HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Electrical and Communications Engineering

Kimmo Mattila

Telecommunications platform productisation for horizontal markets

Thesis submitted for the degree of Master of Science in Engineering
in Espoo 14th April 2003

Supervisor Raimo Kantola, Professor

Instructor: Esa Nurmi, Master of Science

TKK Sähkö- ja
tietoliikennetekniikan kirjasto
Otakaari 5 A
02150 ESPOO
09 -05- 2003

HELSINKI UNIVERSITY OF TECHNOLOGY Abstract of the Master's Thesis

Author:	Kimmo Mattila		
Name of the Thesis:	Telecommunications platform productisation for horizontal markets		
Date:	14.4.2003	**Number of pages**:	104
Department:	Department of Electrical and Communications Engineering		
Professorship:	S-38 Networking Technology		
Supervisor:	Raimo Kantola, Professor		
Instructor:	Esa Nurmi, Master of Science		

The telecommunications industry is moving to a separation of the user and control plane servers. The traffic in both of them is expected to use the IP-protocol for the transmission of information. The trend is to use and to make it easy to use modules provided by third parties as parts of these network elements. It has also been suggested that current players in the telecommunications industry could act in a horizontal business model, where they sell their components as parts of products made by other vendors.

The above mentioned business model and the ongoing standardisation work in the industry yield common, standard-based platforms for telecommunications systems. A platform is a set of subsystems providing application programming interfaces for application developers of derivative products. These kinds of platforms have previously been proprietary for each vendor. The case company is planning productisation of such a platform for the horizontal markets.

The possibility and interest of a vendor to buy a platform from a peer in the markets is based on the fact that when platforms are common and fulfil the same standards, the differentiation of products is done with the applications. Buying a platform from outside releases the resources of a vendor to focus on the application development.

In this thesis, the general requirements for a telecommunications platform, especially in horizontal markets, are studied. The focus is on the requirements that are not related to the actual functionality of the platform, since the internal customers and standardisation work in the industry have presented and will present the most important functional requirements. However, a challenge in the productisation work is to implement processes for handling the functional requirements of horizontal customers.

The study defines the terms 'telecommunications platform' and 'productisation'. Productisation means the definition work of sellable entities. The study presents the current situation of the platform in the case company and the organisation that is developing it. The study defines what actions belong to a productisation process and derives an action plan for the case company on how to solve the detected problems. It compares the present situation to theories found in the literature and to two product offerings in the market segment, which are used as examples.

Keywords: Productisation, horizontal markets, platform, telecommunications platform

TEKNILLINEN KORKEAKOULU | Diplomityön tiivistelmä

Tekijä:	Kimmo Mattila		
Työn nimi:	Tietoliikennejärjestelmäalustan tuotteistaminen horisontaalimarkkinoille		
Päivämäärä:	14.4.2003	**Sivumäärä**:	104
Osasto:	Sähkö- ja tietoliikennetekniikan osasto		
Professuuri:	S-38 Tietoverkkotekniikka		
Työn valvoja:	Professori Raimo Kantola		
Työn ohjaaja:	Diplomi-insinööri Esa Nurmi		

Televiestintäteollisuudessa ollaan siirtymässä erillisiin verkkoelementteihin käyttäjä- ja kontrollikerroksilla. Liikenteen molemmissa oletetaan näissä molemmissa siirtyvän IP-protokollan päällä kuljetettavaksi. Yleisenä suuntauksena on käyttää näissä verkkoelementeissä kolmansien osapuolien tuottamia ohjelmistokomponentteja ja tehdä niiden sovittaminen näihin tuotteisiin helpoksi. On myös esitetty, että nykyiset toimijat televiestintäteollisuudessa voisivat toimia horisontaalisessa liiketoimintaympäristössä, missä ne toimittaisivat omia komponenttejaan osaksi muiden valmistajien tuotteita.

Edellä mainittu liiketoimintamalli sekä käynnissä oleva standardointityö teollisuudenalalla tuovat esiin tarpeen televiestintäjärjestelmien yhteisille, standardipohjaisille järjestelmäalustoille. Järjestelmäalusta on joukko alijärjestelmiä, jotka tarjoavat sovellusohjelmointirajapintoja lopullisten verkkoelementtituotteiden kehittämiseen. Tällaiset alustat ovat aiemmin olleet valmistajakohtaisia. Toimeksiantajayrityksellä on suunnitelmia tällaisen järjestelmäalustan tuotteistamisesta horisontaalisille markkinoille.

Valmistajien kiinnostus ostaa järjestelmäalusta toiselta toimittajalta perustuu siihen, että alustojen ja niiden rajapintojen ollessa standardien mukaisia, tuotteiden erottuminen toisistaan saavutetaan sovelluksilla. Näin ollen Järjestelmäalustan hankkiminen toiselta toimittajalta mahdollistaa valmistajan intensiivisemmän keskittymisen omien sovellusten kehittämiseen.

Tässä työssä tutkitaan, mitkä ovat yleiset vaatimukset tuotteistetulle tuotteelle ja erityisesti sovellettuna televiestintäjärjestelmäalustoihin. Tutkimus painottuu järjestelmäalustan ei-toiminnallisiin vaatimuksiin, sillä alustan sisäiset asiakkaat ja teollisuudenalan standardointityö ovat jo esittäneet ja tulevat esittämään tärkeimmät toiminnalliset vaatimukset. Tuotteistamistyön yhtenä haasteena on kuitenkin muodostaa käytännöt horisontaaliasiakkaiden toiminnallisten vaatimusten käsittelylle.

Tutkimus selvittää termien televiestintäjärjestelmäalusta sekä tuotteistaminen merkityksen. Tuotteistamisella tarkoitetaan myytävissä olevien kokonaisuuksien määrittelemistä. Tämä työ esittelee toimeksiantajayrityksen nykyisen järjestelmäalustan sekä sitä kehittävän organisaation. Työssä määritellään mitä toimenpiteitä tuotteistamisprosessiin kuuluu sekä annetaan ehdotuksia siitä, miten toimeksiantajayrityksen tulisi ratkaista nykyisen järjestelmäalustan puutteet. Järjestelmäalustan nykytilaa on verrattu kirjallisuudessa annettuihin vaatimuksiin sekä kahteen esimerkkeinä käytettyyn saman teollisuudenalan tuotteeseen.

Avainsanat: tuotteistaminen, horisontaalimarkkinat, tietoliikennejärjestelmät, järjestelmäalustat

ACKNOWLEDGEMENTS

Many people have supported me during the process of writing this Master's Thesis. I am very grateful for them, as they have made it possible for me to work within an interesting subject and without having to think other difficult issues. Especially I want to thank the following people:

- Mr. Kari Seppälä for being my co-worker and instructor in the early phases of this project,
- My line manager Ville Makkonen for giving me the opportunity to concentrate on this thesis and keeping my other responsibilities as minimal as possible,
- Ms. Tarja Miettinen for her patient help in the documentation issues, for giving me hints on how to proceed and for forwarding me to the right people when necessary.
- My instructor Esa Nurmi for lots of time given for this project and for his supportive and friendly attitude also towards my simple questions,
- My folks Aila and Harri back home for frequently asking how I am doing but at the same time keeping the safe distance from commenting my daily work,
- And last, but not least, my soon-to-be wife Satu for keeping the low profile when I have been frustrated with the work and for doing our wedding-arrangements practically on her own during this spring.

Espoo, April 13, 2003

[illegible]

Kimmo Mattila

TABLE OF CONTENTS

GLOSSARY

3GPP	3rd generation partnership project
ADI	Application Development Interface
API	Application Programming Interface
ARPU	Average Return Per User
BCR	Basic customer requirements
BHCA	Busy hour call attempts
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
EJB	Enterprise Java Beans
FSN	Full Service Network
GUI	Graphical user interface
IP	Internet Protocol
IPR	Intellectual Property Rights
IRP	Integration Reference Point
ISO	International Organization for Standardization
ISV	Independent Software Vendor
IT	Information Technology
ITU	International Telecommunication Union
J2EE	Java 2 Enterprise Edition
OEM	Original Equipment Manufacturer
OSA	Open Service Architecture
PDF	Portable Document Format
PDM	Product Data Management
PDMA	Product Development and Management Association
PTO	Public Telecommunication Operator
R&D	Research and Development
RPM	RedHat Package Manager
SAF	Service Availability Forum
SCTP	Stream Control Transmission Protocol
SEE	Software Engineering Environment
UML	Unified Modelling Language
VAR	Value Added Reseller
XML	Extensible Markup Language

1. INTRODUCTION

1.1 Background

There is a trend towards a convergence of communication networks, where all traffic is based on the IP protocol. Drivers for this development are the desire to reduce system complexities and to decrease the cost of ownership. This development has raised a need for standardised telecommunications platforms. Services are built on top of these platforms. They offer standardised computing services that help the application developers, as they do not have to build basic functionalities themselves. These kinds of platform products must fulfil industry standards and offer common application programming interfaces (API) for developers.

For telecommunications equipment vendors this is a new playground, as products are no longer offered only through vertical proprietary routes. Instead, there is a new horizontal mode of business, where parts of products can be offered as platforms for the end products of other companies. These platforms must offer a standardised API for developers and ensure interoperability with co-operators and even with competitors. The customers in this business are different, because they are the developers of the end products, not their end users as before.

Fulfilling the expectations and requirements of horizontal markets is a new challenge for telecommunications equipment vendors. This thesis introduces the requirements for a product in this business area and gives suggestions for the case organisation on how to ensure the acceptance of its product in it.

1.2 Research problem and objectives of the study

The objective of this thesis is to define the requirements for a telecommunications platform in order to call it a complete product. The main problem is to find out what defects the platform under development in the case organisation has compared to the general requirements and some similar products in the industry.

The concern of the case organisation is that its telecommunications platform is currently designed for in-house use. It wants to find out what modifications and enhancements have to be made to be able to sell the platform as a complete product in horizontal markets.

The research problems can be categorised as two main problems and their subproblems as follows:

1. What are the general requirements for a telecommunications industry platform product?
    a. What are the general requirements for a product, especially in the telecommunications industry?
    b. What kind of concepts do other vendors provide?
2. What modifications and enhancements have to be made to the in-house platform in order to sell it as a product to the horizontal markets?
    a. What deficiencies does the current platform of the case organisation have compared to sellable products in the industry segment?
    b. How to solve the present situation's gap in a systematic way?

Solving the research problems through their subproblems gives a guideline for the case organisation of what requirements a telecommunications platform must fulfil in order to be successful in the horizontal markets of the new network infrastructure.

1.3 Methods and structure of the thesis

1.3.1 Methods

The literature research in the beginning of the thesis concentrates on defining some of the terms and trends in the industry segment. It is done to clarify the environment for both the author and the reader. The literature part is rather large, although there is an attempt to keep the size of each subsection compact. The reason is that the thesis combines issues from several domains to understand the whole scope of the study.

Business science is by its nature mostly a science of planning. Therefore, ideal normative methods and results are rarely available. Olkkonen (1993) says that it is more a question of planning and evaluating plans against present reality. Hirsjärvi et al. (1998) list features of applied research. Those features indicate that applied research is rather solving problems than gaining knowledge, rather generating broad effects than measuring and testing variable relationships and rather developing programs or services than developing and testing theories. Applied research has little similarities to other research and researchers in it are generalists rather than specialists (Hirsjärvi et al., p 128).

Olkkonen presents a formula for science of planning, which contains four stages (Olkkonen, pp 32-33):

1. Define the concept of the design in hand accurately and preferably in your own words. It is important to set goals for the design and to define its relationship to actions.
2. From definitions, criteria for the design's goodness are derived using literature and common sense. Using these criteria, it should be possible to evaluate whether a design is better or worse than some other.
3. Using Step 2, the characteristics of a good design are developed.
4. Finally, the results of Step 3 are compared to the present situation and new, advantageous features are seen. If possible, this is verified by testing.

The structure of the empirical part of the thesis follows the technical norm of economics science, originally referred to as technical norm of practical syllogism by von Wright. In this technical norm, the situation is A, it is wanted to be B, and therefore actions X have to be taken (see Figure 1). In this thesis, situation A is known, although it is not very clear, but it is explained in as much detail as possible. Situation B in the future is also presented briefly, as there is a clear vision of it. The problem is to define the actions X that have to be taken to change the situation from A to B (Olkkonen, p 57).

