applied resources, process and project models) 4 Production viewpoint (Technical issues of the system's lifecycle after release)
Additional	The most used viewpoints	Viewpoints differ a lot from	The organization was

remarks	<p>were logical and technical viewpoints.</p> <p>The viewpoint model was very generic, possibly due to the software integrator business; a system integrator must adapt to customer environments and their infrastructure.</p>	<p>Company A; this is due to the requirements driven from used development platform (EPOC) and mobile networks. Especially technology viewpoint is formed from mobile standards and regulations and limited set of capacity in mobile devices.</p> <p>Also the legal issues and competition with tight schedule bring demands to viewpoints.</p>	<p>fairly young in the process of finding its place in the organization.</p> <p>High degree of integration among systems and services were also present.</p> <p>In this company the software is mostly run on server side, thus the production organization was seen as principal stakeholder.</p> <p>The organization was also responsible of driving the reuse of developed components.</p>
----------------	--	--	---

Smolander et al. made some observations based on these three companies. The business model affects the choice of viewpoints. The interesting fact is that it seemed that the non-technical factors had greater influence to the chosen viewpoint model than the technical factors. All of the companies had a component-and-connector style viewpoint in use. As there were only three companies at hand it should be noted that a definite argument cannot be made based on this study. But nevertheless it gives some insight on the factors that influence the choice of viewpoint framework. (Smolander et al. 2002.)

3.9 Notations used in Architecture Descriptions

The notations for architecture documentation can be divided into three categories:

- Informal notations are usually graphic presentations made by using general-purpose diagramming and editing tools. Informal notation do not usually have any common form of representation, it's very common to have different visualizations for different systems. Semantics in informal notations are characterized by using natural language and formal analysis is usually an impossible task to do.
- Semiformal notations are expressed by a standardized notation, but the semantic meaning of each element is incomplete. Rudimentary analysis can be used when architecture descriptions are analyzed. UML is a typical example of semiformal notation when it is applied to architecture descriptions.
- Formal notations have a precise semantics and these notations can be analyzed both by syntax and semantics. Formal notations are usually referred as Architecture Description Languages (ADLs). Some of these notations suit only to certain types of

software architectures, but some allow also flexible use in different types of architectures. The usefulness of ADLs lies in the fact that they support automations, such as code generation or automatic analyses.

(Clements et al. 2011, 53)

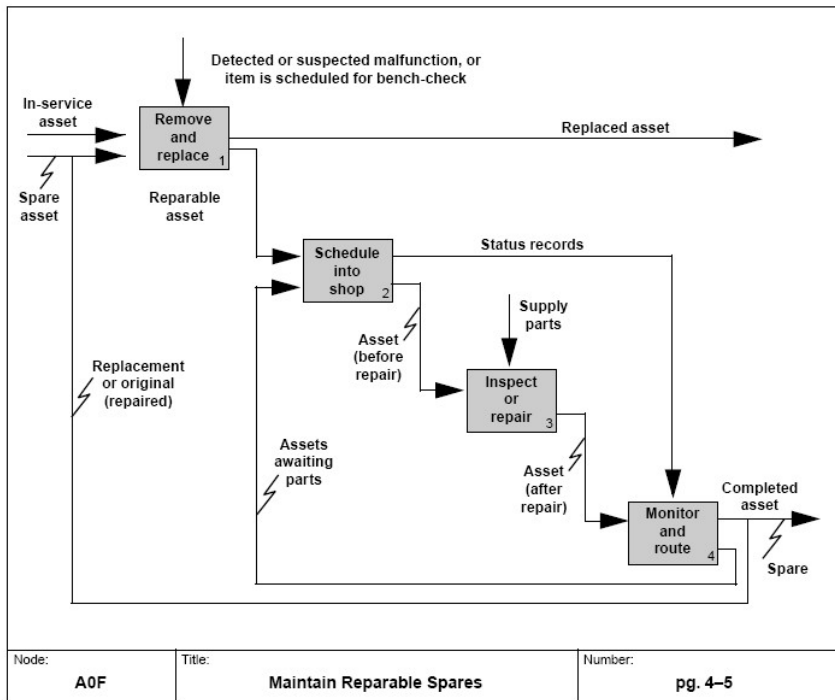
The decisions between notations is a trade-off situation. The more-formal notations take more time and effort to take into use, but they also give possibility to do automated analysis and reduce the space for misunderstandings. On the other hand, more-informal notations are easier to create, but they might be incomplete and raise a lot of questions about the documented solution. (Clements et al. 2011, 53.)

The formal and semiformal notations can be used both in implicit or explicit viewpoint models, as explained in chapter 3.5. In the following chapters I will describe the three semiformal notations UML, IDEF and Archimate. The reason for not to look at formal notations such as ACME is that the architecture landscape of Arek is very diverse and it is useful to have some semantic freedom in notation when it is applied in heterogeneous environment.

3.9.1 IDEF

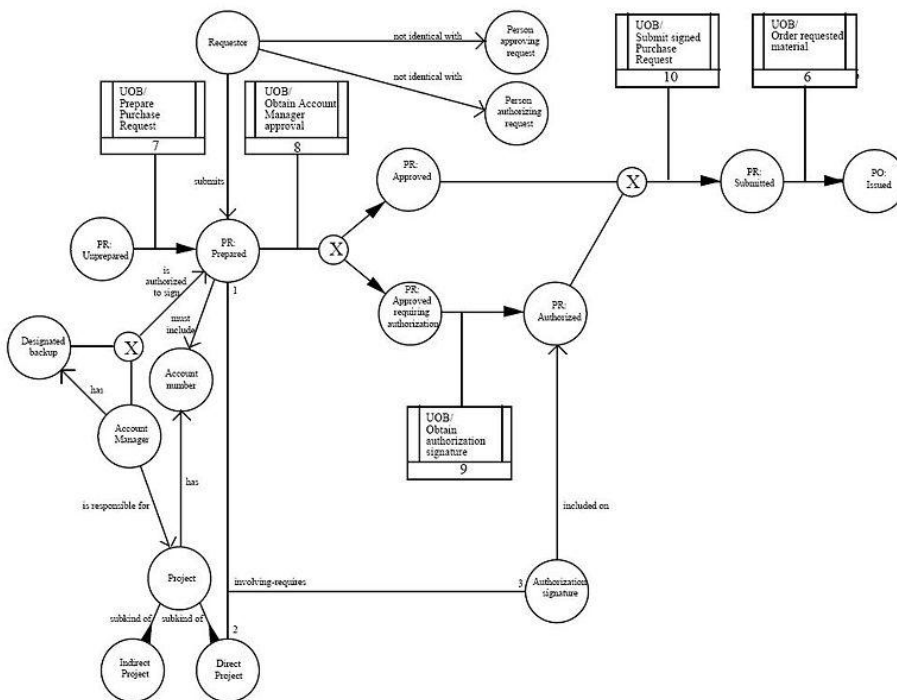
IDEF is a name for a family of languages and methods intended to be used for enterprise modeling and analysis. IDEF comes from words Integrated Computer-Aided Manufacturing (ICAM) **DEF**inition and it has a strong military background. It was developed by the US Air Force Program for ICAM. Currently there are 16 IDEF methods and the most used ones are IDEF0, IDEF3 and IDEF1X. (Lankhorst et al. 2009, 34 - 35.)

IDEF0 is meant for functional modeling. The idea is to model elements that control the execution of the function, the actors performing the function, the data consumed or produced by the function and the relationships between these functions. (Lankhorst et al. 2009, 34.)



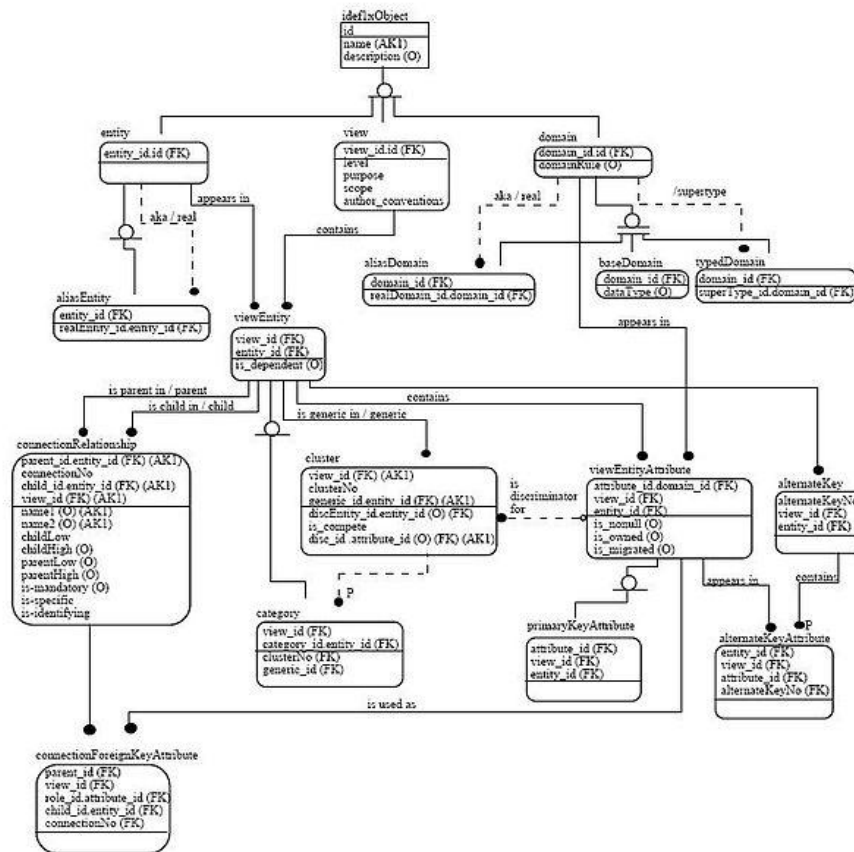
Picture 11: Example of IDEF0 notation (Wikipedia 2012b)

IDEF3 is meant for modeling processes, it has the concept of process flow diagram that shows what tasks are performed and in what order. IDEF3 also shows the logic of decisions in processes and gives methods for analyzing and improving the workflow. (Lankhorst et al. 2009, 34.)



Picture 12: Example of IDEF3 notation (Wikipedia 2012b)

IDEF1X is used to create logical and physical data models. In these models the data entities and concepts are modeled within one are of interest to be used e.g. in database building. (Lankhorst et al. 2009, 34.)



Picture 13: Example of IDEF1X notation (Wikipedia 2012b)

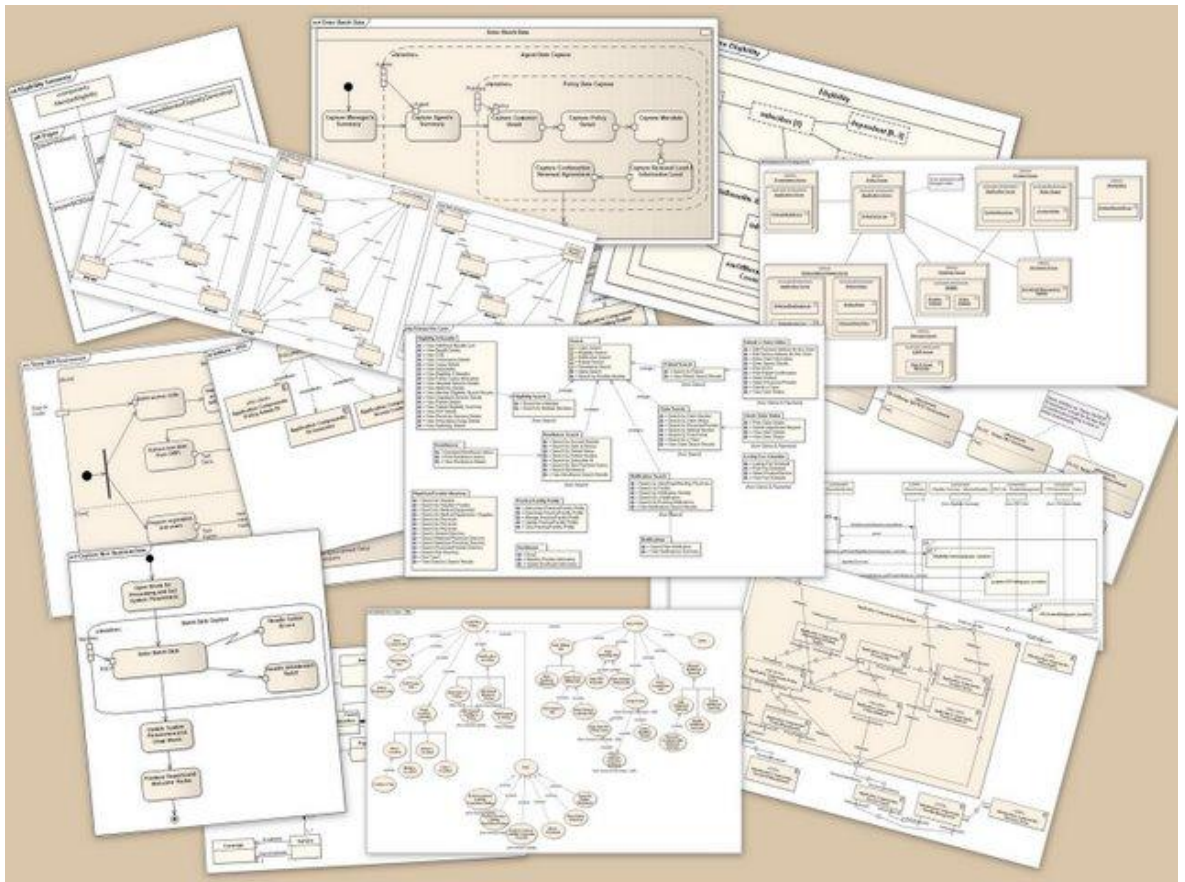
The problem with IDEF is that it gives no advantage over UML or Archimate and the different IDEF models cannot be combined easily, they are mostly isolated. Thus these cannot be use very well for architecture descriptions where the importance is to see how things relate to each other. IDEF is not an option for Arek as there are much more suitable alternatives for notations to be used in architecture definitions.