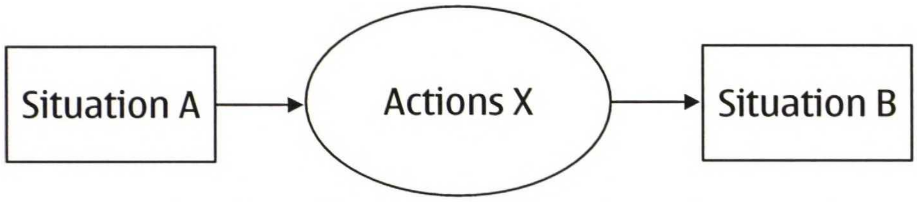

Figure 1: Technical norm of practical syllogism (Olkkonen, p 57, originally Wright)

1.3.2 Structure

The thesis is divided into 9 chapters, which form up a logical insight into the problem at hand.

In Chapter 2, the current trends in server markets and the idea of horizontal markets is introduced to give an overview and reasoning of what a telecommunications platform is needed for and why it is developed. Chapter 3 presents general requirements for an industry product and describes the various approaches for productisation, which are found from the literature. Chapter 4 introduces the idea of product platforms and especially telecommunications platforms. It also describes the characteristics of software products and platforms as well as their development work.

Chapter 5 studies some products in the market segment to give an idea of what are the characteristics for a product usually in this area. The products are studied in the sense of what are the non-technical features that make them products rather than their technical functionalities. The chapter also introduces one important standardisation work that is ongoing in the telecommunications platform area and the requirements it induces to a platform product.

In Chapter 6, a platform currently developed by the case company is introduced. Its core features are presented but its non-technical features are studied more carefully. In addition, the problems it is likely to face in the market area are compared to the requirements of the literature references and to the solutions of other vendors.

Finally, in Chapter 7, suggestions are given to the case company on activities that have to be done in order to define a sellable platform product for horizontal markets. It includes defining the areas where most of the defects occur and giving a plan on how these gaps should be solved.

Chapters 8 and 9 discuss the applicability of the results, give a conclusion to the thesis and present some thoughts on further study in the area.

1.4 Introduction of the case company

The case company is an international telecommunications equipment vendor. One of its several business groups provides mobile, broadband and IP networks and services related to them. This business group consists of several different divisions.

The division in question develops radio access, mobility core and wireless broadband network solutions. It also produces mobile Internet and multimedia enablers and operations support systems for network providers and operators in their transition from voice communication to the mobile Internet business. In the core networks area, it develops circuit switched and packet switched solutions, offering both evolutionary and revolutionary paths to all-IP networks (Case company intranet, 22.10.2002).

This thesis is done for one of the units of the above-mentioned division. The purpose of the unit is to create enabling platforms for application, control and gateway layer products for other units both inside and outside of the division. This thesis is a part of their study about moving to a horizontal business mode and selling the platforms also outside the company (Case company intranet, 22.10.2002).

2. SERVER PLATFORMS AND HORIZONTAL MARKETS

2.1 Introduction of the application domain

In next generation networks, all traffic is expected to be based on the Internet Protocol (IP). In technological visions, the core network consists of an IP backbone, which is surrounded by an intelligent edge of access gateways for different terminals. Customers are no longer distinguished based on the access medium but on the services they use. A concept of global reachability requires access independent systems. Intelligent edge connects users to different services, which are implemented in service point servers (MITA, pp 19-21, 239).

There are currently many different kinds of networks, such as mobile, fixed and the Internet. As the traffic type in these is merging, so has to do the technology. A general industry view assumes all traffic being IP based. It allows all communication services to be carried over a single transport network. This enables the integration of voice, data and multimedia services, which offers operators new revenue possibilities in addition to cost savings, scalability, flexibility and efficient network operations. It is seen as the only solution for handling the constantly increasing traffic volumes and changing demand for data rate in the service-based network (MITA, p 239).

Server types in the network include (MITA, p 30):

- Access network servers (e.g. Radio Network Controller (RNC), Media Gateway (MGW) or Gateway GPRS Serving Node (GGSN)),
- Connectivity servers (e.g. Home Location Register (HLR) or Connection Processing Server (CPS)),
- Application servers (e.g. Web or Email servers),
- Enabling service servers (e.g. directory, presence or content delivery servers).

Three main roles can be identified from communication technology: access, transmission and services. In telecommunications, operator organisations have so far been rather vertically oriented, as they have kept all three roles. In the Internet, however, the approach has right from the beginning been to divide these three, which has given opportunities for new, smaller players to enter the market (Webb, p 229).

Regulators have noticed the benefits of this development also. A European Union policy is that networks should be open to all users and service providers so that any willing party can attach to the existing infrastructure. This requires rules by which networks become interconnectable and interoperable, i.e. standard solutions are needed (EU, p 10).

Benefits for operators from new network architectures are (Webb, pp 67-69):

- Reduced cost of operation, because only one core network has to be maintained for different kind of access methods.
- Ability to carry a variety of traffic types through one network.
- Simple access to already available Internet-based applications.
- Quick provision of new services since multiple third party suppliers can produce them. They do not have to care about the underlying network architecture, because they use a common protocol.

Cost and flexibility are the drivers for the transition from circuit-switched networks to all-IP ones. This philosophy will bring besides the IP-protocol also modularity and end-to-end approach to the telecommunications. IPv6 is even more suitable for wireless Internet. Its benefits are better scalability, a huge address space and integrated security support (Webb, p 241).

The deregulation of telecommunications sector benefits the users by more choice, more solutions that are standard based and lower prices. Arrival of new companies means more products and services, they are of better quality and their prices are cheaper (EU, p 9).

Today vendors still often introduce specialised and proprietary designs and operating systems for specialised workloads, which makes it impossible to gain economies of scale and application portability (Weiss, p 1). Fabricius (1998) expects the telecommunications vendors to split into three segments (Fabricius, p 1):

- Full service network (FSN) solutions vendors (e.g. system providers like Ericsson, Nokia or Siemens),
- Niche and original equipment manufacturer (OEM) companies, whose products are parts of the FSN product offerings,
- Component and OEM vendors, who sell components to the two former ones.

As a future trend Fabricius sees that large FSN vendors are moving from full in-house product concepts to outsourcing strategies to avoid losing their focus or competitiveness by choosing the wrong technology. At the same time, niche or OEM vendors have an opportunity to enter the business as they can fulfil the gaps that bigger vendors have left open due to their outsourcing (Fabricius, p 5).

For niche and OEM vendors staying at the leading edge of development is of key importance. That is because their customers, bigger vendors or telecommunications operators tend to buy the components or solutions that are technologically best and of cheapest price. Niche and OEM vendors are characterised by their innovative organisation, high performance of research and development (R&D), strong product management, outstanding logistics and clean standardised software interfaces to the products of other vendors (Fabricius, pp 7-8).

In reshaping the business, the telecommunications providers have a foundation role. They are the coordinators and guarantors of service quality across the network. They must learn to co-operate with providers of other layers of infrastructure to assure the effective delivery of services across the network (Wainewright, p 15).

2.2 Telecommunications platform products

Experts in MITA (2001) suggest that there are three principles according to which a network architecture model should be designed. These principles are (MITA, p 27):

- Architecture is divided into independent sub-architectures.
- Architecture is open, modular and hierarchical.
- Architecture can tolerate different technological changes of individual components. Change inside a component should not affect the whole architecture.

The Internet value added services era that we are facing requires ever more powerful and flexible software. During the coming decades, new areas of functionality and value added services are expected to emerge. Focus of information technology (IT) industry will move from dealing with platforms to fully exploiting them and making products on top of them. Similarly, the focus of embedded software is moving from producing software for specific hardware to building software platforms for others to use. Application developers can more easily build intelligence to new machines, as they do not have to worry about the platform. One type of software platforms is expected to be the standardised operating systems that are ported to new types of machines to help in building embedded software (Hoch et al., pp 217-219).

Meyer and Lehnerd (1997) stated that the work put on the development of a platform pays well off, as new market segments can be addressed with it and the development of new derivative products is easier and cheaper. The latter comes true as developers of derivative products have familiar basic blocks, a common look and feel, which they can use to build new products (Meyer & Lehnerd, pp 35-37).

A common view in designing network elements is to use a layered element model. In this model, the architecture is divided into three layers: the application, middleware and platform layers (see Figure 2). The application layer contains user interfaces, application logic and support functions for them. Middleware layer isolates the application layer from the platform and thereby allows the application layer to be built on top of multiple platforms. It provides tools for application designers to implement new services without touching the platform. The platform layer contains operating system specific software and hardware.

In order to fully separate these layers, interfaces between them must be well defined and they cannot overlap. Typically, interfaces are provided to other layer programmers as application programming interfaces (API) (MITA, pp 27-28). The word ‘platform’ is nowadays used much in different contexts and the assumed meaning of it varies a lot. In this thesis, the word refers to a telecommunications platform that offers a set of APIs for developers of different telecommunications applications. From the viewpoint of the case organisation, a platform contains both the platform and the middleware layer of the view in MITA.

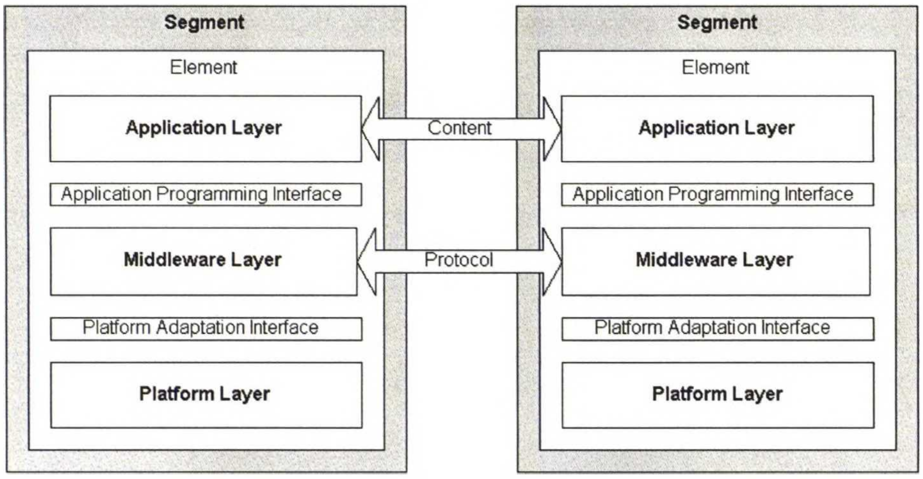

Figure 2: Layered elements and interaction between them (MITA, p 28).

In the MITA view, the middleware layer contains two parts, the operating system part and the platform support stack and Internet protocol stack part. The layer uses services provided by the platform layer through platform adaptation interface.

The platform layer consists of operating system core modules, device drivers and other more hardware related functions. It provides a platform adaptation interface to upper layers for local resources and as well an access independent interface for accessing network resources (MITA, pp 35-36).

The middleware and application layer communicate also through their external interfaces with their peers in other network elements, using of course resources from their lower layers. The application layer interchanges application content between elements and the sessions are build up with protocol interfaces between middleware layers of different elements.

Supporting new requirements of applications comes from the fact that the content and applications are separated in the future. As applications are distributed over the network, the infrastructure must ensure the principles of consistency, authenticity, timeliness, integrity and persistency to complete the action processes between them (Wainewright, pp 6-7).

A success recipe for the future is to use standardised components on top of standard platform architectures. Technical infrastructure for 'componentware' shall be a platform and its extensions that are compound from components and their interfaces (Sallinen & Seppänen, p 17).

2.3 Entering new markets

In rapid technological and economical change, a company cannot solely rely on its existing products. It must have an innovative new-product development process. Product innovations include improvements to existing products, designing new ones and making extensions to make new product lines and enter new market segments. Kotler et al. (1999) call this process new-product development. They explain a product innovation to be a new idea, a product or technology that is delivered to customers who perceive it novel or new. The innovation process identifies, creates and delivers a product or service that has not been in the market area before. It should not be mixed with invention, which is a new idea or technology that may or may not deliver benefits for any customer (Kotler et al., pp 603-604).

Technological changes shape the levels of business and can give new opportunities for organisations to address completely new markets. Projecting new markets and opportunities requires co-operation from various entities in a company, including marketing, sales, finance and product development (O'Grady, pp 147-148).

According to Cooper and Edgett (1999), the word 'new' in product development means a product that represents a change from the customer's perspective. It is a change that is noticeable. They also mention, that a project can be new either from internal (company) or external (marketplace) perspective. Cooper and Edgett provide six classifications of what can be treated as new. The approach is derived from the service sector (Cooper & Edgett, pp 10-13):

- New to the world services, that have never been seen anywhere else before,
- A new service line that is not new to the markets but to the company,
- Addition to an existing service line of a company,
- Improvements to existing services,
- Repositioning of services to new markets or customers,
- Cost reductions, which are modifications to services that are not visible to the customers.

In the service sector, a transformation is a process where a company composes new service products from its existing knowledge and experience to a new common knowledge. It is not a complete process, but a set of actions for product development to use as a means to develop different services (Vaattovaara, p 3). Cooper and Edgett remind that probably most of the companies use a mix of the above-mentioned new-projects all the time (Cooper & Edgett, p 13). The service sector is also used as an example in this thesis, because there too the productisation means packaging the existing knowledge of a company in to sellable entities.

There are many examples of how new products that were thought to be good, fail. Reasons for this can be in wrong estimates of markets, designing the product for wrong target group or failures in the product pricing or marketing. Kotler et al. list success factors that are common to new products that have succeeded (Kotler et al., pp 604-605):

- Unique superior product (quality, new features, higher value for users),
- Well-defined product concept before development,
- Meets market needs,
- Technology and marketing synergy,
- Quality of execution,
- Market attractiveness.

Fulfilling market needs and finding good enough quality in horizontal markets is an even bigger challenge, as the customers can be company's own competitors in the end products. They must be ensured that there are true benefits for them to start using a platform made by an other vendor.

2.4 The idea of horizontal markets

Concentration on the core competencies has replaced the vertical integration as an organisational principle in telecommunications industry (see Figure 3). One reason for this development is that as business is global, companies do not have to secure their backs by attending a large number of different industries. Instead, they take the advantage from regional balancing globally. Global competition requires them to focus more on the core functions. One cannot afford to oversee much functionality with little time given to all of them (Lillrank and Forssen, p 19). The rapid development of technology requires companies to focus on their core businesses and seek for opportunities to concentrate only on a part of the value-chain. The same development allows other firms to find their opportunity in other segments or in tying up these segments together (Vaattovaara, p 1). Creating an integrated, end-to-end delivery infrastructure requires close co-operation between these providers and enterprises (Wainewright, p 7).

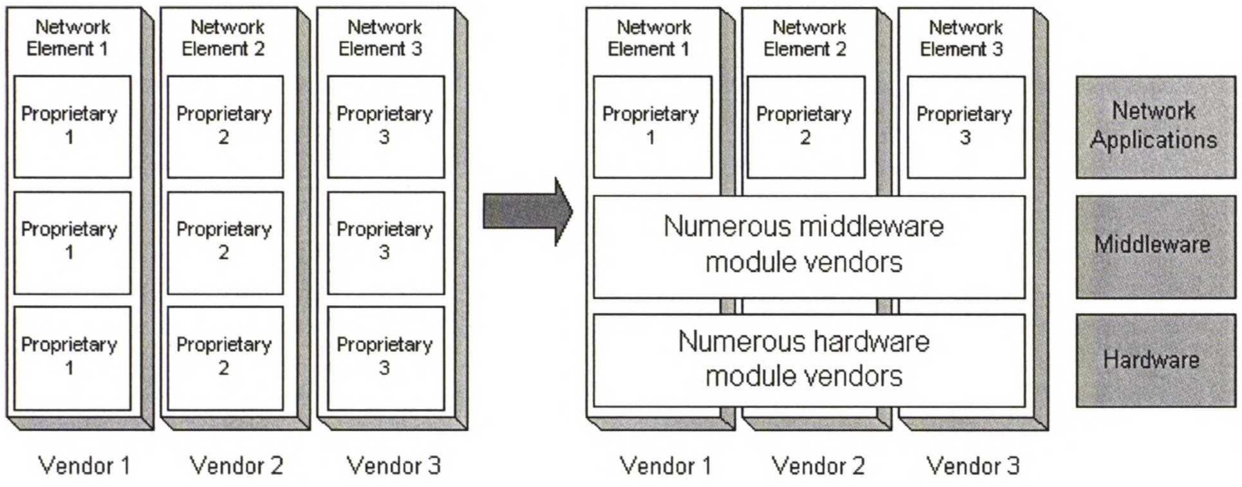

Figure 3: From proprietary to standard based model of the telecom equipment markets

The strategic positioning definitions by Porter (1996) introduce different approaches for the product or service portfolio of a company. One approach is that it tries to serve all the needs of a certain customer group in the industry segment, which is called needs-based positioning. Another approach is to provide a subset of products or services to the whole industry segment. This is called variety-based positioning (Vaattovaara, p 26, originally Porter 1996).

It is commonly believed that large telecommunications companies can do almost any projects they want, that they have infinite resources for building and selling complex systems. This is not true, because large companies have many different projects ongoing at the same time and must constantly allocate resources between them according to priorities. The recent economical recession in the industry segment has even accelerated this development. As a result for this, smaller companies are sometimes more capable of developing some specific technology. In larger companies, especially new innovative technology areas are harder to get through because these companies have to concentrate also on maintaining the existing products. For smaller companies the development strategy is much simpler. They cannot afford having specialists for several different technology areas in their payroll. Instead, they concentrate on the chosen product and the chosen technology and can be very efficient in implementing it (Webb, pp 150-151).

Wainewright explained what is happening in the pure software industry. His list of three types of independent software vendors (ISV), which operate in different levels of the software architecture, can be adapted to telecommunications also (Wainewright, p 17):

- Application ISVs who deliver complete packages of software to perform a certain action as a whole.
- Infrastructure ISVs who develop platforms and tools, which are used to design, build and manage components and applications.
- Web service ISVs who develop software for single simple functions that are a part of the offerings of application ISVs.

A company should fit itself to one or several of these categories. It is seen wise that companies try to do co-operation in these fields and not to compete in every one of these sectors. Instead, they can provide partial solutions to a competitor and similarly buy some parts from their competitors. Markets that are structured this way are called horizontal markets.

A part of the horizontal approach originates from the critical importance of partnering. Successful companies admit that they cannot do everything on their own. Partners are used to fulfil technology and time-to-market gaps. It is important to have strong partners and many of them. Successful companies have partner webs that include several informal but yet performance-driven partnerships (Hoch et al., p 181).

Companies must ensure that their solutions are not tied to near partners only. A common trend in the software business is that end-users pick their favourite combination of suppliers to form their final implementation. In this sense, it is critical to use standard-proof interfaces and listen to the end-user requirements and expectations (Hoch et al., pp 190-191).

As partner webs are formed, they must also compete against other webs. There is usually no single monopolistic solution worldwide for any application. Webs are competing with each other, which is called inter-competition. At the same time though, companies that are committed to the same partner web using a certain technology, compete against each other inside the web for their share of the pie. This is called intra-competition. A stiff intra-competition actually helps companies to give their best and therefore provides good enough solutions to win the inter-competitive battle (Hoch et al., pp 200-202).

As webs are formed, some companies always put larger effort on them than others. They are called the web shapers, as they establish the platform and control the key components (for example Microsoft with Windows and SUN with the Java platform). Other companies follow these shapers, adapt the technology and expand the web. They are called the web adapters. They are attracted by the growing revenue potential inside the web and are keen to join it (Hoch et al., pp 203-204).

Division can also be made to module providers and integrators. Module providers produce modules that integrators integrate into their products. O'Grady (1999) stated that an integrator usually determines the architecture of the product (acting as a web shaper). However, the current trend is to work on the architecture together with the suppliers, customers and competitors, and to this way create a mutual marketplace together (O'Grady, pp 18-20). One could think that, in the end, shapers still do better since they can control the web infrastructure. In fact, this is not necessarily the case, because some companies are better in shaping the web while others are in adapting it. The situation can go vice versa when companies play different roles in different webs (Hoch et al., pp 203-204).

According to Hoch et al., although shapers always give away some value when letting in some adapters, the growth of the web is the only way to grow the shaper company's own business. This kind of approach often pays back big over the time. For example, Sony got huge revenues from its PlayStation game console by giving away the development of games for it to independent developers. In some cases, resistance to give away value to partners can collapse the web thoroughly (Hoch et al., pp 204-206).

Organisations must decide where they are in the module support chain, are they module suppliers or integrators. Integrators assemble final products from modules provided by suppliers and are responsible for products meeting the customer requirements. The task of the suppliers is to provide modules that are compatible with required standards and interfaces. Naturally, some companies can belong to both of these groups (O'Grady, pp 43-47). There are problems in this approach, though. Offering a platform in a segment where a company itself competes results in "church and state" dilemma. For example, SUN Microsystems owns the Java and J2EE standards but has also an own competitive J2EE solution (SUN ONE Application Server) (Natis, p 4).

Integrators can either use commercial off the shelf (COTS) modules or use specially developed modules from an internal or external supplier. The specially developed modules may still be based on industry standards. Using internal suppliers makes the organisation to act as both a supplier and an integrator (O'Grady, pp 43-47).

The generic COTS modules are usually a safer choice, because they have already been in use and tested in practice, unless of course it is the first user in question. They also are likely to face competition, which reduces development times and costs and which should increase quality. There are, however, problems related to them. Suitable COTS modules might not always be available, so product cannot solely be based on those, which makes the architecture planning difficult. Relying solely on COTS components opens the product also to more probable competition, as there are no proprietary entry-barrier elements. In a COTS-based product environment, competitive advantage is often achieved through other aspects of a product such as support services, quality or reliability. Using specially developed modules gives the integrator a possibility to implement the functionality of the product just as it wants. It also makes it possible to retain the intellectual property rights within the integrator company and to address a part of the variable and capital costs of development to the supplier. By combining both approaches, an integrator can achieve faster development cycles and lower costs by using COTS modules and gain competitive advantage and barriers to entry the market with the special modules (O'Grady, pp 45-48).

Component and OEM vendors are seen as sub-suppliers of components and part products to other vendors. They do not have a relationship with the final customer at all. Their strategy is to gain as many big vendors for their customers as possible to gain the economies of scale and therefore competitive prices. Characteristics for this kind of a vendor are OEM partnerships, a generic product platform with standard hardware and software interfaces that enable easy integration with third party elements and a focus on high volumes and low costs, which means a little smaller margins and a strong product management (Fabricius, p 8).

One problem for the industry with increasing use of platforms and partner webs comes from the issue of who owns the customer interface, especially in embedded software. Especially this applies to the maintenance phase of relationships. If the value chain of a product is very long, it can be problematic for the customer or the nearest provider to define where, for example, noticed problems regarding a certain part of the product are reported. New types of licence and service agreements must be negotiated for these relationships (Hoch et al., p 220).

3. PRODUCTS AND PRODUCTISATION

3.1 Building a product

3.1.1 Components of a product

What is a product? Kotler et al. define it to be anything that can be offered to a market for attention, acquisition, use or consumption and that might satisfy a customer want or a need. The term 'product' means more than just tangible objects. Besides physical objects, a product offering can mean services, persons, places, organisations, ideas or a mix of these (Kotler et al., 561). Clarke (1997) puts it even simpler, as he states that a product is a normal offering of a supplier, which can be ordered by nominating the product identifier. Product Development and Management Association (PDMA) term definitions define the term 'product' to describe all goods, services and knowledge that can be sold. Products are collections of attributes (features, functions, benefits and uses) and they can be either tangible, as in the case of physical goods, or intangible, as in the case of services, or they can be combinations of those two (PDMA glossary).

A company must decide what are its core products and what can be left optional. A product is composed of core features, that give the product its main functionality and the main benefit for the user, enabling features that are not core features but are required in order for the core feature to function and supporting features that can be left optional and priced separately (Sipilä 1996, p 64).

Mattila (2000) defines the same division in service sector so, that a product contains (Mattila, pp 35-36):

- The core service, which is the one that the customer really wants and which delivers the user need,
- Support services that make the use of the core service easier,
- Additional services that are optional parts in the offering and enrich the value of the product to the customer.

Additional service packages are seen as an essential part of the offering. Customers want to get the additional and extra services from the same provider as the basic service (Mattila, pp 35-36). Besides the core product itself, Vitikainen (1995) included into a product an imaginary product and additional services and reserved a possibility for some new solutions also. To the imaginary product she included the product name, trademark, outfit, package, perceived quality and the product description. To the additional services she included the delivery, installation, training, warranty, finance, service, user instructions and the spare part support (Vitikainen, pp 42-43).

Kotler et al. divide a product similarly into three levels: the core product, the actual product and the augmented product (see Figure 4). Core product is the one that the customer is really buying. It gives the problem solving entities or core benefits for the user. The actual product is what the producer really produces and what the customer really sees. It is built around the core product. Core product has attributes like quality level, features, styling, brand and packaging. In a technical product, features are the ones that implement the core problem solving functionality. The augmented product is finally built by adding consumer services and benefits on top of the core and actual products. Services and benefits are, for example, warranty, training, maintenance and installation as well as after sales customer support. These augmented features eventually become an essential part of the product for the customer and they cannot be left out from later versions (Kotler et al., p 561).