3.9.2 UML

UML might be the best known notation used in capturing software designs. UML provides many tools, in the form of diagrams and notations, in a single package. UML 2.0 has thirteen diagram types and the notation has been a subject of extensive studies and debate since it was released in 1999. (Dashofy 2007, 25; Lankhorst et al. 2009, 39 - 42.)

Lankhorst claims that UML is mainly intended to be used by system designers and it is a good notation for those who are familiar with computer science. Originally UML was developed for

object-oriented software design but the version 2.0 adds also some features that are useful for enterprise architecture modeling. (Lankhorst et al. 2009, 39.)



Picture 14: Collage of UML diagrams (Wikipedia 2012c)

As a notation UML has a basic element called object. This object can be connected to other objects through a link. Objects can be for example persons or machines. Links can describe any kind of relation between objects. Links can be for example relations or dependencies. All UML diagrams base themselves on objects and links. However the diagrams are sort of sub-languages that use the concept of object and link to enrich the notation. In addition the notation has an Object Constraint Language (OCL) that extends the two concepts by using textual meanings and visual styles. OCL introduces additional concepts such as stereotypes, tagged values and profiles. While there are many diagram types, UML does not, however, have any meta-model or strict separation of diagrams and it lacks standardization by allowing large number of extensions to exist. For this reason the UML is not a formal notation as classified by the Clements et al. (Clements et al. 2011, 53; Lankhorst et al. 2009, 40.)

For the same reason UML should be treated more like a notation than a viewpoint framework like the ones described under chapter 0. Good thing is that UML has a very wide tool support and it is widely used all around the world.

From learning point, Lankhorst et al. (2009, 40) claims that while UML diagrams are rich in notation and expressiveness, the downside is that these diagrams are quite hard to read by new users of the language. This means that UML has a steep learning curve, but when you get used to it, the language is fairly easy to use.

For Arek, UML has been, and will be a important notation. But the notation should be used in designing technical details. When we create architecture descriptions the UML has one bad side effect: it is quite hard to understand and learn by the business stakeholders. For this reason a simpler, yet rich enough, notation should be used in architecture descriptions.

3.9.3 Archimate

Archimate was originally developed in 2005 by the Telematica Instituut at Netherland. Archimate was developed together with representatives from Dutch government, business and academic institutions. In 2008 the Archimate was transferred to the Open Group and in 2009 the Open Group published Archimate 1.0 as a formal standard. (Telematica Instituut 2005; The Open Group 2009.)

The Archimate language is designed to be like an umbrella language of architecture. It is meant to be used with languages such as UML and BPMN in order to give a more abstract description of enterprise and systems architecture and yet let options for detailed design. Lankhorst et al. claim that Archimate is useful as organizations usually use informal notations or detailed design languages like UML that are quite difficult to understand for a non-expert. Use of these kinds of notations leads to misunderstandings that hinder the collaboration between stakeholders and the architects. The promise is that with Archimate one can model any global structure within each domain in an easy to understand way for non-experts of the domain and the relevant relations between different domains. (Lankhorst et al. 2009, 85 - 87.)

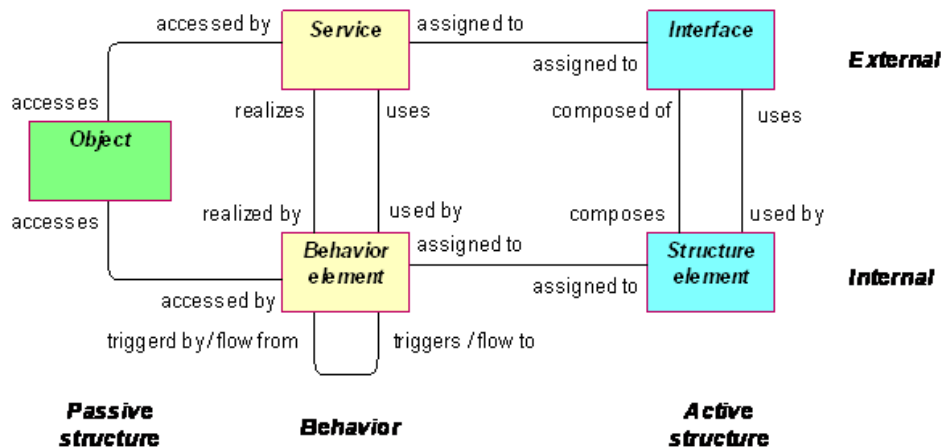
In this thesis, Archimate was used to create a software architecture description. The assumption was that it should be possible and it should also provide compatible inputs to the enterprise architecture work. The other reason for trying Archimate is that service orientation is strongly present in it.

The service orientation shows in Archimate as it presents links between each architecture layer by using services. These three layers are:

- Business layer, which includes products and services to external customers realizes by business processes and performed by business actors or roles.
- Application layer, which supports the business with application services that are realized by application components.
- Technology layer, which offers infrastructure services needed by the applications. Infrastructure services are realized by devices and system software.

(Lankhorst et al. 2009, 88 – 89.)

Archimate presents also three aspects for modeling: active structure, behavior, and passive structure. These aspects follow the natural subject-verb-object elements that human languages have. This element makes Archimate easier to understand for stakeholders. These aspects are then separated to external and internal views. The figure below shows these core concepts of the Archimate that are present in each of its layers. (Lankhorst et al. 2009, 89 – 90.)



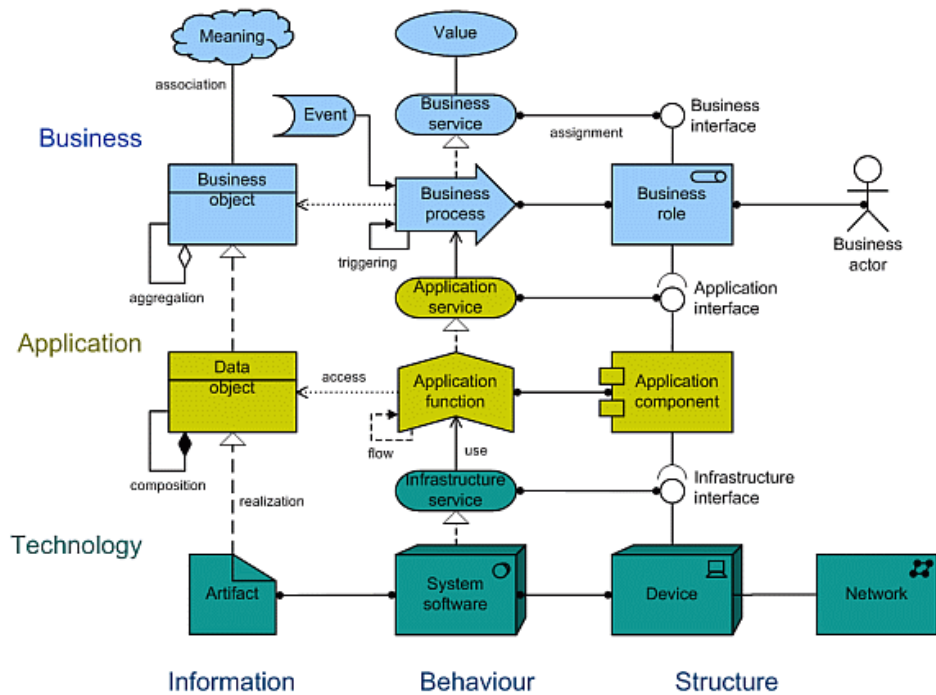
Picture 15: Archimate core concepts (The Open Group 2009).

The core concepts shown in Picture 15 are:

- Service, a unit of essential functionality/behavior exposed to its environment.
- Interface, a unit that presents the external view of structural element and gives access to the behavioral counterpart, Service.
- Internal structures are the realization of Services and Interfaces. These are the Structure element (e.g. actor), that is participating in Behavior element (interaction).
- Objects are used in the interaction by Behavior element that is realized through a Service.

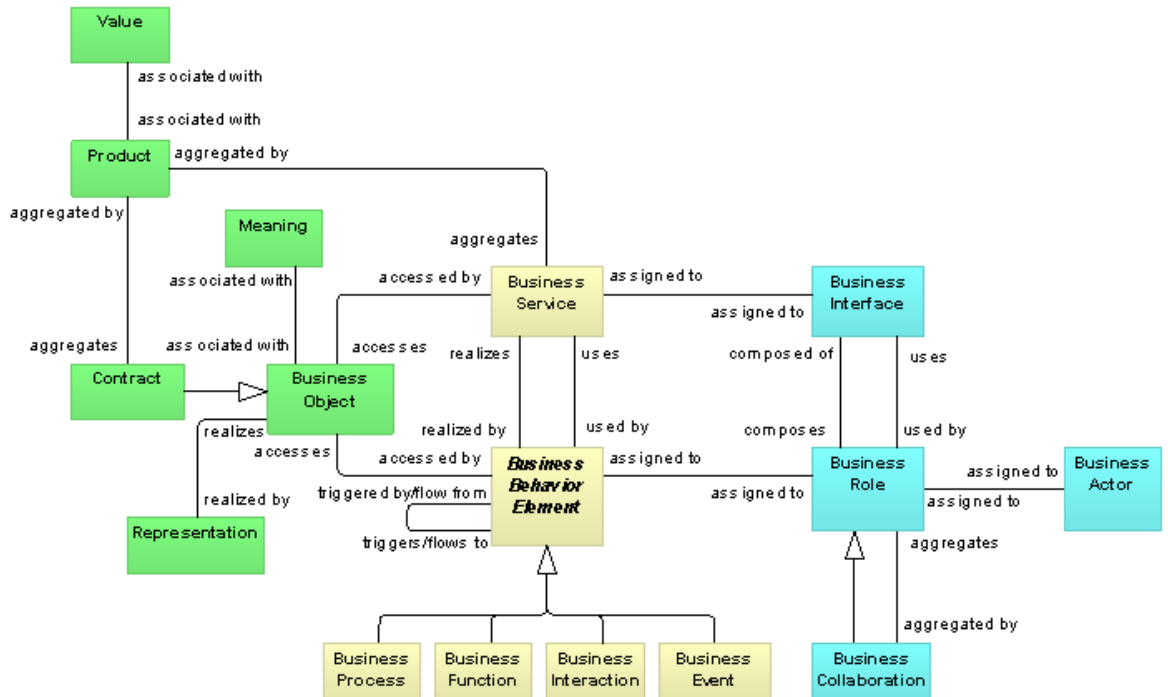
(Lankhorst et al. 2009, 90; The Open Group 2009.)

The synthesis of core concepts and layers forms the most important conceptual elements in Archimate language as shown in Picture 16.



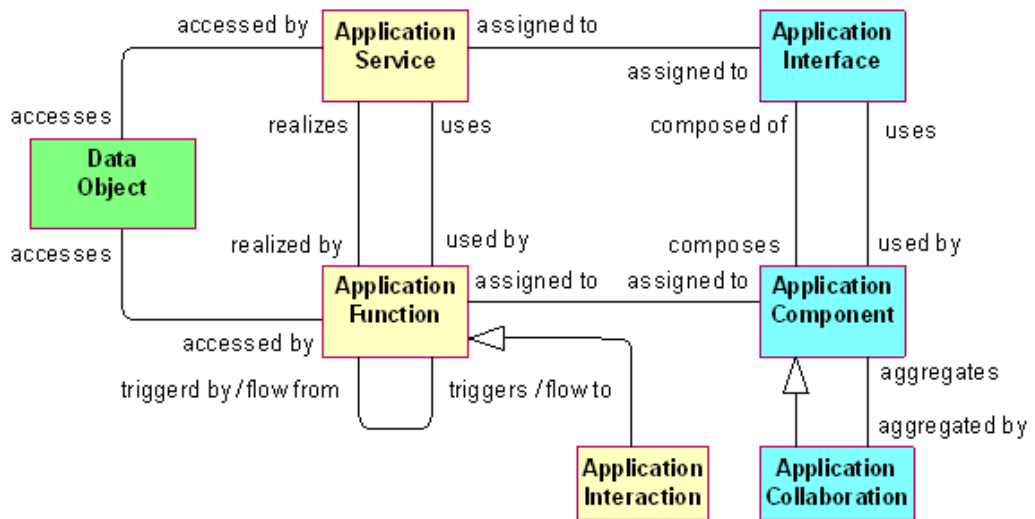
Picture 16: The most important Archimate elements (Archimate 2012a).

When making software architecture the relevance of business layer (Picture 17) is not the highest one. But as the software is supposed to be used by the business it is important to understand what is happening also in the business. In business layer, a business actor is a person or a group of persons, such as business units, that perform work on some kind of business role. A business actor working in some role has also an interface assigned as part of a business service. A typical example might be an insurance clerk (role) in a call center unit (actor) that provides phone services (business service) and clients can call to the center by using a phone (interface). Nowadays, as describes also in section 3.1, an organization is usually running business processes that try to create value for business. Processes start from an event that triggers the need for business process, such as a phone call from a client (event). This call might relate to a need of making an insurance claim (business object) that has some meaning, like information on what the insurance might cover e.g. life and travel insurance.