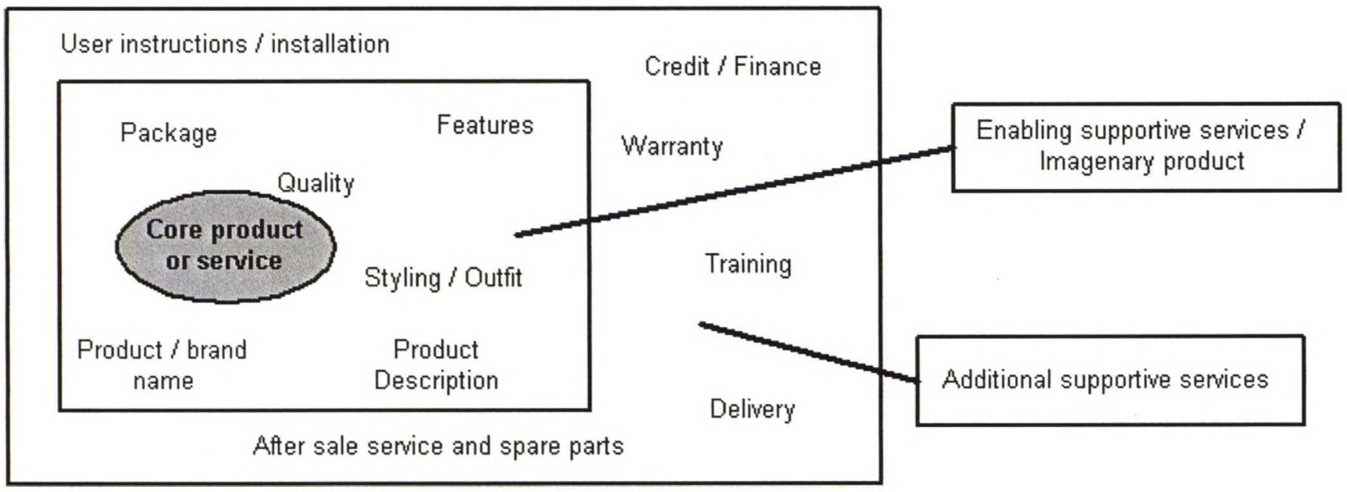
User instructions / installation
Credit / Finance
Package
Features
Warranty
Enabling supportive services / Imagenary product
Quality
Core product or service
Styling / Outfit
Training
Product / brand name
Product Description
Additional supportive services
Delivery
After sale service and spare parts

Figure 4: Three levels of a product (modified from Sipilä 1996 p 64, Vitikainen p 43 and Kotler et al. p 562).

One approach is to treat the components as individual products and sell them separately. Packaging these components into a sellable entity is seen only as a means to help the buying decision of the customer (Sipilä 1996, p 65).

Requirements for an industry product in general are (Junttila, p 30, originally Jaakkola & Tunkelo, 1987):

- Customer required features,
- Price/quality ratio,
- Usability,
- Maintainability,
- Usage security,
- Design,
- Ease of installation,
- Applicability and environment,
- Environments effect on the product,
- Performance,
- Reliability,
- Human factors,
- Ergonomic factors,
- Production issues,
- Packaging and distribution channel requirements,
- Interfaces to near environment,
- Product's effect on environment.

All of the above-mentioned issues are in some sense applicable also for the software and telecommunications industry, if some special conditions are taken into account. A successful product must fulfil minimum requirements from all categories and beyond that have some superior features to gain competitive advantage (Junttila, p 31)

Besides product functionality, a vital issue for a product to be accepted by the customers is that it is easy for them to buy the product. That is, there has to be an effective distribution channel. There has to be personnel that take care of the effective delivery of the product to a customer. Delivery can be handled by the producer company itself and leave only the marketing and negotiation tasks to the distribution organisation. Junttila states that in a software company the distribution organisation is also responsible of the training, seminars and guidance that is defined in contracts (Junttila, pp 75-76).

For the distribution channel, the sales organisation for a product must be chosen. It can affect the sales results of the product considerably. It must be noticed, that the sales organisation probably also needs training and consultation about the product in various points of their sales process.

Product branding is vital when selling and marketing products, though it is more important in consumer markets. It can help in differentiating the products. Brand is a name, a symbol, a design or a combination of these that identifies a product and differentiates it from its competitors. Brands, unlike other IPRs, do not have expiration days; they are dedicated to their owner forever (Kotler et al., pp 570-575). There are four options on how branding can be done (Kotler et al., pp 579-580):

- Using manufacturer's own brand,
- Using a private brand (owned by a middleman, a distributor or a store),
- Licensing a brand (used in royalty products, for example Porsche sunglasses),
- Co-branding (usually done under license terms, for example Kellogg's Healthy Choice).

When delivering a platform product for horizontal markets, i.e. also for possible competitors, branding is somewhat a problem. It is possible that the customers do not want to be identified by a brand of the original company. On the other hand, the product must have a name in order to sell it to any customers, since it is not only a simple part or raw material. However, the brand of a platform shall be recognized on its target market, among the network application developers, not among the end users of a network element.

Significantly important factors for any product are the decent user, installation, service and system manuals. They are important, for example, in adapting to different market environments (Vitikainen, p 20). Documentation items that are required for a software system product include (Junttila, p 61):

- Installation guides,
- User guides,
- Documentation of services,
- Training material,
- Marketing material,
- Maintenance and other information material for internal or distribution channel use,
- Agreements, licences and registration documents.

Target groups for documentation are the customers, the marketing and distribution channels, trainers, the maintenance and product support as well as the management and product development. Above-mentioned documents exclude technical documentation, which is meant for internal use only (Junttila, p 61). However, in the case of a platform, some technical documents must be provided, since the target customers are application developers. Part of the documentation can nowadays be in electrical form. In theory, all documentation, starting from the source code in a software product, can be implemented as a big amount of hypertext documents that are linked together (Haikala & Märijärvi, p 65). Junttila includes into the product documentation also online helps in the software, but for a horizontal platform product, that is more a functional requirement, although the trend for software products is to move from paper versions of user guides to online helps (Junttila, pp 66-67).

3.1.2 Product categorisation

A product is an article, a service or a combination of them provided for a customer. It can be divided into pieces and its contents and structure can be presented based on different criteria, for example by its bill-of-material or functionality structure. A product can be understood in a broad or in a narrow sense. In the narrow approach, all subcomponents are products themselves whereas the broader view emphasises the total offering produced for the customer (Lahtinen, p 35). Tiihonen (1998) reminds that a product is not necessarily an end product sold to the customer. Instead, it can be a component of a larger product that is sold to another unit in the same or another company (Tiihonen I, p 2).

Products can be categorized in a number of ways. One of the most obvious ones is the division to consumer and industrial products. Consumer products are bought by their final users whereas industrial products are bought by businesses for further processing. Thus, the purpose of the product is different than in consumer markets. Using this categorisation, a same product can be an industrial product in one occasion and a consumer product in another. Industrial products can be further divided into three categories (Kotler et al., pp 563-566):

- Materials and parts are goods that become a part of the final product through processing or as a component. OEM products belong to this category.
- Capital items are products that help the manufacturer in the process as installations. Examples of these are factories or accessory equipment like drills or computers.
- Supplies and services are products that are not in touch with the actual product at all. Supplies are operational facilities like heat, electricity or paper and repair or maintenance items like paint or nails. Services can further be divided into maintenance and repair services and business advisory services.

Traded items generally can be categorised by the degree to which they are productised. Clarke defines four classes of products:

- Standard products are items that can be ordered from a supplier catalogue. It is common for goods and services that they are productised, because it usually reduces their unit cost.
- Commodities are products that exist in an identifiable form and are available from a variety of sources. Typical examples are stocks and other financial instruments and primary products like crude oil or coffee.
- Custom-build products, which are designed to meet the specific needs of a specific customer for a specific purpose and are bought through a tendering process.

- Customised products, which are used in situations, where the product or service does not exactly solve the needs of a customer, but they can be fulfilled by modifying the basic product. This is especially true when the product is a combination of goods and services.

A telecommunications platform can be thought to belong to Kotler et al.'s category 'materials and parts'. Selling it to another vendor for application development requires of course also repair services and advisory consulting.

3.1.3 Product requirements

There are several different requirements for a product in order it to fulfil all the expectations of a customer and to be suitable for the use that it is meant. For example, there can be preferred production costs, selling price or quality requirements defined (Lahtinen, p 36). Those are defined to help the comparison of different products and their business cases. The customer requirements must also be categorised. There is always both a positive and a negative side of customer requirements, things that the customers want and things that they especially do not want (Junttila, p 29).

Liukko (1994) categorises the contents of a product into three categories: minimum requirements, additional features and new features. Minimum requirements are things like security, faultlessness and reliability so that the user can do the tasks he or she assumed to do with the product. These are also called the basic customer requirements (BCR). Additional features are not necessary but customers are positively surprised having them. New features are the ones that differentiate products from each other. They are features that are not required in the cheapest basic products (Junttila, p 29, originally Liukko 1994).

Meyer and Lehnerd propose that customer requirements should be clear to all levels of an organisation, no one organisational part should own the customer. There are two kinds of customer requirements, perceived and unperceived. The latter ones are also called latent needs. The perceived needs are directly required by the customers but the latent ones are often discovered somewhere totally outside of market researches. Often the engineers themselves discover these needs as users of the product. In the case of the horizontal markets, the fact of having internal customers helps a lot in discovering these needs. Meyer and Lehnerd present that the goal of understanding the customer requirements should be common to both the marketing and the R&D. Too often, they say, it happens that junior marketing managers are in contact with the R&D to discuss customer requirements and improving a product. Those meetings are the places where most experienced marketing people should give their effort to the product (Meyer & Lehnerd, pp 44-45).

Requirements for a system may be either functional or non-functional. The formers are statements of services that the system should provide, how the system should respond to inputs and how it should act in certain situations. Non-functional requirements are constraints for the services. They can be constraints for timing or for the development process or statements about the supported standards (Sommerville, pp 118-119).

The non-functional requirements can sometimes be more important than the functional ones. If some constraints are not met, customers might not accept the system. In extreme cases, it cannot be taken into use at all. The non-functional requirements reflect often to the process rather than the product itself. They are presented from system quality or system maintainability points of view. Sommerville (1995) divides the non-functional requirements into three categories (see Figure 5, Sommerville, pp 130-131):

- Product requirements,
- Organisational requirements,
- External requirements.

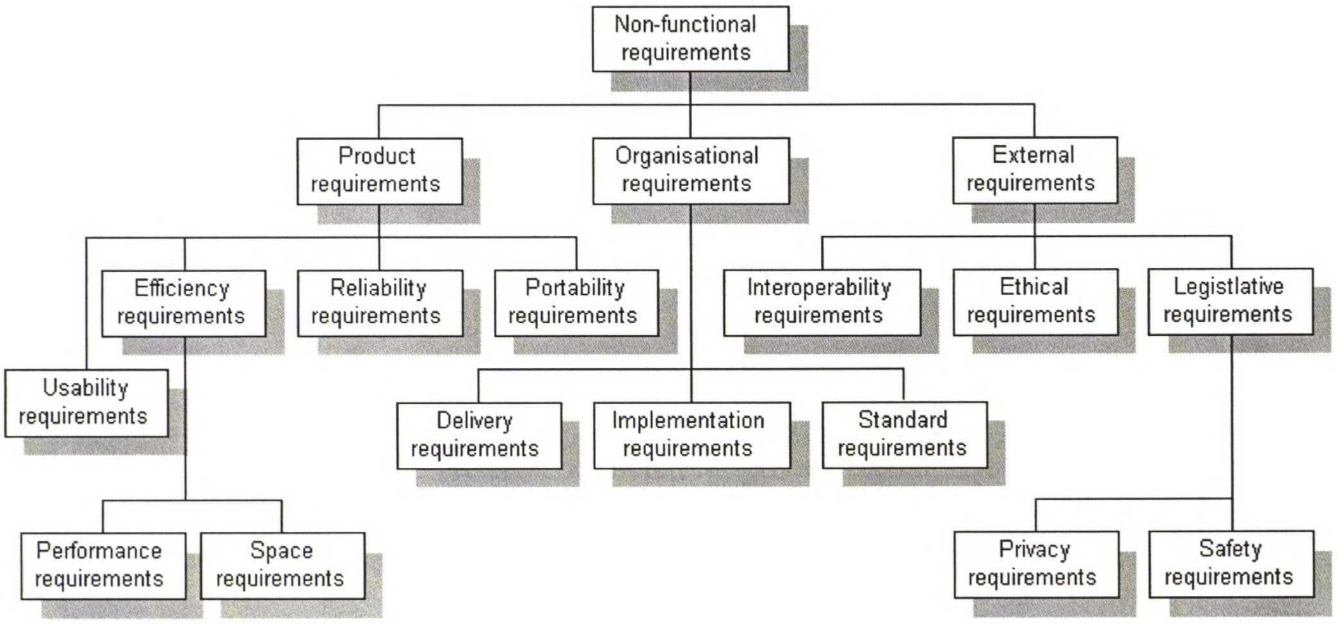

Figure 5: Types of non-functional requirements (Sommerville 1995, p 131).