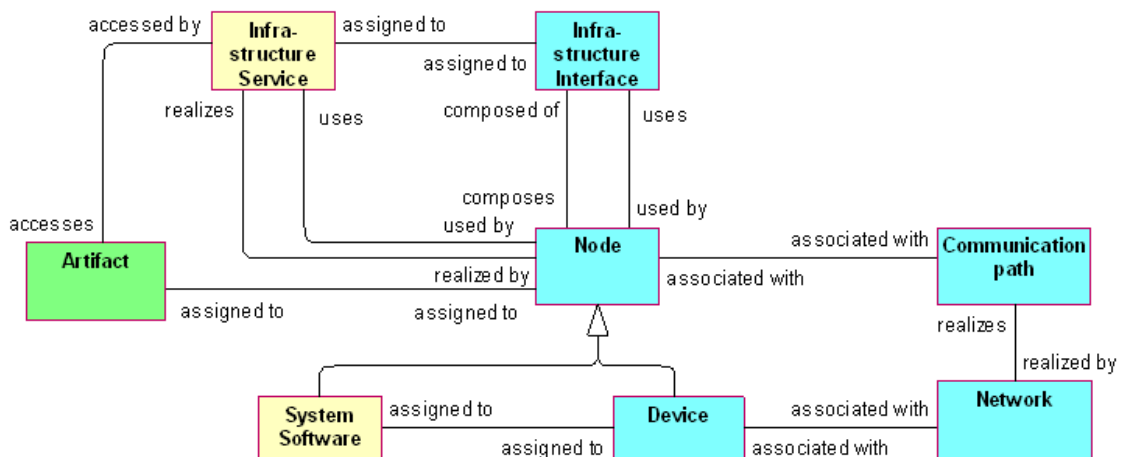
Picture 17: Business layer metamodel (The Open Group 2009).

The application layer (Picture 18) has concepts that are also found in different UML diagrams. Application component is a similar structure as component in UML, it provides an interface that a business role (by using, for example a graphical user interface) or also another application can access (by using, for example a HTTP/SOAP based web service). Through these interfaces an application provides functionality that is realized through a application service. The functions can also be seen as service operations in technical services. Through these service functions a data (data object) is also transmitted, like data of insurance claim that the clerk might store in claim handling system. (The Open Group 2009.)



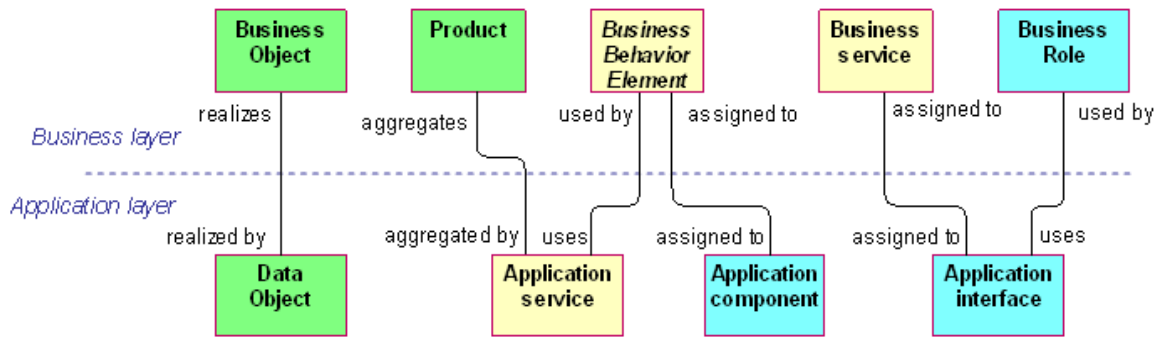
Picture 18: Application layer metamodel (The Open Group 2009).

The technology layer (Picture 19) provides services (infrastructure services), accessed by application components via interfaces (infrastructure interface), that are needed to run the actual business applications. The services are provided by some system software such as database management system or application server software. This software is then deployed on devices, like physical or virtual servers, that are in a network, like local area network inside the company. The data objects used in application layer can be stored in system software as artifacts. An artifact can be for example a file or a database table. (The Open Group 2009.)

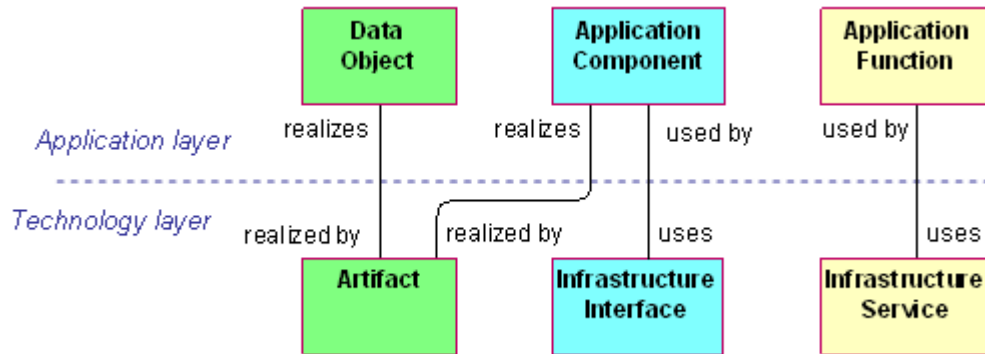


Picture 19: Technology layer metamodel (The Open Group 2009)

In addition to the three layers, Archimate defines how these layers can be combined together. This feature is called Cross-layer dependencies (The Open Group 2009). The allowed combinations are shown in Picture 20 and in Picture 21.

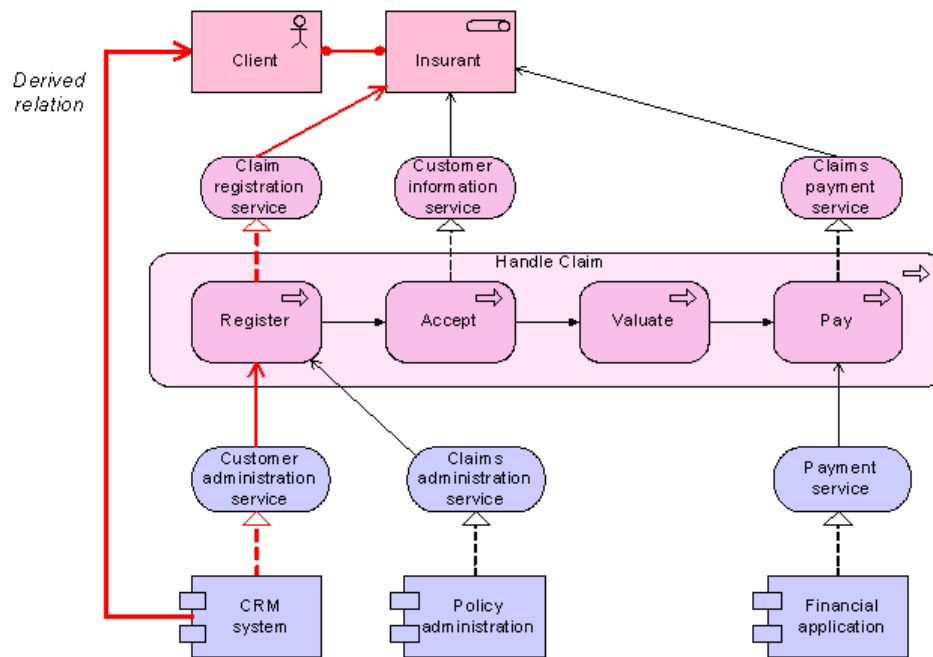


Picture 20: Business - Application relationships (opengroup)



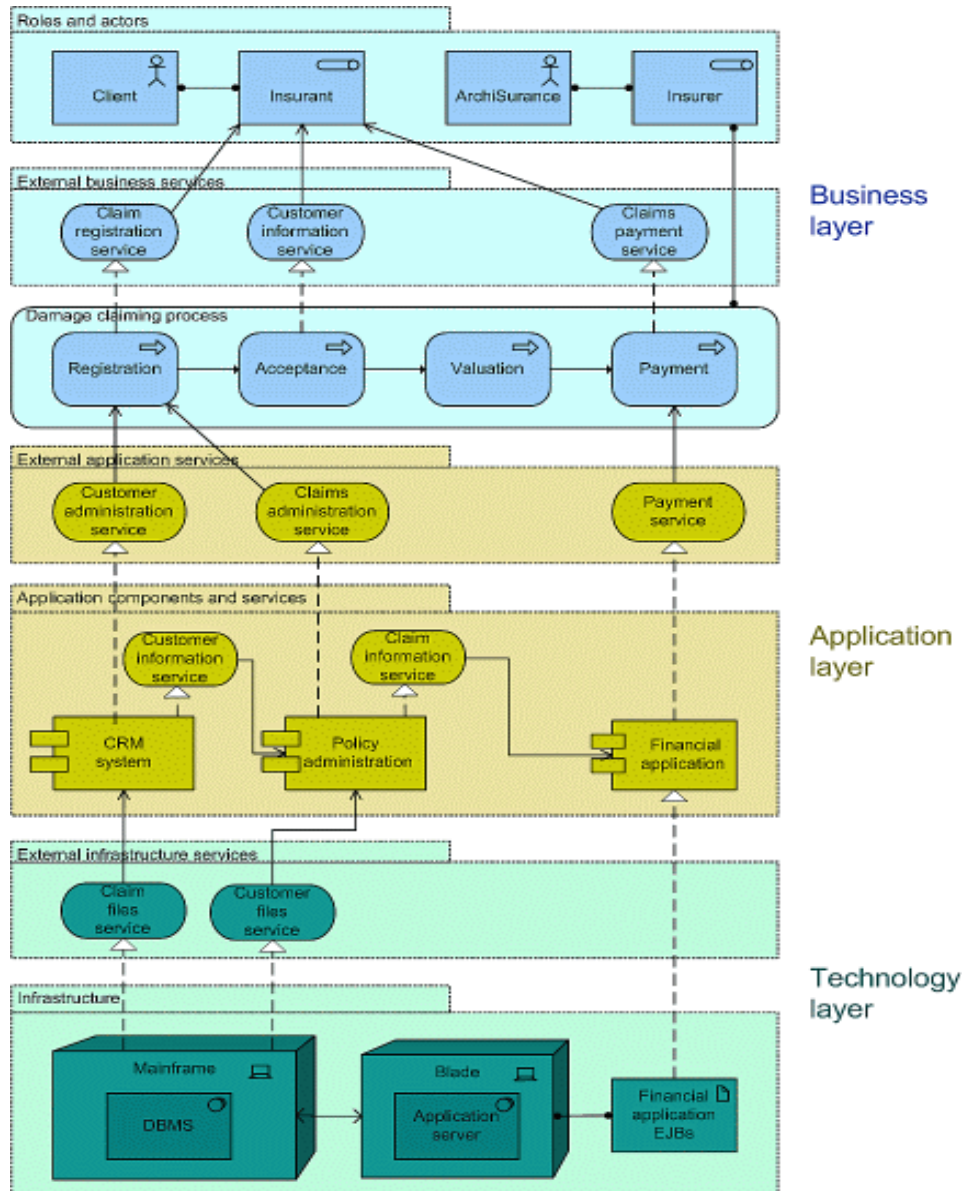
Picture 21: Application - Technology (The Open Group 2009)

Archimate also has twelve different relationships that can be used to connect Archimate concepts to each other. Archimate has seven structural and two dynamic relationships. In addition to structural and dynamic relations there are three other relationships meant to be used in showing grouping, junctions and specializations. (The Open Group 2009.) By using these relationships and the concept of a derived relationship (Picture 22) it's possible to determine the chain of dependencies between architecture layers.



Picture 22: Derived relationships (The Open Group 2009).

When the layers, concepts, relationships and dependencies are combined the Archimate language shows its power (Picture 23). It is possible to create an architecture description that offers the possibility to drill down into architecture from different angles.



Picture 23: Archimate example (Archimate 2012b).

The Archimate does not, in my opinion list as formal ADL as the concepts are more or less vague and leave room for interpretation. This is not a bad thing altogether as discussed earlier, but there were also several other reasons to use Archimate, for example over UML or IDEF:

- The possibility to link the system architecture to enterprise or extended enterprise level architecture seemed very promising.
- The concept of product in business layer metamodel provided a way to support SaaS based business model of Arek (see section 3.3.1).
- Service oriented architecture is a architectural style at Arek and the archimate is strongly utilizing the concept of service.
- The language seemed quite easy to understand even by the business representatives.

For these reasons Archimate was used in this thesis. In addition a Finnish public sector recommendation, JHS179 (JHS-suositukset 2011) states that it is recommended to model enter-

prise architecture by either using Archimate or UML. As some of Arek customers operate in public sector it would be nice to provide architecture descriptions by using the same notation.

3.10 Synthesis of the theories

When an enterprise or systems architecture is modeled within an organization, it is very important to understand the business where the company is involved with. This includes the surrounding business environment and the operated business model. In this theoretic part we went through the business model (SaaS) of the case company (Arek Ltd.) and the current architectural style (SOA) that the company had invested in. After understanding this it was possible to compare the suitability of different architecture frameworks and notations in order to find the most suitable combination to be used in describing systems architecture.

It is also important to know how humans read diagrams. This helps to create more understandable diagrams. When we have made architecture descriptions it is very important to keep them up-to-date. To achieve this it is important to understand how business processes of a company relate to the architectural governance activities performed in daily basis. It is also useful to know how to re-engineer the future state processes in order to keep architecture descriptions up-to-date more effectively.

The actual work done based on these theories is described in the following sections and in attachments 3, 4 and 5.

4 Documenting and managing software architecture in practice

The following chapters describe how the architecture descriptions and architecture governance processes were improved at Arek. These results build a good foundation for the business innovation of the Arek's customers.

4.1 Case Arek Ltd.

Arek Ltd. builds and maintains information systems to earnings related pension insurance sector of Finland. Arek has approximately 40 own employees and around 150 people from different subcontractors, such as IBM, Accenture, Logica and Tieto. The annual turnover in the year 2010 was 67,9 million Euros. The turnover consists mostly from software services that Arek sells based on its Software as a Service business model, consisting of around two hundred SOA based service products. (Arek Ltd. 2012.)

The technological environment in Arek is heterogeneous, consisting from systems created through 4GL, PL/I and Cobol based mainframe systems to distributed J2EE based systems.

4.2 Current state

In the following sub-chapters I describe the current state of Arek's Software Architecture Documentation and its maintenance process it was in the beginning of year 2010. The current state was captured by studying existing Architecture Documentation and interviewing both external and internal stakeholders. At the end of this part I also describe the main problems in the current state while also reflecting my conclusions against the found theory sources.

4.2.1 Software Architecture Documentation

The first task in this study was to browse through all relevant architecture documentation in Arek. The resulting list of documentation formed the necessary material for further analysis.

After browsing the documents it seemed that most of the documentation consisted of architecture descriptions concerning individual systems. A complete view of the whole architecture landscape was only found in a few PowerPoint presentations. The situation screamed for enterprise architecture, so it was very important to choose the framework and notation that could support enterprise architecture discipline in the future. This need influenced decisions in the second research cycle of this thesis.

While analyzing the documentation, it also came clear that there were two significant views missing in architecture documents. The missing views were development and operational architecture. It was interesting that the stakeholders also indicated the importance of these views. And when comparing companies, it seems that some environmental factors in Arek were quite similar with the environment and views used in a telecom company C as described in chapter 3.8. Does the age of a company and/or the vast use of server side software drive the increased need for development and operational views of architecture? This question remains to be answered, but it was clear that at least in Arek these views were needed.

The current state analysis should not be based solely to document analysis as the stakeholders are readers of the documentation. To make the analysis richer, a semi structured interviews were made with the relevant stakeholders (total of 13 persons):

- Arek's customer (3 persons).
- Project manager responsible for development projects (1 person).
- System manager responsible for one or more systems at Arek (1 person).
- Operational staff responsible for Arek production (1 person).
- Architects from Arek architecture team (2 person).
- Support services, e.g. testing, test environments, methodology (3 person).
- Executive board member (1 person).
- Operations management staff responsible for Arek technical infrastructure (1 person).

The interviewees were given a chosen set of questions regarding the understandability and usability of the architecture description (see attachment 1). An existing architecture description of an enhanced and partly modernized mainframe system was also given as an example document to the stakeholders.

After 13th interviewee a saturation point was reached. The comments kept repeating themselves. At this point a decision was made to end the interview part of research cycle. Next step was the result analysis. A total of 88 different comments were given regarding the current architecture documentation. The most recurring comments in the interviews were:

- It should be possible to follow the functional chain of Application Services (published and required) from the architecture description.
- Architecture description is currently missing documented decisions (rationale), why something is made like the way it is.

- Terminology section is missing.
- Each application of the system should be explained separately, this might make application descriptions easier to read.
- The maintenance processes of this document should be designed with care.
- Each section of the document should have its target audience defined.

On the other hand it was clear, that even though the current state descriptions were in the need of some improvement, the stakeholders felt that the documentation was important, quite easy to understand and the overall abstraction level was pretty good.

Interviewees were also given a task to nominate single most important part of the documentation. The results are shown in Table 4.

Table 4: Most important sections of current state architecture description

Stakeholder	Opinion
Customer	The services published and used by the system (including the channel through which the service is used, e.g. http/soap). Description on how these services link to the actual business of the client. Chain of services, as a result of service composition of SOA. Description on how the services are realized by the applications. Description of application architecture and responsibilities of each application.
System managers	The services published and used by the system (including the channel through which the service is used, e.g. http/soap) System overview. Description of the real world surrounded by the system.
Production managers	Overview of the system infrastructure. Execution architecture. System overview.
IT architects	The services published and used by the system (including the channel through which the service is used, e.g. http/soap) System overview.
Support services	Description of application architecture and responsibilities of each application.

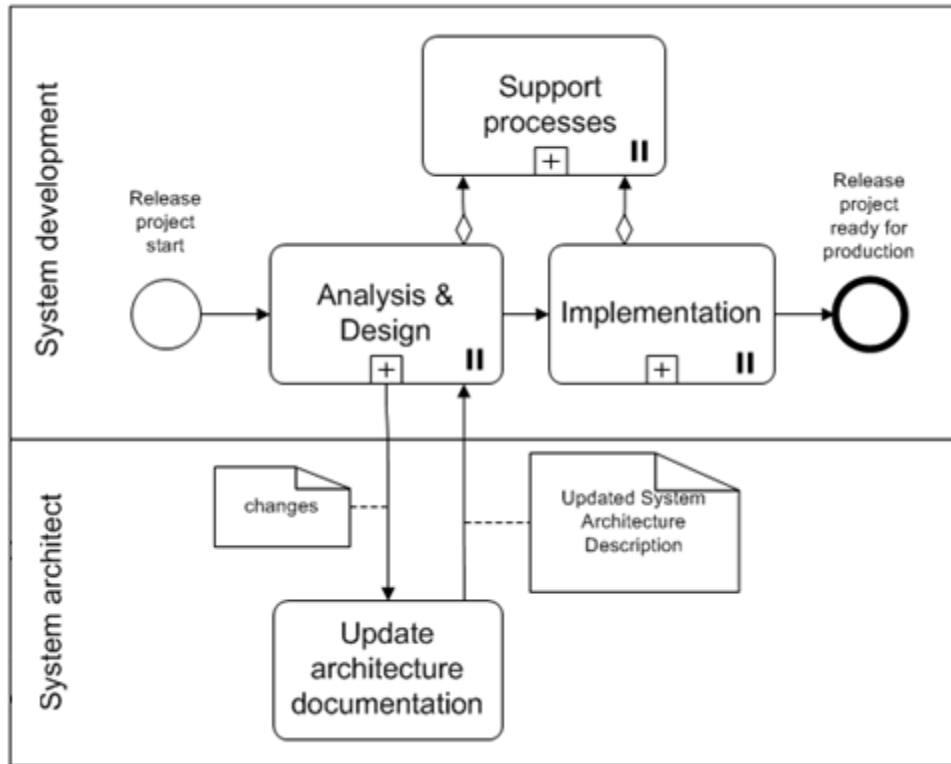
	<p>The services published and used by the system (including the channel through which the service is used, e.g. http/soap)</p> <p>Overview of the system infrastructure.</p> <p>Execution architecture.</p> <p>System overview.</p>
Development projects	System overview.
Executive board member	System overview.
Operational services	<p>Overview of the system infrastructure.</p> <p>Execution architecture.</p> <p>Quality attributes.</p>

The importance of system overview and the concept of service seemed to be important to the stakeholders. Especially the importance of service was an interesting point. Does the SaaS business model play any part in this interest? Or does the interest have anything to do with the service oriented architecture? Especially customers emphasized the importance of service. Probably both SOA and SaaS have something to do with this, after all, the service is usually same as the product on which the customers are willing to pay for.

All in all, the current state analysis revealed many things that should be improved in the future state of architecture descriptions.

4.2.2 Software Architecture Documentation maintenance process

The software architecture documentation process was discovered and drawn (Picture 24) from scratch. This was done by interviewing the architects at Arek and by asking how the maintenance is actually done.



Picture 24: Current state Architecture Documentation maintenance process at Arek

While the overall process seemed to be missing, the update architecture documentation sub-process was actually documented. Based on the interviews, this sub-process seemed to work well, so no major updates on it was necessary.

The biggest concern was that the current process was executed only during projects. The project manager had a lot of responsibility in triggering the actual architecture documentation update. The architect had a very little change to guide the project before it was started. Architects role was mainly to watch after the quality of the system and to update architecture description; if someone was kind enough to tell architect that a new project had started.

The improvement of the maintenance process was an inevitable step in the second research cycle. It missed a lots of needed governance aspects.

4.3 Future state

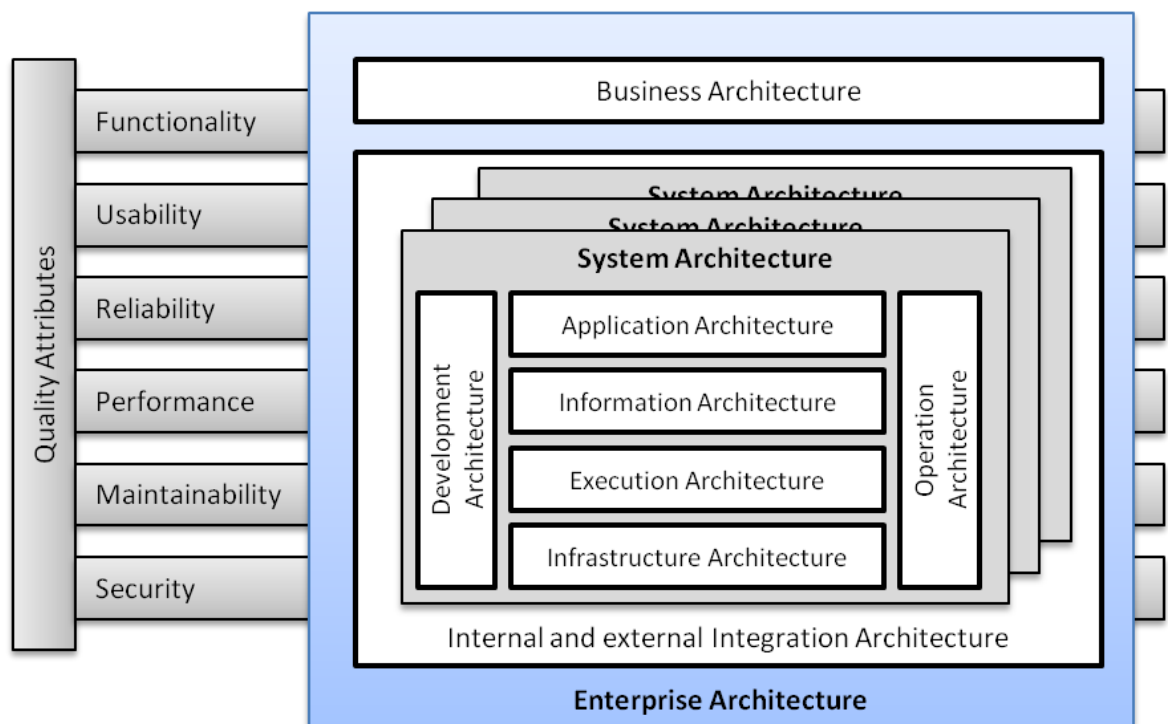
The development of future state begun on further analyzing the content of documents collected on first research cycle. The content was analyzed based on results from stakeholder interviews, internal discussions and theoretic frameworks represented in chapter 0 and its sub-chapters.

Based on the results of content analysis, a suggestion was made concerning the future state of architecture documentation. The researcher decided to see how Archimate notation and metamodel would fulfill the needs in Arek. At least the concepts, abstraction level and support for enterprise architecture modeling seemed to be decent.

To be honest, before reading through the theory and analyzing the results of research cycle, the original idea was to document architecture meta-model by using UML notation. But as Archimate seemed to be suitable notation and metamodel the plan for own metamodel was abandoned. It clearly is a good idea to combine theory with practice.

4.3.1 Software Architecture Documentation

At first, the Archimate viewpoint model and some other useful viewpoint aspects from the theory were adapted to the existing Arek architecture concepts (Picture 25) which were more or less also the interesting viewpoints that should be documented.



Picture 25: Division of Architecture Concepts at Arek

The researcher saw that there were no reasons to change the already learned concepts, but a new concept and viewpoint, Business Architecture was needed. The reason was that the reason of existence of systems was actually never thoroughly documented. Some improvements were made also to the current concepts as their descriptions were clarified.

The used concepts are:

- Enterprise Architecture
 - A documented description of an Enterprise, its goals, business processes required to meet the goals, information systems supporting business processes, technologies and methods used to support development of information systems. EA is the glue between other Architecture concepts.
- Business Architecture
 - A documented description of an Enterprise or Extended Enterprise, its strategy, governance structure, business processes. Business Architecture also describes the business critical information needed in information systems that are used to support the business processes.
- Integration Architecture
 - A documented description of principles, chosen structures and technical solutions used to integrate information systems into each other. Integration architecture is split into external integration, meaning B2B integration (public services) and internal integration meaning integration within an Enterprise (private services).
- System Architecture
 - A documented description of a system embodied in its components, their relations to each other, and to the environment, and the principles guiding its design, development and operation.
- Application Architecture
 - A documented description of components and their responsibilities, services, interfaces and interaction.
- Information Architecture
 - A documented description on how the system stores, modifies, controls and distributes information.
- Execution Architecture

- A documented description of general services and control structures that support the development of applications.
- Infrastructure Architecture
 - A documented description of runtime structures and physical and virtual devices used to run the system.
- Development Architecture
 - A documented description of tools, methods, configuration management and environments required to develop the system.
- Operation Architecture
 - A documented description of operating model used to be followed when the system is in production use.
- Quality Attributes are cross cutting concerns that exists on each level of architecture
 - Functionality is the description of essential functional features that the system must support.
 - Usability is the description on features on how effective, usable and efficient the system is.
 - Availability is the description on how the system available to use on specified times and how the system can recover from errors.
 - Performance is the description on how the system is able to perform the required response and throughput times and how the system capacity can be scaled.
 - Maintainability is the description on how the system is able to be flexible to the inevitable changes.
 - Security is the description on how the system is able to control and monitor the functions performed by the users and how to discover and recover from the problems in information security structures.