In this thesis, the emphasis is mainly on the organisational and external requirements, as pure product requirements for the system in this case are already derived from the needs of the internal customers. In fact, as a mostly existing product is productised to a new market, the product decisions already made for it act as constraints for the new product. One must also notice that the internal customers have not presented all the requirements there are. The platform development engineers discover the latent needs also themselves and there are requirements that originate from standards.

3.1.4 Discovering successful products, product strategy

Successful enterprises must be evolutionary; they must make new products to hit new markets. This is achieved through periodic enhancements to the product and its manufacturing technologies. Some of them are breakthroughs while others are only incremental enhancements. To be successful, a company must produce a continuous stream of new products.

The traditional approach of firms has been quite the opposite, concentrating only on a single product at a time. This results in a bad organisation, as projects must compete against each other inside a firm for resources. Measurements for the budget, lifecycle and product breakeven are made on a single product focus. This results in lack of commonality, compatibility, standardisation and modularity. A solution for this situation, presented by Meyer and Lehnerd, is to turn the whole portfolio of business segment into a common platform and a stream of derivative products built on top of it. They presented a range of successful companies, like Black & Decker or Compaq, who have renewed their product portfolio this way. Although their approach concentrates on the own products of a company, the ideas of building a good platform can be adopted also to selling the platform itself (Meyer & Lehnerd, pp 1-2).

Final value to the user is realised only when the product is sold. Knowledge products resemble the packaged products. They do not transmit the information to users, but enable them to use knowledge in new ways (Lillrank & Forssen, p 15). In a similar way, a telecommunications platform does not necessarily give any applications to horizontal customers, but it improves their product development process with the basic functionality already implemented.

Kotler et al. say that there are three attributes for a product: quality, features and design. Quality is the ability of a product to perform its functions. It includes the overall durability, reliability, precision and the ease of operation and repair for the whole product or its services. These attributes of quality must always be viewed from the customer point of view; what level of quality they want and are willing to pay for. After the selected quality level is reached, it must be consistently kept there. Otherwise, the perceived quality of the product is not good enough anymore (Kotler et al., pp 566-567).

Vaattovaara (1999) talks about total perceived service quality, which is the difference between the expected and the experienced quality. The expected quality is derived from the promises of market communications or word of mouth as well as from the own needs of a customer. To the experienced quality image affects the technical (what) and the functional (why) quality of a service (Vaattovaara, p 9, originally Grönroos 1990). Quality is a part of the total value of the service perceived by the customer. Value can be defined with the following equation (Vaattovaara, 14, originally Heskell et al., 1997):

$$\text{Value} = \frac{\text{Results produced for the customer} + \text{quality of the process}}{\text{Price to the customer} + \text{costs of acquiring the service}} \qquad \textit{(Eq. 1)}$$

Features of a product are the attribute that is usually looked at most. They are the ones that are seen to differentiate the products from their competitors. One of the most challenging things for a company is to derive what are the right features that should be implemented into a product. It is seen that the best method is to do customer surveys and to find the information there. The customers can be asked what features are most important now and what would they like to have in the future. All features should then be evaluated based on what customers are willing to pay for them and compare it to the cost of producing them (Kotler et al., pp 567-569). For the case company, the fact that there are already vertical in-house customers helps in defining a part of the features that the customers want. Effective use of this information requires of course that it is adequately documented and managed.

The last attribute in the list of Kotler et al. is the product design. It means the look and feel of the product and the way how they affect to the functionality of the product. This attribute is usually connected to consumer products, but also industrial products have their design. Most industrial products are used by the experts of their field in the companies, and their judgement on how the product fits for its purpose, on its architectural elegance and on its ease of use should be thought in the design of a product (Kotler et al., pp 569-570).

Meyer and Lehnerd also talk about the elegance of a product design. It means that the product is easy to use and it serves its purpose without complications. According to Donald A. Norman, well-designed objects are easy to understand and interpret whereas poorly designed ones are frustrating and difficult to use. For engineers, elegance can mean both inner and outer qualities. They consider deeper issues like appropriateness of the selected components and the way that those work together, how they are integrated. For software products, engineers try to implement intuitive user interfaces, which make the simple tasks easy and give possibilities for users to do more complicated tasks (Meyer & Lehnerd, pp 82-83).

3.2 Productisation

3.2.1 Term definitions

A traditional view to productisation is that it is a function, which is done after the product development. It is seen as a distinguished part of the processes, separate from the product development. In rare cases, especially in service sector, the activities of a firm have developed to a level where the productisation is only trimming these activities into a superior product entity. Mostly still, the productisation approach should be in mind throughout the whole product development process (Vitikainen, p 22, Junttila, pp 18-19 and Sipilä 1996, pp 33-34).

Seppänen (2002) stated in his thesis that productisation is a phase in the product development process, where the cost and price structure is defined to a product or to a service and the necessary marketing activities are done (Seppänen, p 4). Also Vitikainen used a narrower approach as she stated that productisation is understood to include:

1. Market analysis,
2. Technical analysis,
3. The product development itself,
4. Organising the product training,
5. Producing the marketing material,
6. Creating an installation organisation.

In general she said that the productisation contains documentation of everything important related to the product (Vitikainen p 6, 22).

Mattila (2000) understood that productisation as a function starts from the product idea. After the product idea is accepted, it must be produced to meet the target market and its delivery must be planned. Before productisation, its benefits and risks must be identified. She refers to Sipilä (1996), whose definition for the professional services productisation is: "Defining, designing, developing, describing and producing a service provided for a customer in such a way that the customer benefits are maximal and the goals of a professional company are achieved". Vaattovaara states, that in the manufacturing industry, productisation is just seen as a part of the product development, where an innovative idea is made commercially viable (Vaattovaara p 22, Mattila p 20 and Sipilä 1996 p 36).

Productisation is a process where knowledge is packaged into sellable entities. It results in a product that is multipliable, protected, usable, individually defined and that is sellable. It helps in assuring the quality of service given for the customer. It is by its nature mostly standardisation and defining of the user interface and documentation of a product, protecting the product and implementing things related to distribution, customer support and product management. The level of productisation increases as the approach is moved from a tailored product via a customised to a pure mass product (see Figure 6). A company tries to form repeatable entities, whose cost per performance ratio is optimal and which are made as ready as possible to wait only for an order from a customer. Highly productised entities are complete customer solutions and require only few additional services (Junttila, pp 13-15 , Nybom, p 3 and Hoch et al, p 34).

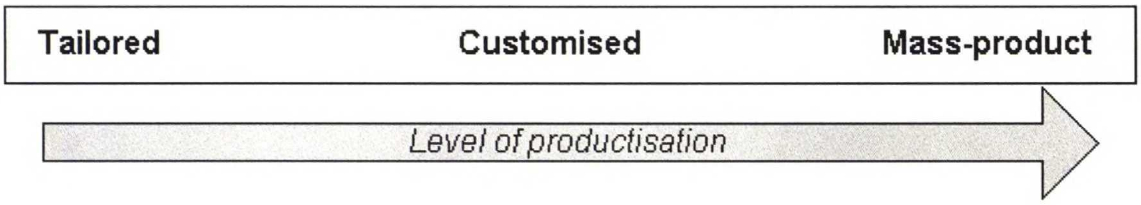

Figure 6: Level of productisation (Junttila, p 15).

Mattila (2000) refers to ISO standardisation criteria, where productisation is defined to be defining, clarifying, specifying and profiling the customer services that an organisation is producing. It is not just defining products, which can be goods or services, but clarifying the whole production of the customer service in to a more controllable form. Better product defining and clarification leads to the customer being more aware of what he or she is buying. In practical terms this means, for example, producing the product description and sales documentation (Mattila, p 21).

Productisation can be divided into external and internal parts. A customer sees only its external results, i.e. the products and their versions and all the supporting functions around it. Internally, productisation is a question of organising the functions more effectively and efficiently (Sipilä 1996, p 47).

3.2.2 Reasons and benefits of productisation

The task of productisation is to make a product easy to implement and sell. Productisation should be clearly in mind during the whole development process. Goals for different products are derived from the business strategy of a company (Vitikainen, pp 16-17, originally Voipio 1995).

Benefits from productisation to customer are clearly defined elements of a product, better quality assurance and clearer pricing. Reliability of the organisation is improved since the customer expectations and the quality are easier to measure in productised entities (Mattila, pp 22-23). In the case of this thesis, these same benefits come naturally also to the internal customers as the case company productises its telecommunications platform.

Productisation helps product pricing, as a price tag is easier to put on a well-defined entity. It must be decided, what is the price for additional or support services. In addition, the customised parts of the product are essential to be priced separately (Mattila, p 40). After all, in a productised product the customised parts generate most of the costs with a new customer.

A well-defined product makes comparison of alternatives easier for a customer. It helps the marketing department to promote a product and the sales department to get better margins from it. Customers trusts the product and the company more, when the goods or services are productised and they can be sure that they are not used as a pilot group for the product. Putting a fixed price tag for a product is easier than just for a set of goods or services. When the items are productised, they can be put also to a product catalogue, which in turn helps the nowadays important electronic business (Sipilä 1996, pp 19-20).

In an engineering company studied by Vaattovaara, the productisation of services created following benefits (Vaattovaara, p 8):

- Improved customer satisfaction as perceived quality of service is increased,
- Clarified marketing and sales activities,
- Improved efficiency and effectiveness of the service production activities,
- Improved manageability of operations.

Productisation decreases the development costs in a long time-scale and in most cases also increases the price asked for the product. That way, the margins of the product for the company increase. On the other hand, productisation helps the comparison of different products, which tightens the competition and can have the opposite effect (Sipilä 1996, pp 20-21).

Productisation leads to better-defined organisation and processes. It also forces to think through all the property rights inside a company in advance to avoid conflicts later on. Overall, according to Sipilä (1996), productisation can be the factor that puts the company to the so-called circle of success (see Figure 7).

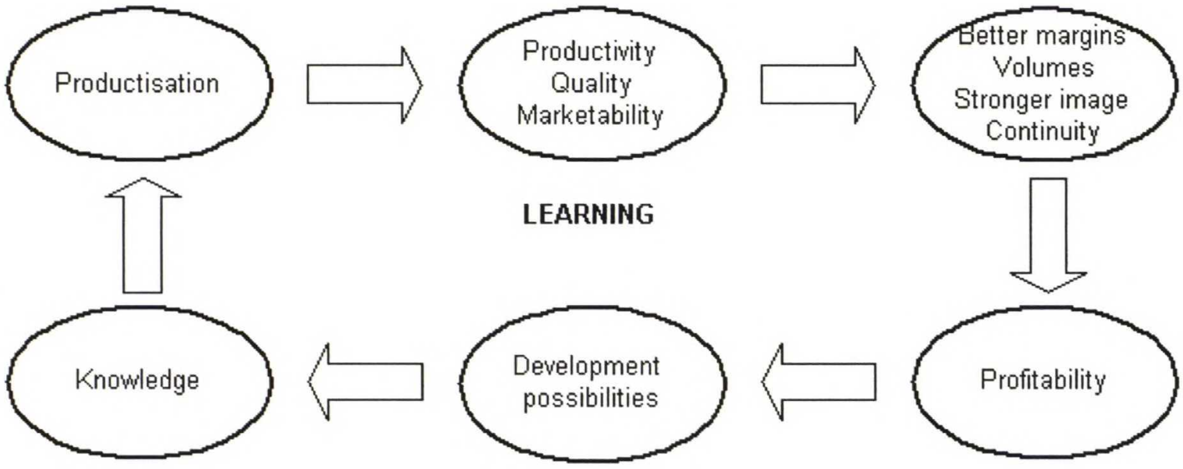

Figure 7: The circle of success (Sipilä 1996, p 22).

3.2.3 Product development

Product development is a function, whose purpose is to develop a new or an enhanced product. Development is a multistage process that contains tasks from finding information to the development of manufacturing processes. The task can also be developing an existing product to be better or to be cheaper to produce. It can also mean applying existing product to a new environment, where certain parts have to be redesigned (Jokinen, pp 9-10).