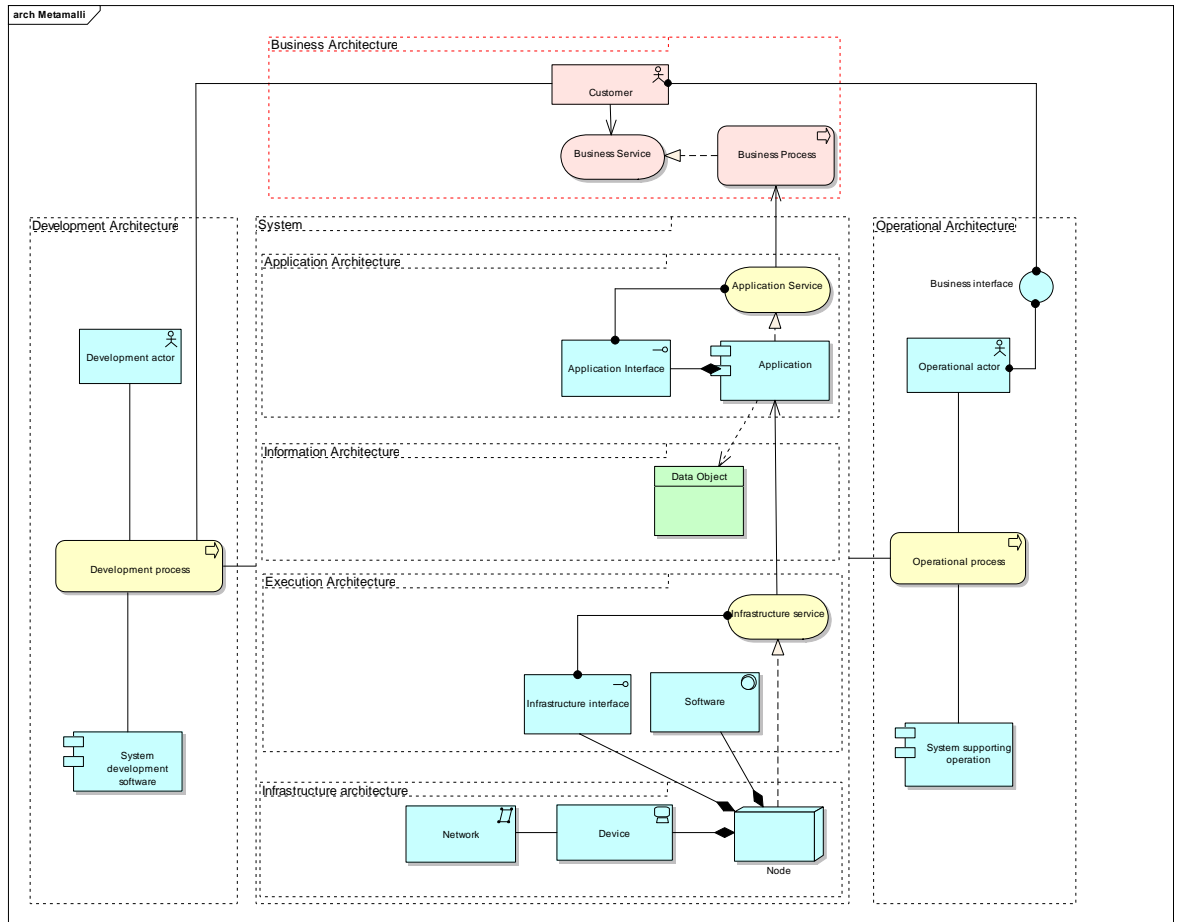
Analysis of the current state, as described in chapter 4.2, revealed that there was a need to discovering functional chain of application services in order to make impact analysis on the system landscape while still keeping the documentation maintainable. This need actually made impossible to continue using the current way of documenting systems by using basic office tools, like MS Word and MS Visio. A metamodel and formal viewpoints were needed to accomplish system architecture descriptions that could be analyzes by using a computer. This is important because the impact analysis, as a very work intensive task, is most effective when it

is at least semi-automated. Maintainability also improves when already created and identified components in architecture descriptions can be re-used.

Archimate was chosen as the metamodel and notation, but why? During the first action research iteration stakeholders told that they do not have strong feelings towards neither the formal or informal notations. As a conclusion a decision was made to go with the semiformal notation that best suits the stakeholders of Arek in order to take advantage of various modeling tools in the market. There seemed to be quite many notations, but the most promising was again the Archimate. It suits well with the Service Oriented Architecture style and SaaS business model used in Arek. And while it lacks some features of a perfect ADL it provided some semantic flexibility that was needed while it was applied against current architecture practice at Arek. The need to support enterprise architecture discipline played also a big role when the decision of using Archimate was made.

There was a need to improve the architecture descriptions with ability to make impact analysis, thus a modeling tool was chosen. Enterprise Architect from Sparx Systems was a cost-effective tool for this project as it provided functionality that didn't exist in open source tool alternatives. Even though a specific tool was used in this thesis, it should be noted that results can be implemented in any tool that supports Archimate and provides ways to re-use modeling components. The ability to do impact analysis and reports is also important when choosing a tool. It should be noted that in actual production, a modeling tool should also fulfill some other requirements, not covered by this thesis, such as effective version control and possibility to connect to external data sources.

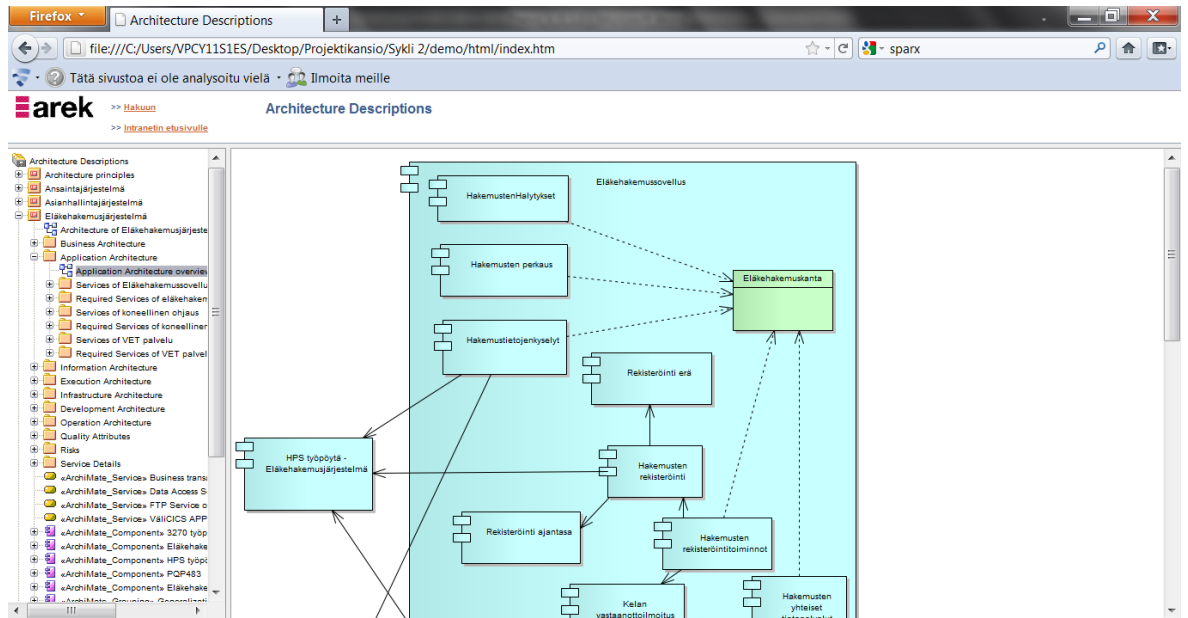
To make impact analysis possible, Archimate framework needed to be mapped with the concepts defined in Picture 25. A high level overview of this mapping is shown in the Picture 26. It was very important to position the core concepts to Architecture layers. Mapping between layers is a crucial task before you can do actual impact analysis.



Picture 26: Mapping of Archimate with Arek Architecture layers

A viewpoint library is a good way to ensure linking between different viewpoints and architecture layers. The idea of viewpoint library is specified in IEEE 1471-2000 standard. The idea of a viewpoint library is to specify the allowed content of each view and to analyze the possible impact of changes that could happen in views.

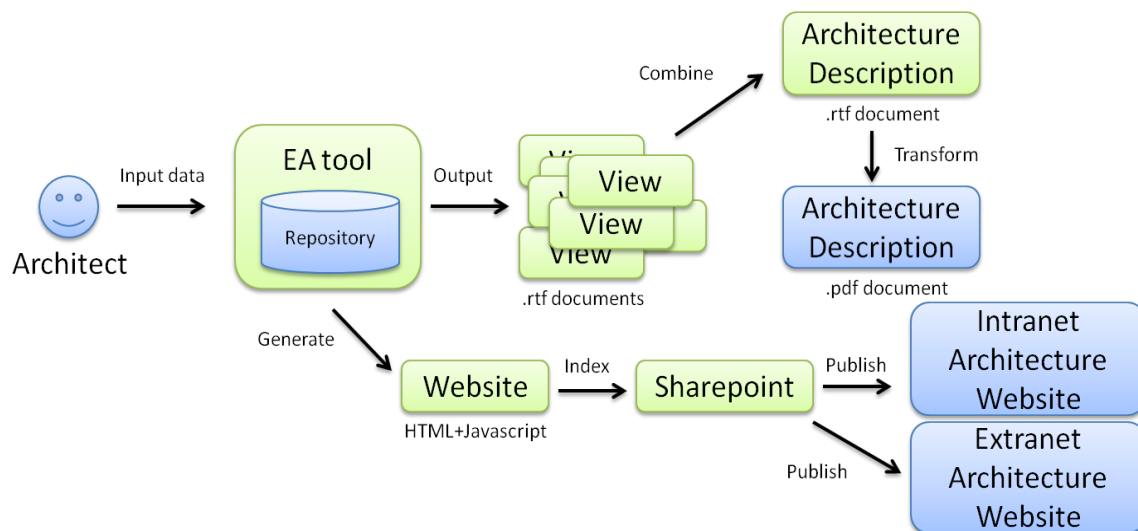
The biggest task was to do document a system by using the chosen tool, notation, metamodel and viewpoints in order to produce a real example made on top of applied theories. After days of modeling, a example architecture description was ready. It was published as a PDF-document and also as a website (Picture 27). This model was then validated with stakeholder interviews and group discussion both in and outside of Arek.



Picture 27: Architecture Description as web site

The used tool provided a web version of architecture description, but to improve the usability, it was integrated with Microsoft SharePoint based intranet of Arek. By doing this, the architecture description had its content searchable by using keywords in the SharePoint search. For example, if you happen to need information about “Eläkehakemuskanta”, a database containing pension claims, you could just write name of database and dive directly to the architecture of the database and the components accessing it. The integration also created the possibility to publish architecture description website to an extranet, from where Arek customers could also access the same architecture content by using their own PC.

While validating the results of the new architecture description, the customers also told that they want to access architecture descriptions from the extranet. In order to minimize the maintenance effort of the descriptions, a central repository should be used. The PDF-versions and web content can then be generated automatically by using the selected tool. Picture 28 shows what publishing functionality was implemented while making this thesis.



Picture 28: Architecture Description multi-channel publishing at Arek

The candidate future way of documenting architecture, was validated with the stakeholders by using same interview technique and almost the same question set (see attachment 2) as in the first iteration. Based on the interviews with different stakeholders the final architecture description had the content shown in Table 5.

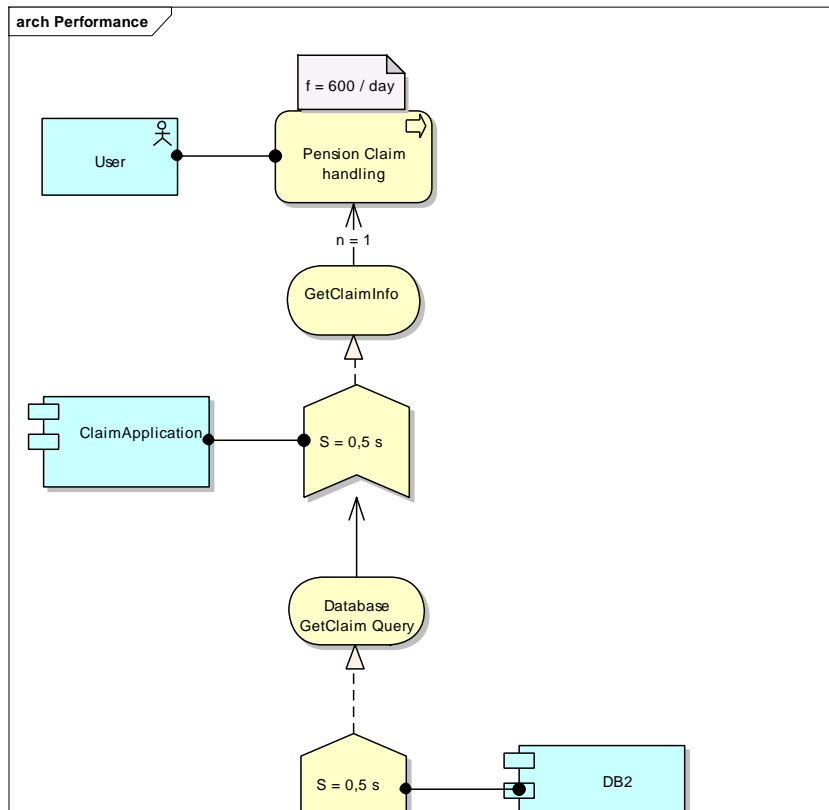
Table 5: Content of the final architecture description

Section	Content
Introduction	<ul style="list-style-type: none"> • Purpose of the document with best-before-date. This best-before date was seen especially important by the stakeholders, by using this date they can be assured that the document is kept up-to-date. If the date has passed they can assume that the document is not trustworthy anymore. • Exclusions, explaining things that aren't included in the document. • Short explanation of used architecture viewpoints. • Architecture principles based on company strategy.
Business Architecture	<ul style="list-style-type: none"> • Purpose and reason of systems existence. • Business processes supported by the system. • History of the system. This was seen as very important content in the architecture description. By understanding the history of system you can try to understand the solutions made in the system. The case system used in this modeling exercise had its oldest parts coded in the beginning of 1990's, so the system was almost twenty years old in the time of writing this thesis. • System users. • Technical overview of the system for top management. • Products offered by the system. This part is directly related to SaaS business model described in chapter 3.3. Each relevant public service offered by the systems has a associated product with it, e.g. service for retrieving pension claim status of a person has a product code EHJ0001. This code tells a lot to SaaS customers of Arek as the prod-

	uct has pay-per-use charge.
Application Architecture	<ul style="list-style-type: none"> • Overview of application architecture. This section describes how the system is divided into components and how these components are connected with each other. System is further split into applications (sub systems) that provide services, but the overview describes the high level structure of the system. • Published services. This part describes, by sub-application level on which services do they publish to other systems or system users. The interviewees made clear that it's very important to link some detailed information to each service, like transaction ID, service ID and the name of program realizing the service. This enables stakeholders to find the services very fast, e.g. by using the search function of intranet. • Interfaces were also seen as very important information by the stakeholders. As the service can be provided through one or more interfaces (e.g. soap interface, batch interface or web-browser) it's crucial for system integration to know how the service can be accessed. • Required services. This section describes, by sub-application level on which services the system uses from other systems.
Information Architecture	<ul style="list-style-type: none"> • Overview, explaining the ownership of the data, data owned by the system and required from the other systems. The most important section seemed to be the description of concepts used in the system (conceptual model).
Execution Architecture	<ul style="list-style-type: none"> • Overview, explaining the common services utilized by the system. These services can be provided by middleware or self-made components, deployed on logical nodes. • Detailed information from each part, e.g. how the batch handling is implemented on FTP server and what artifacts need to be customized in order to make a new batch run working.
Infrastructure Architecture	<ul style="list-style-type: none"> • Overview, showing how the logical nodes are divided into physical nodes. Here it was very important to use the actual names of physical nodes, e.g. the actual name from which the operation staff really identifies the Windows server located in Helsinki, apart from the one that resides in Vantaa. • Infrastructure network, showing the firewalls, routers, VLANs and network capacity used between physical nodes. • Infrastructure communication, showing how the logical nodes should be connected to each other (ports used) in order for the system to work properly.
Development Architecture	<ul style="list-style-type: none"> • Overview, explaining the used development methodology, key roles, tools (e.g. version control and build management systems), programming languages and test environments.
Operation Architecture	<ul style="list-style-type: none"> • Overview, showing the key service support processes, roles and interfaces (like phone and email) how customers can contact operational teams, tools (for example ticketing tool) and support systems (such as example backup system). • End-to-end monitors, showing what SaaS services are monitored by which monitoring transactions. This section was important to operation staff, this way they can deep dive into architecture structure when there is an error in the system. They can also communicate more easily to customers with business terms as they can map e.g. E2E_ETEAHKY

	<p>transaction to a service of retrieving pension claim information. The operating staff also felt that this helped them in the making of impact analysis. In a hypothetical example one might inform customers that: “Arek has detected a system failure and the pension claim information cannot be retrieved currently at the pension companies. For this reason the Claim Delivery process is disrupted in the offices around Finland. We are currently working on a solution to get the services up and running within thirty minutes.”</p>
Quality attributes	<ul style="list-style-type: none"> • Quality attributes describe the known quality factors that relate to the system, these are for example security, performance, usability and availability.
Risks	<ul style="list-style-type: none"> • Overview of the future risks the system might come up with. These are mostly strategic level risks that must be taken care of. This part is important e.g. due to the longevity requirements of the Arek systems, the systems must be usable by the business for tens of years.
Viewpoint library	<ul style="list-style-type: none"> • Each viewpoint is explained in the library, e.g. what concerns do they address and what symbols are used in diagrams.
Glossary	<ul style="list-style-type: none"> • Explains the terms used in the documentation.
Compliance check	<ul style="list-style-type: none"> • Compliance check with IEEE 1471-2000, having a list of items that are checked in order to make the document compliant with the standard.
Service details	<ul style="list-style-type: none"> • For many stakeholders, high level pictures are rarely enough. A service details view concentrates with more detail to the Service. View answers for example to questions on through which interface the service is delivered and what kind of software is needed in order for the service to function. • This view was seen important only in the web version of the architecture description. The reason was that only through web version you could move horizontally and vertically through each of the architecture viewpoints and layers of the information system.

The architecture description also makes different kind of analysis possible. For example it is possible to analyze how the system can respond to performance requirements. By adding some parameters (Picture 29) to the model it's possible to calculate things such as maximum throughput the system can provide to business users within a day. The same kind of analysis can also be made to other quality attributes: controls can be attached to model in order to validate information security and reliability percentages can be used in order to support the design high availability configurations.



Picture 29: Quality analysis

From Arek perspective the interesting possibility lies in the ability to pre-calculate the cost of building new services to an existing system or to provide analytical support for redesigning production deployment process in order to minimize planned system downtimes.

As the architecture descriptions are available also to Arek customers, they can anticipate and optimize more easily the cost and performance of their business processes. This is based on the fact that the Arek services have a price tag and performance information.

All in all the stakeholders were pleased from the improvement of the architecture descriptions. The most important stakeholder comments are summarized in section 4.4.1.

4.3.2 Software Architecture Description maintenance process

The architecture descriptions need to be up-to-date as explained in the previous sections of this thesis. For this reason there must be processes in place in order to assure that the descriptions are up-to-date. In the beginning it was decided that the processes should be modeled with industry standard. A BPMN notation was chosen as it is a widely used notation in business process modeling. The notation is described in chapter 3.1. The actual processes are documented in attachment 1 and 2 of this thesis.

The future state processes were made together with the actual participants of the processes. As there were some linking processes, such as system change management, the process owners of those processes were also involved in this task. The improvement job was done by applying BPTrends method.

In the end, the following processes were described (see attachment 3)

- Architecture Change Management Process.
- Feasibility Study Support Process.
- Verification and Audit Process.

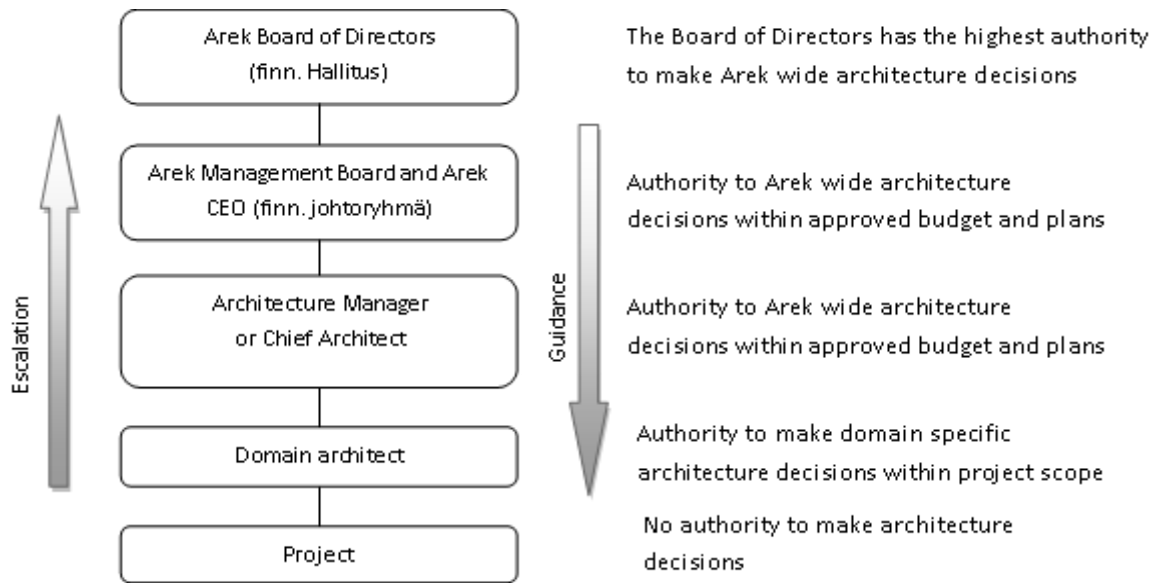
For these processes, three guidelines were written (see attachment 3):

- Decision making model.
- Determining project impact.
- Performing enterprise architecture compliance reviews.

In addition to the processes and guidelines, the already existing process of making and updating system architecture documentation was slightly modified (attachment 4) as it already existed in the current architecture practice (Picture 24).

The guidelines were important in order to provide more detailed instructions to participants of the architecture processes. One of the most important guideline was the decision making model. It was designed to be executed by five levels of hierarchy (Picture 30). The highest authority is in the Arek Board of Directors, which consists of representatives of company owners. This was the way to ensure that the biggest decisions are always business oriented. The lowest level of hierarchy is the project –level which has no authority make decisions regarding architecture. All lower level architecture related decisions are made by the domain

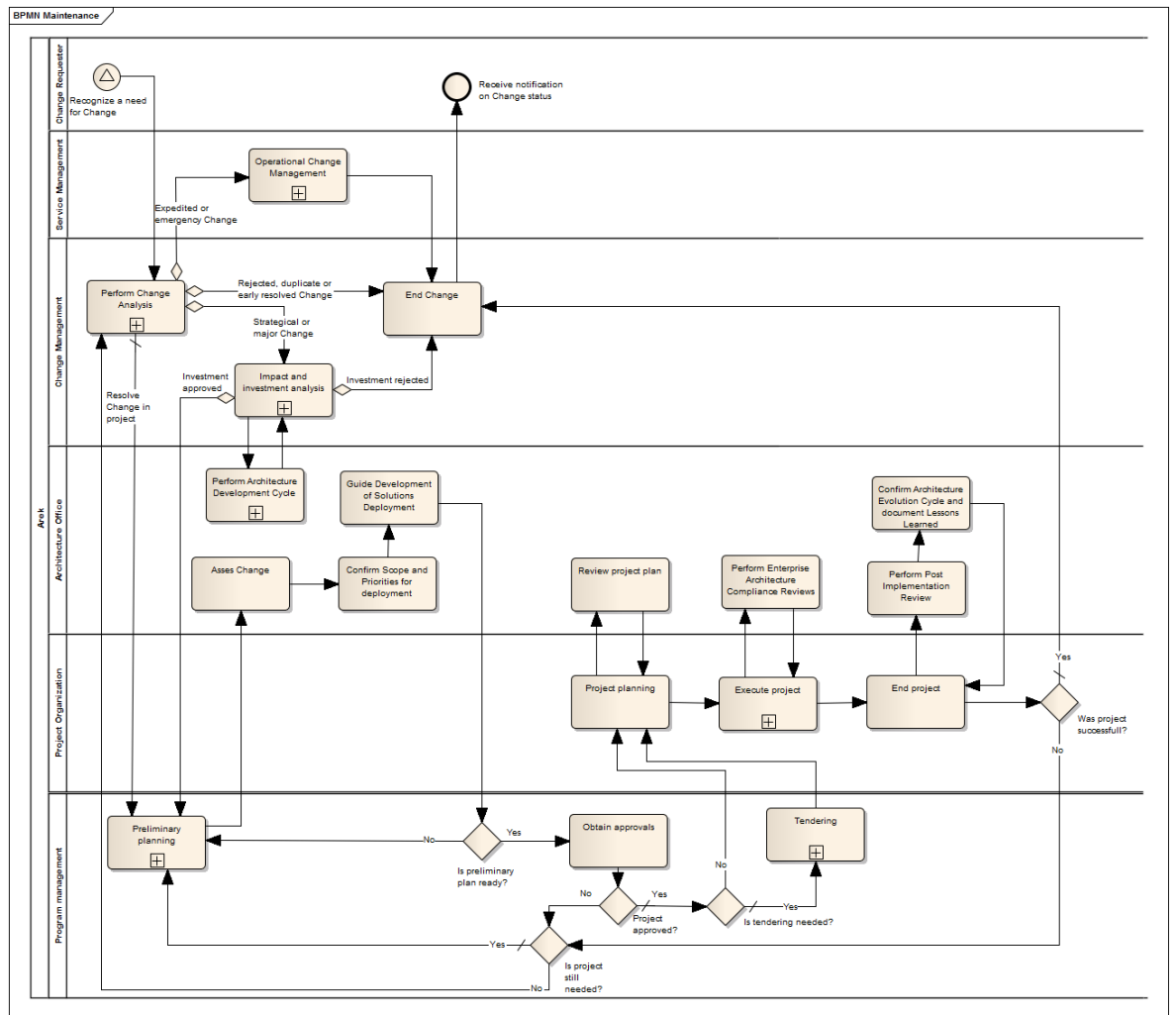
architect. He or she can escalate decision making if his/her own authority to the issue at hand is not big enough.