Jokinen (2001) divides product development work into four stages with decision points between them. The stages are start-up, sketching, development and finalisation. As a result for the start-up phase, there is a decision made of whether the development is started or not. After sketching, one draft for the product is selected, which is then improved in the development stage into a final construction suggestion. In the finalisation phase, details are checked and final documentation for the product is made (Jokinen, pp 14-17).

New product development decision requires always a customer need for the product and a possibility to implement it. Without these conditions, it is not reasonable to start a new development project. Sometimes these conditions appear by accident but in most cases through a systematic work (Jokinen, pp 17-18).

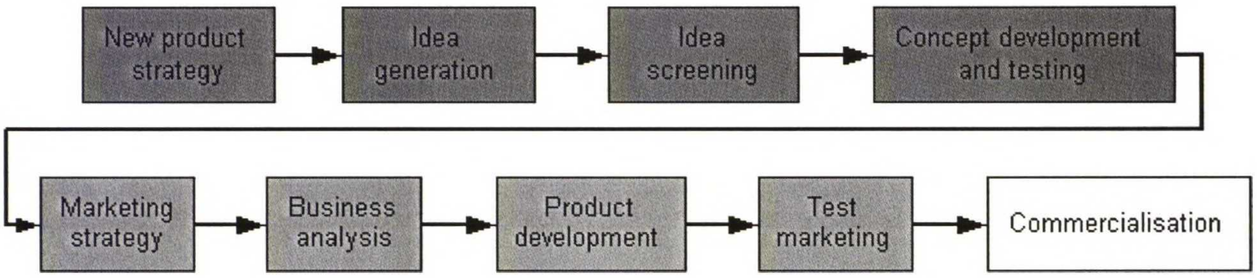

Figure 8: Steps in new-product development (Kotler et al., p 607).

For creating a successful new product, Kotler et al. present a new-product development process, which contains nine stages (see Figure 8 above). The process starts with defining a new product strategy, generating new product ideas and then screening them and dropping the poorer ones out. From the selected possible product ideas, a more detailed concept is developed and it is tested with target customers. After that, a marketing strategy and a business analysis are done before the actual product development. Before the full launch, test marketing is done in a more realistic environment (Kotler et al., pp 606-623).

Industrial products have a little different test marketing methods than consumer products. Industrial products can be promoted in distributor premises or the sales and marketing can be given a full advertising, sales and marketing support for a certain restricted geographical area. These are used to determine the customer reactions and they do not differ a lot from the consumer product promotion as such. Test marketing in this particular case is somewhat easier, as the case company has also internal customers, who can give the required feedback for the platform product (Kotler et al., pp 621-622).

Final stage in the list of Kotler et al. is commercialisation, which means introducing the product to the markets. Issues to be considered are timing, the place of launch, the target customer groups and the action plan on how to bring the product into markets. In this stage, full manufacturing power must also be in use to meet the expected demand. The PDMA term definitions define commercialisation to be a process of taking a new product from the development to the market. It generally includes production launch and ramp-up, developing marketing materials and programs, developing a supply chain and the sales channel, and developing the training as well as support and services (PDMA glossary).

Cooper and Edgett introduced a similar 13-stage process for new-service development. In addition to the above-mentioned issues, it included the development of the process in parallel with the system design and training of the own personnel before the launch. It can clearly be seen, that their model is created for the service sector since the process itself is actually the product that the service company is selling (Cooper & Edgett, p 30).

3.2.4 Productisation process

Many authors have described different kinds of productisation processes. Recently the term has mostly been used in the service sector. In technology and manufacturing driven businesses, the approach emphasises more that the productisation should be in mind during the whole product development process. Still, especially in the case of this thesis, similarities can be found from the service sector approach of packaging and commercialising the existing company internal knowledge or product.

Vaattovaara describes a four-stage productisation model for engineering services in his dissertation. The stages are (Vaattovaara, p 38):

1. Product screening,
2. Product analysis and concept construction,
3. Development of the service package,
4. Development of the service implementation processes.

In the first stage, the existing services are analysed and the parts to be productised are found out. It is not generating of new products, but conceptualising of the existing businesses or products. The second stage analyses the customer requirements, markets and competencies together with constructing the new product (Vaattovaara, p 38).

When developing the service package, the contents of the product are defined. The product is divided into modules to communicate the features to the customers and to help its implementation. This includes presenting the key value from the product or service for the customer. In this stage the necessary enabling services are defined as well as the not necessary but value adding facilitating services and the optional administrative services for managerial needs like billing (Vaattovaara, pp 38-39).

In the final stage, the service implementation processes are defined and the service components are derived. Those are the product bases, which describe the whole contents of the service, co-operation processed between the producer and the customer and finally a service production process, which defines the actions to be done in the provider-side of the service (Vaattovaara, p 39).

Lahtinen (1995) also introduced a productisation process for the service segment. It consists of four stages (Lahtinen, p 37):

1. Preparation work,
2. Forming the products,
3. Inside and outside marketing,
4. Follow-up.

Preparation work includes the actions from phases 1 and 2 in the model of Vaattovaara. It finds out the starting points and categorises current products and processes. The procedure includes studying literature and possible references from other similar organisations. It serves also the continuous improvement of processes and their quality. After these issues are clear, a plan is made on how to go through the process of moving the current pieces into a product (Lahtinen, p 37).

In forming the products -stage, different options of products are recognised and a product hierarchy is formed. Production costs of different alternatives are measured by analysing the value chain. It is also reasonable to evaluate if the products are effective for their purpose and what benefits do they generate. Quality requirements are also derived in this stage. The stage is similar to the phase 3 in Vaattovaara's model (Lahtinen, p 37).

The inside and outside marketing phase presents and informs about the products and operations to different stakeholders. Written documents and product descriptions are produced for this purpose. They present the contents of the product as well as its structure, price and quality features (Lahtinen, pp 37-38).

Follow-up and evaluation is needed to keep up the continuous improvement of products. First, there must be systems for making statistical reports from the products. These systems should then produce information about product success and effectiveness as well as inform about the changing customer requirements (Lahtinen, p 38).

Kitcho (1998) divides a product launch process in three stages: assessing the product of the company and the markets for it, developing a marketing strategy and planning and implementing the launch (Kitcho, p 16). In this thesis, the focus is in the first and third ones. The marketing strategy in horizontal markets is done by other stakeholders (see Section 6.2 for details).

Productisation should be started by defining the whole product concept, because it is not always clear, especially in software products. Issues to be defined are the purpose of the product for customers, interfaces to other systems and the required equipment for it. Besides these technical issues, also customer segments, pricing policy, maintenance and support as well as documentation and the future development should be thought (Vitikainen, pp 16-17, originally Voipio 1995).

Mattila (2000) summarized the phases and tasks in them for productisation model (see Figure 9). Her model consists of three stages: identification of knowledge, producing the product to the markets and delivering the product there (Mattila, p 52).

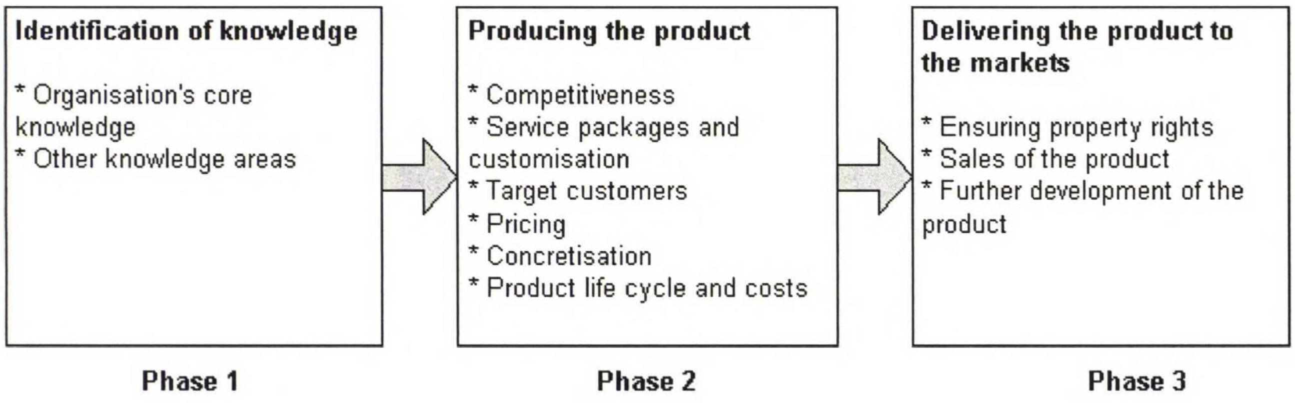
Identification of knowledge
* Organisation's core knowledge
* Other knowledge areas
Producing the product
* Competitiveness
* Service packages and customisation
* Target customers
* Pricing
* Concretisation
* Product life cycle and costs
Delivering the product to the markets
* Ensuring property rights
* Sales of the product
* Further development of the product
Phase 1
Phase 2
Phase 3

Figure 9: Summary of a productisation model (Mattila, p 52).

Mattila (2000) talks about concretisation of a product, which is a phase where the product gets its outfit. In service products, and in goods, this means, for example, making the product description documents for company internal use and the product brochures for marketing activities. A product description should describe the purpose of the product, its customer benefits, the customers and goals, competitors, price, the delivery principles and responsible persons for the product. A product brochure for outside use concretises the product in the customer point of view. It must be clear enough to point out the key features of the product and not promise anything that the product cannot deliver. The name of the product is also an essential part of the product brochure and the product description. The more abstract the product is, the more important is its name and therefore it should always be protected (Mattila, pp 41-42 and Sipilä 1996, p 94).

Choices for level of productisation vary. Only a part of the product can be productised, for example the documentation, or there can be a productisation-plan where the level of productisation is upgraded in phases. Besides the application area, the user group and openness of the system also affect to the level of productisation. Technically, openness means using standardised interfaces, which make it possible to transfer the product to different environments. Independence from specific suppliers or partners is avoided with this (Junttila, p 15).

3.2.5 Productisation plan

A productisation plan should be made right in the beginning of a product development project, same time with the project planning and requirement analysis. The productisation plan is meant for a common use of the management, marketing and product development. A goal is that only the development personnel have to read the technical documents and the requirements in technical sense are explained in the productisation plan. Productisation plans can be made in several levels, together they all should support the decisions made in the productisation process (Junttila, p 16).

A productisation plan should include the following (Junttila, pp 16-17):

1. Product strategy description,
2. Product description,
3. Arguments of the product existence,
    a. Customer need analysis,
    b. Competitor analysis,
    c. List of target market areas,
4. Internationalisation and localisation plan,
5. Life cycle description of the product and sub life cycles,
6. Goal market positions, goal price and selling predictions,
7. Product development work organising,
    a. Human resources and management,
    b. Subcontracting,
8. Distribution channels,
9. Documentation,
10. Quality goals and quality assurance,
11. Software product copyrights,
12. Services,
13. Product management,
14. Marketing,
15. Finance,
16. Appendices.

This is a quite comprehensive productisation plan structure. There are both business reasoning and technical arguments combined. A productisation plan is almost like a product description document, which is introduced by several authors.

Composing a product description document for customers is a key function in productisation. It forces the company to put on a paper concrete issues about the product and think it from the customer's point of view. Because the document is meant for the customers, it cannot be too long. Things must be said clearly and shortly. This procedure actually helps in clarifying the product properties for the company itself and therefore supports the product development (Sipilä 1996, p 98).

Sipilä states that a company should make the product description of a productised new item for internal use. This document can be used as a basis for product brochures that are used in marketing. Things that Sipilä includes in the description document are (Sipilä 1996, pp 74-77):

- Product's name and general description,
- Product's purpose and benefits for a customer,
- Market potential, customers and goals,
- Competitors and substitutes,
- Fitting to company's strategy,
- Product description (core features, maximal model of the product),
- Versions and variants of the product,
- Concretisation of the product (how a design is put into a sellable item),
- Customer or project references,
- Price,
- Delivery times,
- Responsible persons,
- Further actions of productisation and product development,
- Effects on processes.

Kitcho also mentions the importance of a product description document. Before the launch of a product, there must be a clarified description in order to market it. Additions to the Sipilä's list above are (Kitcho, pp 23-30):

- Compatibility with industry standards,
- Information about first beta customers and description for their testing of the product,
- Rules for sales support, like installation and customisation consultation,
- Description of post-sales support, like installation, training or communications,
- Description about the product training given for the sales personnel and for the customers, including the training materials to be produced.

Defining and reserving a good product name is essential, when putting a new product to the market. If the product satisfies some customer need well, it probably will face competition. That is why product names must be reserved in early stage of the process. It is also a question of what the customer gets when it buys a right to use or own the product. The product must be clearly identifiable (Sipilä 1996, p 94).