Picture 30: Decision making hierarchy

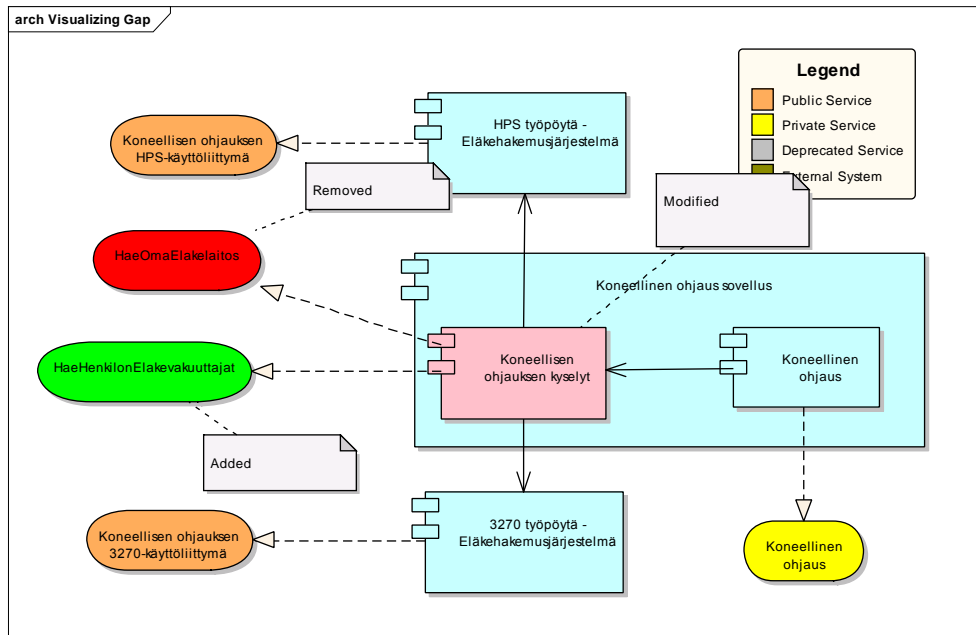
The decision making model is complemented by the guideline of determining project impact to each project. Basically the guideline is based on assessing the level of architecture impact the project is going to create and the cost of executing the project. Highly impacting and costly projects must be watched more closely and these projects must be given reinforced architecture support. Low cost and low impact projects only need minimal architecture support as the update of architecture description baseline is usually enough. When the project is executing, architects make compliance reviews to the work products of projects. This is to assure that the project is following the designed architecture. All this is part of the actual core process of architecture management, the process of Architecture Change Management (Picture 31).

The process is integrated to processes of feasibility study, impact analysis, investment management, change management, program management and tendering. Integration to other processes is crucially important, architecture is not something you do in ivory tower. Architecture is part of everyday life, part of the key processes in organization. In this process the wheel has not been invented again, thus there are applied theories behind it. The actual architecture documentation management part is based on the centralized pattern as referred in chapter 3.2. The tasks performed by architects before, during and after a project are based on steps within TOGAF9 ADM-cycle.



Picture 31: Architecture Change Management

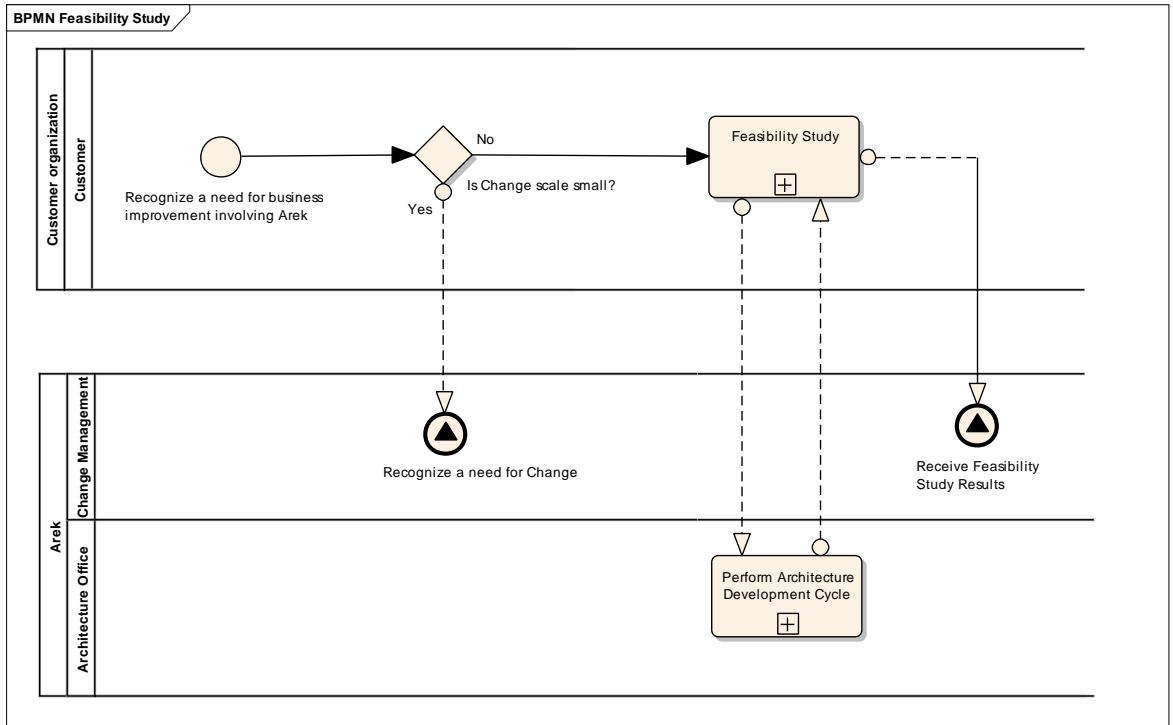
The possibility to make impact analysis (to define "delta" in the change) in architecture development (see Picture 4 in chapter 3.2) is very important. The visualization of the gap caused by transition from current architecture to target architecture is very important in architecture change management process. A proposed visualization used in this thesis is shown Picture 32. The modified parts of architecture are shown in pink, removed in red and added in green.



Picture 32: Visualizing the Gap

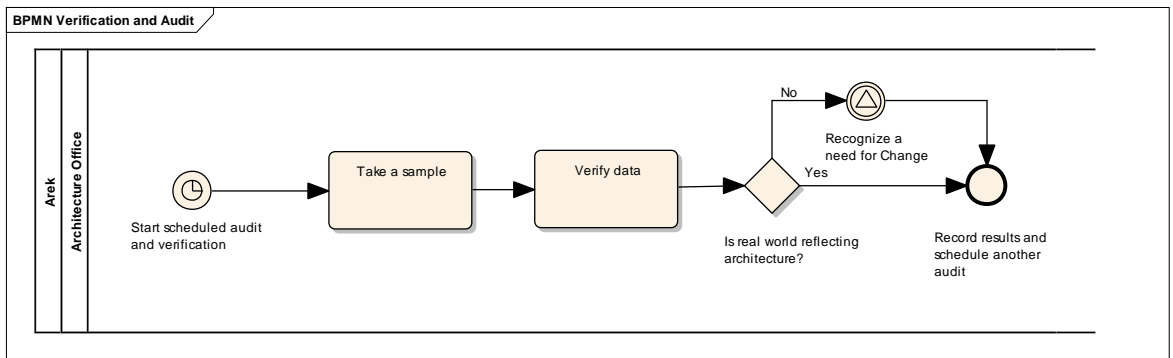
Archimate metamodel and the tool support make the impact analysis possible. It is very important feature of any architecture tool to discover dependencies between e.g. components and services.

Before any change management can happen a change is needed. The feasibility study process helps customer to discover new business opportunities. This process ensures that small changes can be implemented quickly while big changes are evaluated through a TOGAF ADM cycle.



Picture 33: Feasibility study process

On the other hand, the verification and audit process ensures that the architecture compliance can be checked by the architect. This is done by scheduled audit and verification timetable in which architect chooses architectures to be evaluated and verified. Any compliance problems that need fixing are then raised as changes to the change management process.



Picture 34: Verification and audit process

The more detailed processes with descriptions of each step can be found from attachment 3.

4.4 Validation of future state

In the following sections I will describe how the project results were validated.

4.4.1 AD validation

Architecture descriptions were validated by the stakeholders by conducting personal semi-structured interviews. Almost the same question set was used to validate the suggested way to document architecture (see attachment 2) as it was while gathering information in the first research cycle (see attachment 1). The hypothesis that the suggested architecture description would be better than the original one was correct.

A total of 12 persons were interviewed from the following stakeholder groups:

- Arek's customer (3 persons).
- Project manager responsible for development projects (1 person).
- System manager responsible for one or more systems at Arek (1 person).
- Operational staff responsible for Arek production (1 person).
- Architects from Arek architecture team (3 person).
- Support services, ie. testing, test environments, methodology (1 person).
- Executive board member (1 person).
- Operations management staff responsible for Arek technical infrastructure (1 person).

During the interviews all interviewed stakeholders indicated that the architecture description was significantly better than it was at the beginning of the project. However, a saturation point was not reached during the interviews. Interviewees pointed out small details, that needed fixing, but no major gaps were present in the architecture description. As most of the interviewees took part to interviews also in the first research cycle it was enough to make an assumption that while the description was better than before, the smaller detail-oriented problems gained more attention in the interviews. A total number of 63 different comments were caught regarding the suggested architecture description. All interviewees stated that the Archimate notation is far better than the one used before. They also stated that the architecture description is a very important document.

The most important positive points were:

- "I would definitely use the Architecture website over paper document" (12/12).
- There is no pointless information in Architecture Description (12/12).
- Models are great and the use of colors is very good (7/12).

- I would use the paper document in some situation (7/12).
- "We have definitely needed Architecture Descriptions like this one" (4/12).

The most important points for improvement were:

- Service Details should be shown only in Architecture website (5/12).
- A reference to change management practice would be a nice addition (4/12).
- ServiceID should be shown in service descriptions (4/12).
- System life cycle information should be written (3/12).
- "I'd like to have some description of test environments" (2/12).

The stakeholders hoped also some minor additions to the documentation that were beneficial to them, such as showing the serviceID information associated with the service. A viewpoint for products and end-to-end monitors was also in the wishing list. As these wishes were easy to fulfill they were implemented during this project.

The question on what is important information in architecture description was repeated during the interviews. All stakeholders groups gave the same answer as they did in the first interview round. This strengthened the belief that the architecture description has all the necessary viewpoints. And again, all stakeholders indicated that there is no useless information in the architecture description.

When the architecture description was nearly completed, the joint architecture group of pension insurance sector showed interest towards this thesis. A discussion was taken with the architects of the extended enterprise context. Based on this discussion some minor modifications were made to the description to support the linkage not only to the enterprise architecture, but also to the architecture of the extended enterprise.

4.4.2 Process validation

The future state processes were validated in workshops, together with the stakeholders. The workshop results were documented and the processes were updated according to the observations. It should be noted that the actual process validation can only be done in the practice. As the actual process roll-out to the organization was out of the scope of this project the usability of processes is based mostly to the stakeholder expertise.

5 Results discussion

From Arek perspective, the thesis was helpful. Based on the studies made during thesis project, Arek decided to start a project to implement the designed governance processes and to document systems architecture with the Archimate notation and viewpoints represented earlier in this thesis. Arek estimates that the new way of documenting architecture and the governance processes will provide significant benefits to the governance of software services.

The research question was also answered, at least in the context of this case. The Archimate notation, together with stakeholder driven viewpoints made it possible to find a way to describe architecture in reasonable way, at Arek. The utilized SaaS business model and service oriented architecture style made a perfect match with the chosen notation and created viewpoint framework. Attachment 5 is an example on how to document service oriented systems architecture. Attachments 3 and 4 describe the maintenance aspects of architecture description. The understandability of the documentation is ensured by the taken interviews and workshops with the stakeholders. The hypothesis that the suggested architecture description would be better than the original one was also correct.

The theories on how to document architecture helped to form a easy to understand architecture description. And the governance processes ensured that the descriptions are trustworthy. The resulting processes and the created documentation practice is also independent from the implementation technology. This makes it possible to document both legacy and modern systems in the same manner and abstraction level.

The action research methodology ensured that the thesis was very practical. The scientific methods and practice walked hand in hand, despite of the known problems related to action research. But at least for this thesis, the method was helpful and it also had similarities with TOGAF ADM, a method for architecture development.

5.1 Contribution to the community and further research

The most important contribution was the discovery of suitable way to document SOA based systems in a company running Software as a Service business model. The result had only little case-specific solutions. It would be interesting to see if the created architecture description model would work in other SaaS based companies.

One discovery was also that there are ready viewpoint frameworks but none of them were completely suitable for Arek. Could it be that the business model and environment of a company mostly affects the viewpoints? There seems to be some indications to support this, the research made by Smolander et al. and the observation from this thesis show some resemblance between companies business models and the architecture viewpoints used in those companies. A further research might be useful about the affect of business model to architecture descriptions.

This thesis also gives some tips on how to document architecture: the use of colors and number of elements for example. A study and even a guide, might be made for architects to help in visualizing complex systems, regardless of the used notation.

As explained in this thesis, the Architecture also makes it possible to execute quality analysis against the modeled solution. One interesting use case might also be the service pricing calculation. The SaaS product prices could be calculated partly based on the architecture descriptions and gap analysis results. It might be useful for Arek to speed up the process of counting product prices by utilizing e.g. cost (building and maintenance costs) as an attribute of an architecture element, whether it is a internal component of a system or a provided service.

A deeper analysis to architecture description usage might also be a good subject for further studies. If the description is published in a web page, as suggested by this thesis, it is possible to track what are the most used viewpoints and search phrases used in finding information. As this thesis covered about a dozen of interviewees the web page analytics could easily cover hundreds of users from Arek's stakeholders.

The results of this thesis can also be linked to the extended enterprise context of the earnings related pension sector in Finland. A study on what would be the best ways to link cross-organizational architectures might be a big benefit for the whole sector. During this project a discussion was also taken with the joint Architecture group of earnings related pension sector.