3.2.6 Documentation

There are lots of references to documentation when productisation is introduced in the literature. In fact in the service sector the whole productisation is about documenting the knowledge and working processes of a company. Also in this thesis the task is to package the existing knowledge and product of the case company and to add some services on top of it in order to sell it to other companies. A major part of this process is the documentation. For these reasons, documentation related to productisation is introduced more detailed here.

The deeper the level of productisation is, the higher is the requirement for the documentation quality. This is because a higher level of productisation usually means mass production and more customers, and personal guidance cannot be given to them all. Documentation regarding installation procedures of a product is very important, because at least in mass products, the user must be able to install the software by itself without difficulties. The installation documentation should include also information about the hardware and licensing constraints (Junttila, pp 61-63).

Haikala and Märijärvi divide the documents in three categories: quality process related documents, project documents and product documents. They mention four kinds of product documents: user guides, installation and operation guides, training material and technical documentation. With technical documentation, they understand all functional specifications, technical specifications and the information about testing and product management (Haikala & Märijärvi, pp 59-63). The latter is usually company internal documentation and may not concern the customers at all.

Probably the most obvious form of documentation is the user guide. User guides should contain easy-to-understand information that informs the user about normal and abnormal behaviour of the system. They should have a logical structure that describes the steps of usage from the start-up through normal functionality to the closing of the system. Junttila states that the product developers should not be the ones that produce the documentation completely. This scarce resource should be saved and use a special documentation group that finalises the documentation based on the draft documents made by the developers. With this kind of approach, too complex technical document style is avoided from the final versions. To make it even clearer, some pilot users can participate the documentation process (Junttila, pp 63-65).

Training material is also a part of the product documentation. It can have different target groups like trainers in the company, trainers in the customer organisation or in distribution channel and the trainees in these organisations respectively. Most difficult is to make the self-study material, because in a self-study, there is no trainer to explain the unclear things (Junttila, pp 65-66).

Maintenance and customer support material is also challenging, because the place where support is given can vary from a separate maintenance unit or a product development organisation to a combination of them (Junttila, p 66).

All the documents should have a common form. Their look should be the same. Every document should contain following information (Haikala & Märijärvi, p 64):

- Name of the organisation,
- Name of the project,
- Name of the document,
- Version number and version history of the document,
- Number of pages in the document,
- Editors of the document,
- Inspectors and the approver of the document,
- Current state of the document (draft, requested for approval, approved)
- Table of contents.

3.3 Product management and services

3.3.1 Configuration control and release policy

Software products consist of several components, which are integrated to larger entities, configurations. These components are, for example, source and binary code files, specification documents, test plans, test modules and user guides. As components evolve during their lifecycles, new versions of them must be made. Evolution happens because errors are found and corrected and new requirements are introduced to the components. Versions of the components can be parts of different configurations. That is why an efficient configuration management must be in place. Haikala and Märijärvi divide the configuration management into three segments: managing the versions of components, managing the versions of configurations and the mode of operation used in handling the two former ones (Haikala & Märijärvi, pp 237-238).

Configurations can version for different reasons, also others than error fixes or new feature requirements. There can be, for example, different kinds of usage environments, the product can be sold in smaller pieces to some customers or there can be customer specific customisations in the product. Haikala and Märijärvi state that after the initial version of a component is produced in a project, all changes to it require an approval, for example from a project meeting. A formal process that they suggest is that after an error report or a change suggestion is received, it should first be approved. Only after that it can be implemented and after testing, the change should be approved again (Haikala & Märijärvi, pp 243-246 and Junttila, p 89).

A company must have defined a policy, according to which new features and corrections are introduced and delivered to the customers. A company can have a product release design document for defining which corrections and new functionalities are taken in to the coming releases and setting the dates for the releases (Junttila, p 89). A system release is a version of the product that is delivered to customers. It contains some new functionality or is an adaptive version for a new environment. A release is more than just a configuration of program blocks of the system. It includes also (Sommerville, p 684):

1. Configuration files for configuring the new release to the target environment,
2. Changed data files of the system release,
3. An installation program for the release,
4. New version of the documentation of the system.

There are different kinds of releases. Enhanced releases contain new functionality and they are delivered to all customers. Between them, repair releases can be introduced to fix problems in previous enhanced releases. Between customer releases, a company can build its own internal version for testing purposes, for example. In Figure 10, this kind of a release strategy is described (Sommerville, p 686-687).

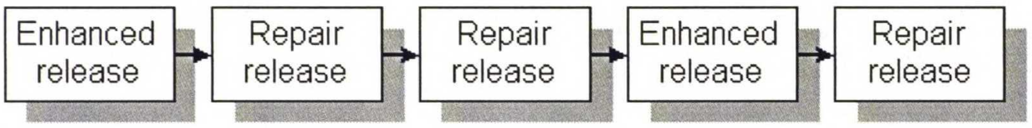

Figure 10: System release strategy (Sommerville, p 687).

Studies in the software industry have shown that every enhanced release that introduces new functionality contains also new errors. Therefore, a company should not deliver new enhanced releases too often. Due to the repair and internal releases between them, the rate of new enhanced releases should be approximately one release a year. To avoid further problems from a single error, the introduced enhanced releases must contain also the error corrections of the repair releases made between the new release and the previous enhanced release (Sommerville, p 687-688).

Release management becomes more complicated when not all customers want the new releases as often as the producer introduces them. The situation depends on which kind of an enhancement it is in question, how expensive it is to install a new version to the system and on which kind of a contract the customer has with the producer (Sommerville, p 687). Different kinds of approaches are, for example:

The selected release policy must be communicated to the customers.

- All corrections include to the initial price for a certain release for a fixed period.
- Error correction patches are included, but enhanced releases are priced separately.
- All new releases of the system are priced separately.

The configuration management database needs to contain information about which customers a new version is delivered to. It also has to contain information about the hardware and operating system requirements for a particular delivery, the different managed versions of the configuration, versions containing a certain component and the number of change requests and error reports that are open on a certain component (Sommerville, p 679).

3.3.2 Customisation

The level of customisation for a product is a strategic decision. A service product can be productised so that it contains standard parts, module parts and customised parts. The standard part is the same for every customer, from module parts a customer can pick what he or she wants and the customised part is made separately for each customer (Mattila, pp 37-38).

There is a conflict between the expectations of customers to get tailored products and service but to pay only a price of standardised, stripped-down mass-product components. When trying to fulfil these expectations, companies that offer total service solutions are in the most difficult situation. Their competitors are most likely adapters of their business idea and offer a poorer customer support, for example. It is a strategic choice of the company into which group it wants to belong and it naturally affects its pricing policy. Selling a product entity or a complete service system is by no means more glorious than selling individual products; it is just a question of different product strategies (Sipilä 1996, pp 65-66).

Mass-customisation means mass-production of specially customised products for individual customers. Formerly, tailoring of the industry products meant that only the largest customers could afford to invest so much that they got exactly the product they wanted. In a high quality product, the customers feel that they get even more than they have asked. They can even think that the product is tailored for them only, although it is just a unique set of standard components. Modularity is seen as the only way to gain the mass-customisation effect (O'Grady, pp 81-85 and Nybom, p 10).

Some level of customisation is needed, because each customer is different. This can easily lead to a situation where the extra costs of customisation outweigh the increased revenue from it. With a larger customer base, it also makes the processes very complex. A general trend is to be somewhere between pure standardisation and customisation. This approach has the benefits of smaller process changes but still with a reasonable product variety (Vaattovaara, p 17).

Heavy standardisation of products leads to a mass-production type of operation as well as to routine-like specialised jobs and decision-making processes. Mass customisation takes the advantage of standardised basic product platforms and processes to meet the highly varying customer requirements (Vaattovaara, p 18). There should not actually be very much distinction between customisation and productisation. In fact, well-productised base products make customisation easier, because the basic things do not have to be designed all over again for each customer (Sipilä 1996, p 16).

A configurable product is a product family that consists of multiple products called variants. According to Tiihonen et al., many firms are nowadays interested in configurable products. However, the way they produce them is still based on individual orders from the customers according to which the products are tailored case-by-case. There is obviously a need for tools to define configurations (Tiihonen I, p 2). Modifications to the basic products can be made by (Clarke 1997):

- Adjusting parameters like colour or size,
- Making some extra features optional,
- Parameterisation of the number of some features, for example number of e-mail accounts in an internet connection service,
- Specialisation, for example by putting a special label to some products,
- Custom extensions,
- Custom modifications, where some elements of product are specialised for a certain customer,
- Supplementary services beside the basic product.

Configuration can be defined as a task of designing a product using a set of pre-defined components while taking into account a set of restrictions on how those components can be combined. Two kinds of configuration work can be identified: the product configuration in a traditional sales-order-delivery process or a so-called configuration design, which takes place already in the product development phase (Tiihonen II, p 2).

3.3.3 Product services

Offering the support and additional services is done for differentiating from the competitors and increasing the sales (Mattila, p 33). According to Kotler et al., the product-support services are a part of the product strategy and they can even be a major part of the total product offering. They are increasingly used as the augmented part of the product that offers competitive advantage to a company (Kotler et al., pp 585-587).

Services to a final customer depend on whether the customer is in direct contact to the company, or via an external distributor. Services included as a part of the product must be clearly documented in the contracts. A basic service is to provide the end users a channel for giving feedback and reporting problems. Regardless of the structure of the distribution organisation, its task is to give this service to the customers. Services for final customers can include (Junttila, p 81):

- Web pages including guides and for example FAQs,
- Contact information for presenting questions,
- Fault management,
- Consultation for special situations or special functionalities,
- Training and training material.

Besides the previous ones, Kotler et al. include also credit and financial services, fast and reliable delivery, installation and after-sales services and repair to the customer product services (Kotler et al., p 587). Services for the distribution organisation depend on whether it is inside or outside the company. The purpose of the services for the distribution organisation is, however, to help it to sell the product to the final customers. Services for a distribution organisation can include (Junttila, pp 81-83):

- Internal web pages including guides and for example FAQs,
- Contact persons,
- Training services (demos, usage examples, training material, training),
- Material and tools related to the product (brochures, guides, pricelists, demos, press releases),
- Product development schedule,
- Functional and technical descriptions of the product,
- Delivery guidance,
- Sales seminars and training days,
- Customer feedback and logistics organisation.

Vitikainen defines the supportive functions to be merely different documentation related to installation, services, training and maintenance. As said earlier, the level of documentation depends on the level of customisation of the product. The level of documentation has two dimensions, formal issues and content issues. The demand on both of these increases as target group goes outwards from the company and as the type of documentation goes from supportive to independent documentation. The document types in Vitikainen's definitions are technical descriptions, installation manuals, user manuals and managerial manuals for maintenance personnel. In delivering the documents to the distribution organisation, Internet is a useful media (Vitikainen, pp 56-61).

Services given to a distribution organisation must also be properly charged. This can be difficult especially if it belongs to the same company with the product development organisation. Pricing should be cheap enough to ensure, for example, that the sales personnel attends trainings and thereby gains the latest information of the product to effectively sell it (Junttila, pp 83-84).

Customer training and marketing are even more important in developing products as they consist more and more of a core product and its supportive services (Vitikainen, p 31, originally Huomo 1993). When analysing new product ideas, also training that it requires should be studied. Level of customer training depends on the level of customisation of the product. With mass products, more emphasis is put on the documentation and self-study material whereas with customised products user training might be a better alternative (Vitikainen, p 49). Training given to the distribution organisation can vary a lot. The level of training depends on the technical complexity of the product. It also depends on the amount of direct user training given by the producer organisation (Vitikainen, p 32 and Junttila, p 83).

Software maintenance services contain three different types: corrective maintenance for error fixes that are discovered in testing or presented in the error reports made by the users, complementary maintenance for implementing new features requested by the customers and adaptive maintenance for tailoring the system to be used in a new hardware platform (Sommerville, pp 660-661).

A goal for organising a customer support organisation is that it burdens as little as possible the actual product development personnel. This requires an organisation with good processes for taking problem reports in and translating them into an easy format for the product developers to solve. There is a model of a four-layer customer support organisation presented in Table 1 below (Junttila pp 81-82, originally Mäkinen 1994).

Table 1: Tasks of the layered customer support model.

Layer 1	The customer contacts trough email or web pages to local distribution channel. Problems should be able to be solved by training and other support documentation material.
Layer 2	The customer contacts via phone, which can solve slightly harder problems than Layer 1.
Layer 3	Problems that are more difficult are solved by experienced designers, technical support or a specialist in distributor's organisation.
Layer 4	Topmost support organisation solves all the toughest problems and is the only one that can take direct contact to product development.