The group showed great interest in project results, the documentation practice with Archimate seemed to work quite well for the benefit of whole pension industry.

As said, this thesis forms a good foundation for Arek and its customers to further develop systems architecture descriptions and allows their linkage to the enterprise architecture context. As the information in architecture descriptions is easy to understand, standardized, maintained and available to all stakeholders, it's possible to create new ideas on how to harvest additional business value based on the results of this thesis.

"in scientia opportunitas", in knowledge, there is opportunity.

Bibliography

Archimate 2012a. What is ArchiMate? Accessed online on 6.3.2012 from http://www.archimate.nl/en/about_archimate/what_is_archimate.html.

Archimate 2012b. Example. Accessed online on 6.3.2012 from http://www.archimate.nl/en/about_archimate/example.html.

Arek Ltd 2012. web-site. Accessed online on 6.3.2012 from <http://www.arek.fi>.

Capilla R., Nava F., Dueñas J. 2007. Modeling and Documenting the Evolution of Architectural Design Decisions. Proceedings of the Second Workshop on SHARing and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, pp. 9 -15.

Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Merson P., Nord R., Stafford J. 2011. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley. Boston. USA.

Dashofy, 2007. Supporting Stakeholder-driven, Multi-view Software Architecture Modelling. University of California, Irvine.

DeRemer, F. and Kron, H.H. 1976. Programming-in-the-Large versus Programming-in-the-Small. IEEE Transactions on Software Engineering. 2(2), pp. 80-86, June, 1976.

Harmon, P. 2007. Business Process Change: A Guide for Business Managers and BPM and Six Sigma Professionals, Second Edition. Morgan Kaufmann Publishers. United States of America.

Heikkinen, H. LT. Rovio, E., Syrjälä, L (toim.). 2007. Toiminnasta tietoon, toimintatutkimuksen menetelmät ja lähestymistavat. 2nd edition. Kansanvalistusseura. Finland.

Hohc, F. et al. 2001. Software as a Service: "A to Z" for ISVs. Software & Information Industry Association (SIIA), Washington, D.C.

Horton, W. 1991. Illustrating Computer Documentation. Wiley, New York.

Hyvärinen, M. 2006. Kerronnallinen tutkimus. Finland. Accessed online on 21.02.2012 from http://www.hyvarinen.info/material/Hyvarinen-Kerronnallinen_tutkimus.pdf.

IEEE, 2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000.

ISO, 1996, ISO/IEC 10746 Reference Model of Open Distributed Processing.

ISO, 1998, ISO/IEC 10746-1 Information technology - Open Distributed Processing - Reference model: Overview. Accessed online on 2.3.2012 from [http://standards.iso.org/ittf/PubliclyAvailableStandards/c020696_ISO_IEC_10746-1_1998\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c020696_ISO_IEC_10746-1_1998(E).zip).

itSMF 2007. An Introductory Overview of ITIL® V3. Accessed online on 6.3.2012 from http://www.itsmf.org/files/itSMF_ITILV3_Intro_Overview_0.pdf.

JHS-suositukset 2011. JHS 179 ICT-palvelujen kehittäminen: Kokonaisarkkitehtuurin kehittäminen. Accessed online on 6.3.2012 from <http://www.jhs-suositukset.fi/suomi/jhs179>.

Kananen, J. 2009, Toimintatutkimus yritysten kehittämisessä. Jyväskylän Ammattikorkeakoulu. Jyväskylä. Finland.

Kuula, A. 1999. Toimintatutkimus – Kenttätöitä ja muutospyrkimyksiä. Vastapaino. Tampere. Finland.

Kuusela, P. 2005. Realistinen toimintatutkimus. Työturvallisuuskeskus. Helsinki. Finland.

Koning, H. Dorman, C. van Vliet, H. 2002. Practical Guidelines for the Readability of IT-architecture Diagrams. Accessed online on 2.3.2012 from <http://www.few.vu.nl/~hans/publications/y2002/SIGDOC2002.pdf>.

Kruchten, P. B., 1995. The 4+1 View Model of Architecture. IEEE Software, vol. 12, no. 6, pp. 42 – 45. Accessed online on 03.03.2012 from <http://www.cs.umb.edu/~jxs/courses/2007/680/papers/kruchten.pdf>.

- Laamanen K. 2003. Johda liiketoimintaa prosessien verkkona – Ideasta käytäntöön. Laatu-keskus. Helsinki. Finland.
- Laitinen, M. 1999. Kertovan muutosselonteon menetelmä. Aikuiskasvatus 3/99. Finland.
- Lankhorst, M. et al. 2009. Enterprise Architecture at Work: modelling, communication, and analysis. Springer. Berlin. Germany.
- Lassila 2006. Taking a Service-Oriented Perspective on Software Business: How to Move from Product Business to Online Service Business. Forthcoming in IADIS International Journal of WWW/Internet, vol.4, issue 1, pp. 70 – 82.
- McIlroy, M.D. 1968. 'Mass Produced' Software Components. In software Engineering: A Report on a Conference Sponsored by the NATO Science Committee. Editors Naur, P. and Randlell B. pp. 138 – 155: Garmish, 1968.
- Metsämuuronen, J. 2007. Tutkimuksen tekemisen perusteet ihmistieteissä. 4th edition. Gummerus Kirjapaino Oy. Vaajakoski. Finland.
- Miller, G.A. 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, Psychological Review, 63:81-97.
- Moser C., Junginger S., Brückmann M., Schöne K-M. 2009. Some Process Patterns for Enterprise Architecture Management. Patterns in Enterprise Architecture Management (PEAM2009) 2009. Accessed online on 23.02.2012 from <http://subs.emis.de/LNI/Proceedings/Proceedings150/8.pdf>.
- OMG 2011. Business Process Model and Notation (BPMN), version 2.0. Accessed online on 21.02.2012 from <http://www.omg.org/spec/BPMN/2.0>.
- Perry, D.E. and Wolf, A. L. 1992. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes. 17(4), pp. 40-52, October, 1992.

Riel, M. 2010. Understanding Action Research, Center For Collaborative Action Research. Pepperdine University. Accessed online on 20.02.2012 from <http://cadres.pepperdine.edu/ccar/define.html>.

Ross, J. W., Weill P., Robertson D.C., 2006. Enterprise architecture as strategy: creating a foundation for business execution. Harward Business School Press. USA.

SEI, 2011. Published Software Architecture Definitions. Software Engineering Institute (SEI), Carnegie Melon University. Accessed online on 27.02.2012 from <http://www.sei.cmu.edu/architecture/start/glossary/published.cfm>.

SIIA, 2001. Software as a Service: Strategic Backgrounder. Software & Information Industry Association (SIIA), Washington, D.C. USA.

Smolander K., Hoikka K., Isokallio J., Kataiokko M., Mäkelä T., 2002, What Is Included in Software Architecture? A Case Study in Three Software Organizations, Proceeding of the 9th IEEE International Conference on Engineering of Computer-Based Systems, p.131-138, April 08-11, 2002.

Suojanen, U. 1992. Toimintatutkimus koulutuksen ja ammatillisen kehittymisen välineenä. Finn Lectura Ab. Helsinki. Finland.

Sääksjärvi, M. et al., 2005. Evaluating the Software as a Service Business Model: From CPU Time-Sharing to Online Innovation Sharing. Proceedings of the IADIS International Conference e-Society 2005, Qawra, Malta, pp. 177 – 186.

Telematica Instituut 2005. Annual Report 2005. Accessed online on 6.3.2012 from <https://doc.telin.nl/dsweb/Get/Document-64437>.

The Open Group 2009. Archimate® 1.0 Specification. Accessed online on 6.3.2012 from http://www.opengroup.org/archimate/doc/ts_archimate/.

The Open Group 2011. TOGAF version 9.1. Accessed online on 23.02.2012 from <http://pubs.opengroup.org/architecture/togaf9-doc/arch/>.

The Open Group 2012. The SOA Source Book. 4th Edition. Accessed online on 24.02.2012 from <http://www.opengroup.org/soa/source-book/intro/>.

TripleTree, 2004. Software as a Service: Changing the Paradigm in the Software Industry. SIIA and TripleTree Industry Analysis Series, Washington, D.C.

Zachman, J. A. 1987. IBM Systems Journal: Volume 26, Number 3, pp. 276. Accessed online on 3.3.2012 from http://www.zachman.com/images/ZI_PICs/ibmsj2603e.pdf.

Wikipedia 2012a. Zachman Framework. Accessed online on 3.3.2012 from http://en.wikipedia.org/wiki/Zachman_Framework.

Wikipedia 2012b. IDEF. Accessed online on 3.3.2012 from <http://en.wikipedia.org/wiki/IDEF>.

Wikipedia 2012c. Unified Modelling Language. Accessed online on 6.3.2012 from http://en.wikipedia.org/wiki/Unified_Modeling_Language.

Wikipedia 2012d. Value Chain. Accessed online on 6.3.2012 from http://en.wikipedia.org/wiki/Value_chain.

Attachments

Attachment 1 - question set in 1st research iteration

Attachment 2 - question set in 2nd research iteration

Attachment 3 IJSTNNN Arek Architecture Governance (classified)

Attachment 4 IYAH032 Arkkitehtuurimenetelmisto (classified)

Attachment 5 IJSTNNN Example Architecture Description (classified)

Attachment 1 - question set in 1st research iteration

Onko nykyinen arkkitehtuurikuvaus mielestäsi ymmärrettävä?

Kuvaile omin sanoin dokumentin hyviä ja huonoja puolia.

Onko nykyinen arkkitehtuurikuvaus mielestäsi sopivalla tasolla kuvattu vai tulisiko sen olla karkeampi / tarkempi?

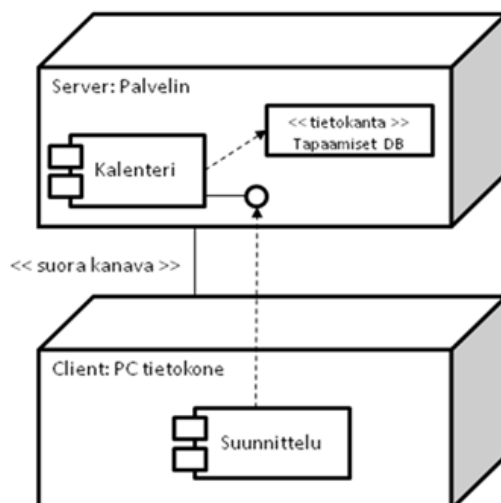
Mikä on mielestäsi tärkeintä tietoa edustamasi sidosryhmän kannalta esitettyssä dokumentissa?

Kuvaile omin sanoin mikä on hyödyllistä tietoa ja mikä ei.

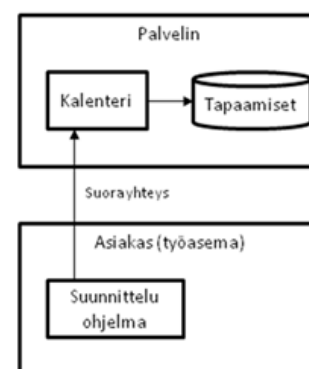
Kuinka usein kaipaat työssäsi vastaavankaltaista tietoa jota liitteen arkkitehtuuridokumentti sisältää ja koetko dokumentin olemassaolon tärkeäksi?

Onko jotain jonka koet puuttuvan arkkitehtuuridokumentista? Kuvaile omin sanoin mitä kaipaisit dokumenttiin lisää.

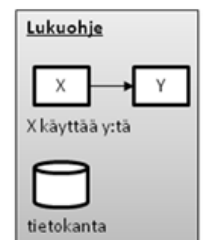
Liitteenä olevassa arkkitehtuuridokumentaatiossa ei käytetä ns. ”formaalia notaatiota” kuten UML-kuvaustapaa. Kumman kuvaustavan koet paremmaksi? Vastaamisen auttamiseksi alla on karkea, ei tutkimuksen kohdeyritykseen liittyvä, esimerkki näiden kahden eri kuvaustavan (UML notaatio vs. oma notaatio) eroista.



UML notaatio



Oma notaatio



Onko sinulla muita vapaamuotoisia kommentteja?

Attachment 2 - question set in 2nd research iteration

Onko arkkitehtuurikuvaus mielestäsi ymmärrettävä?

Kuvaile omin sanoin dokumentin hyviä ja huonoja puolia.

Onko arkkitehtuurikuvaus mielestäsi sopivalla tasolla kuvattu vai tulisiko sen olla karkeampi / tarkempi?

Mikä on mielestäsi tärkeintä tietoa edustamasi sidosryhmän kannalta esitettyssä dokumentissa?

Kuvaile omin sanoin mikä on hyödyllistä tietoa ja mikä ei.

Kuinka usein kaipaat työssäsi vastaavankaltaista tietoa jota liitteen arkkitehtuuridokumentti sisältää ja koetko dokumentin olemassaolon tärkeäksi?

Onko jotain jonka koet puuttuvan arkkitehtuuridokumentista tai onko jotain johon dokumentin tulisi mielestäsi vastata? Kuvaile omin sanoin mitä kaipaisit dokumenttiin lisää.

Koetko arkkitehtuurikuvauksen kuvaustavan (Archimate notaatio) hyväksi?

Onko sinulla muita vapaamuotoisia kommentteja?

Haastattelutilaisuuden loppuksi pääset tutustumaan verkkoversioon arkkitehtuurikuvauksesta, varauduthan loppuksi kommentoimaan myös kyseisen verkkoversion hyödyllisyyttä, mikäli se olisi esimerkiksi saatavillasi Arekin Intra- ja Extranet verkoista.