Layered customer support model should be invisible to customers. They should not know that there are layers and should not notice when their request is moving from one layer to another (Junttila, pp 82-83).

3.3.4 Product management organisation

To manage a modular product, a company must have a product management function. Besides information of the product functionality, it shall manage information about the markets, quality, tools, deliveries, problems and changes of the product (Junttila, p 85).

Kotler et al. suggest that a special multidisciplinary team is established to take care of the actions related to the new-product development process. It must know the goals of the top management and have enough resources in use. It should work in parallel with the actual product development organisation (Kotler et al., p 625). A product team should contain representatives from planning, design, production, sales, distribution and evaluation. The information shared within this team contains (Meyer & Lehnerd, pp 138-139):

- Results of market studies,
- Details of competing products, even on a subsystem basis,
- Platform strategy,
- Engineering designs for key subsystems and interfaces in next platform,
- Schedules,
- Project budget, sales projections and resource needs,
- Product family rollout plan,
- Company executives' comments,
- Unresolved questions.

3.3.5 Sales support

There are many different ways to deliver the products to customers. The channel can involve several different kinds of organisations. There can be, for example, direct sales force of the company, resellers, developers, system integrators, distributors or original equipment manufacturers (OEM). A general categorisation model is to divide the channels to direct and indirect ones. In the direct model, usually with industry or tailored products, own representatives of a company sell the product to the final customers. In the indirect model with mass products, selling is done by external organisations that form the distribution channel (Kitcho, pp 77-78 and Junttila, p 26).

Before a new product launch there is a challenge to get the right information to the sales channel early enough. The channel must get the marketing material well in advance. They also need training and selling tools for the product. In one sense, a company first has to sell the product to the distribution channel and to provide it with tools to sell the product to the final customer (Kitcho, pp 78-79). The R&D organisation must continue this sales support throughout the whole life cycle of the product.

In the direct selling mode, the R&D must understand how the sales organisation inside the company works, which are the contact points and what information they need and how they need it. Marketing material for a direct sales force is different from the one given to customers. Direct sales force can be given company confidential information about the product and information about the strategic objectives of the company. They are also given product reference guides and product configuration tools and guides. Training related to the product is also provided for them (Kitcho, pp 79-82).

Junttila states that there can be separate versions of the user guides for the final users and for the administrators. Materials for a reseller include brochures, demos, slide sets, product description material, product catalogues, pricing lists and newsletters. The same applies to the training material, which can include different information for the trainers than for the trainees (Junttila, p 61).

Indirect channel can mean several things. It can be a value-added reseller or a system integrator, who just uses the product as a part of its own offering. Developers create software applications on top of the platform provided by the company and they use different software tools to do that. They then sell an integrated hardware and software product or just a software product onwards. OEMs use the provided products as parts of bigger complex products that they sell to their own customers. Common for all of these is that their focus is not on the product of the supplier company, but on their own. This means that a company must market the product to them, not to their final customer (Kitcho, pp 82-83).

A company should get the indirect channel to buy its product instead of a product made by a competitor. There are some things to be considered about indirect channels (Kitcho, pp 83-87):

- Determine the business model of the channel and thereby its needs for the product. The channel is primarily trying to fulfil the needs of its customer. Requirements for the supplier product reflect that fact.
- Business relationship and contract agreements with indirect channel must be defined.
- Defining what kind of product information and training the channel needs and how it is given to it. This material is most likely different from the direct channel's material, considering for example technical details and roadmaps.

3.4 Summary of the productisation models

Different authors included several different things to a productisation process in Section 3.2. To sum it up, all of them see the process to contain the following four stages:

- Preparation work that includes determining the customer needs, business reasoning and competitor analysis,
- The actual product development and building work, which includes also documentation and service creation,
- Marketing activities and implementing a product management organisation,
- Delivering the final product, follow-up work, maintenance and further development of the product.

The two points in middle of the previous list define the core productisation process. It includes following activities:

- Creating product descriptions and clear user documentation,
- Creating documentation for further developers,
- Defining modularisation and versions of the product, also from internationalisation and localisation point of view,
- Defining the quality goals of the product, implementing quality assurance and planning of further development,
- Testing existing product and its boundary values and performing integration testing,
- Creating copyright systems and license terms for a product,
- Creating services organisations and systems for:
    - Receiving problem reports,
    - Distributing updates of the product,
    - Training of users.
- Creating a product management organisation for handling the product configurations, requirements and services related to the product.

The issues presented here are studied from a couple of product offerings in telecommunications industry in Chapter 5. In Chapter 6, the telecommunications platform of the case company is introduced and suggestions are given in Chapter 7 on what should be done to it in the productisation project that has been started in the case organisation.

4. PLATFORM AND SOFTWARE PRODUCTS

4.1 Platforms for products

4.1.1 Effective application development

For many communication and collaboration functions, a shared infrastructure is needed. Delays, interruptions and errors in functions cost the parties involved lots of money. Wainewright lists that an effective infrastructure must have many supporting applications, multiple providers and essential standards (Wainewright, pp 6-7).

For a company to succeed with a software platform, it needs to attract enough application developers to use its platform. It remains as a challenge to manage the partner web. There must be some kind of a uniting force as an incentive to join the web of partners. This can mean, for example, a good enough technical support or providing updates. In addition, some companies have found their success on establishing developer communities where developers can put their components for public use and similarly use components made by others (Hoch et al., pp 196-197).

The centre of a web is usually a common technology platform. There are certain benefits for the developers related to it. First, a platform offers standardised APIs for others to use and therefore makes their work easier. Secondly, building such a platform is a huge effort, which saves time from the application developers (Hoch et al., p 199)

The more there are partners already, the more likely a platform will attract new ones to join. Similarly, the more there are developers, the more there are applications, and the more there are end-users willing to buy the basic product. As more customers come in, the more developers want to get a piece of their money. Still, for a kick-start of a community, a good basic platform must be available (Hoch et al., p 197-198).

Traditionally, the term 'platform' in the software industry has meant the special hardware and later on the operating system that an application is designed to run in. Before the de-facto operating system standards, i.e. Microsoft Windows and different UNIX versions, the applications used to be tailored for different hardware 'platforms' like Intel, Macintosh or for special proprietary minicomputers and mainframes. Spread of the Windows and versions of UNIX freed the application software companies from this tailoring and allowed them to do only one product. The software then ran on different hardware, thanks to the common operating systems (Meyer & Lehnerd, pp 174-175). Same development is ongoing now in telecommunications and other sectors also. The application providers can sell their solutions to many vendors and operators if their technological base is the same.

A consequence from the separation of the platform development from add-in modules, i.e. the derivative products, is that the platform team can evolve to be a company inside a company. Meyer and Lehnerd presented examples that confirm this to be a successful managerial development. A platform product team continuously seeks for better designs and enhances its subsystems. It can also do that by looking for modules made by other companies (Meyer & Lehnerd, pp 199-200). This of course requires that the interfaces between product components are clear.

4.1.2 Strategic viewpoints for platforms

Long-term success of a company does not depend on a single product, but it requires a continuous stream of value-rich products. Such products form a product family. They share a common technology and address similar market applications. This common core technology part is said to be a product platform. Meyer and Lehnerd define a product platform to be a set of subsystems and interfaces that form a common structure and from which a stream of derivative products can be efficiently developed and produced. The term efficiently refers to the cost reductions that are achieved in production and procurement of components and materials. A more important thing is that the building blocks of these platforms can be rapidly integrated with new components to address new market opportunities (Meyer & Lehnerd, pp xi-xii). PDMA defines the term 'product platform' to represent the underlying structures or basic architectures that are common across a group of products or that will be the basis of a series of products commercialised over a number of years. Similarly, it defines the opposite phrase, 'a platform product' to be the design and components that are shared by a set of products in a product family and numerous derivative products can be designed from this platform (PDMA glossary).

Cooper and Edgett see platform as an enabler. Building or buying a platform is investing in capability, hardware, software or technology that leads to development of commercial deliverables. Those can be either new, improved or enhanced entities. Characteristics for a platform are that it is more expensive to develop a platform than individual products or services and that it has a broader impact on the business eventually. Projects or products that are derived from a platform are not yet thoroughly known when the platform is designed and it creates a ground for various kinds of projects and products to be developed (Cooper & Edgett, p 141).

The original idea of a platform has been very physical product -based. As many industries have moved towards platform -type of thinking, it is nowadays referred to include also technological and operational capabilities. The analogy to, for example, oil-drilling platforms is still in place. The initial cost of building it is high, but then it can be used to drill a number of holes fast and at low cost. According to Cooper and Edgett, a chief executive of a major telecommunications vendor has said that in a few years the product development will be just turning on a switch in a platform to create a new service (Cooper & Edgett, p 140).

The real power of product platforms lies in the fact that they provide leverage to the product creation process if used properly. According to Meyer and Lehnerd, costs of developing a derivative product include only incremental costs and the investment to the platform, since the development work of the platform can be seen as a sunk cost. This is especially true if a company buys its platform from another vendor like is the idea in the horizontal markets. Typically, the incremental costs are only a fraction of the total costs spent in the platform development (Meyer & Lehnerd, p 41).

Taking into account the above-mentioned issues, there are three important things in platforms. First, decisions concerning platforms are of greater importance than the ones concerning single products, since their impact is more far sighted. Second, the usage and investing for a platform should be justified by knowing a couple of the first derivative products in detail, although there will probably be more of them. In other words, by showing the benefits from a platform for the first derivative products, it is easier to justify the investigation to the platform and to build a better design for it from the beginning. According to Natis (2002), SUN's strategy is to gain leadership in server platforms by offering the markets a set of applications built on its own platforms (Natis, p 6). Third, a company must recognise what kind of a platform it is building. Is it a platform that benefits the customers directly as a platform, indirectly via the derivative products or just some company internal processes, like a new customer information system for sales personnel (Cooper & Edgett, pp 142-143). In the case organisation, derivative products are already planned and developed based on the platform, so the second point of showing the benefits through derivative products should be straightforward.

4.1.3 Platform architecture and modularity

A platform consists of common components, modules or parts and their interfaces. Key parts represent major subsystems of the product platform and interfaces usually refer to the connections between the subsystems rather than individual components (Meyer & Lehnerd, p 7).

A very common approach and a useful model in building a platform is to have an as large standard module as possible and on top of that some modular parts and a few tailored parts for a customer. A customer sees the product from the tailored part point of view, and therefore thinks it to be more customised than it really is. By minimising the tailored part, the product development and delivery times as well as costs can be reduced (Sipilä 1996, p 69). Completely pre-defined product packages are difficult in pricing sense. There is always a risk that the customer wants some extra service afterwards and that cannot be charged properly (Lahtinen, p 36).

Potential improvements achieved through modularity are the reduced development times of products, increased product variety, reduced capital requirements for the new-product development and the increased freedom of design and component change (O'Grady, pp 3-4, 26-27). These benefits are applicable for both the producer and the buyer of a platform.

Modularity is based on assembling predefined modules into a product with some configuration information. Modularity requires knowledge on what modules can be integrated together. This requires that the product architecture is defined as well as the role of each component in it. After determining the roles and the module interfaces, the possible configurations can be described (O'Grady, p 18).

With a conventional product development cycle, the whole process has to be done every time from the scratch, which increases the costs enormously when the product variety increases. With modularity, the product variety is achieved through using different combinations of modules. Number of possible product configurations can be calculated with following equation (O'Grady, pp 95-97):

$$\textit{Number of configurations} = \prod_i y_i \quad \text{,where } y_i \text{ is the number of variants in module i.} \qquad \textbf{\textit{(Eq. 2)}}$$

Of course, the possible number of configurations must take into account the constraints between modules, i.e. their positive or negative dependencies from each other (O'Grady, p 97).

Modularity can be applied to a product in numerous ways (Sipilä 1996, p 69):

- Core part and customer-selected extra parts,
- An entity that is composed of compatible parts,
- Maximal model, that the customer can strip down,
- Basic model, that a customer can change to another version,
- A basic combination, which includes continuous maintenance and update service.

In many occurrences, a concept of subsystem is introduced. Subsystems are well-isolated entities encapsulating a set of functionalities (see Figure 11). A general requirement for subsystems is that it is possible to change their implementation completely without any influence on other subsystems. Also changing another subsystem should not have influence on this particular subsystem. From these requirements, the following interface requirements can be defined for subsystems (MITA, p 39):

- It provides a support interface for upper layers using this subsystem.
- It contains an adaptation interface for using lower level functions.
- It has an optional protocol interface for communicating with other subsystems in the same level.

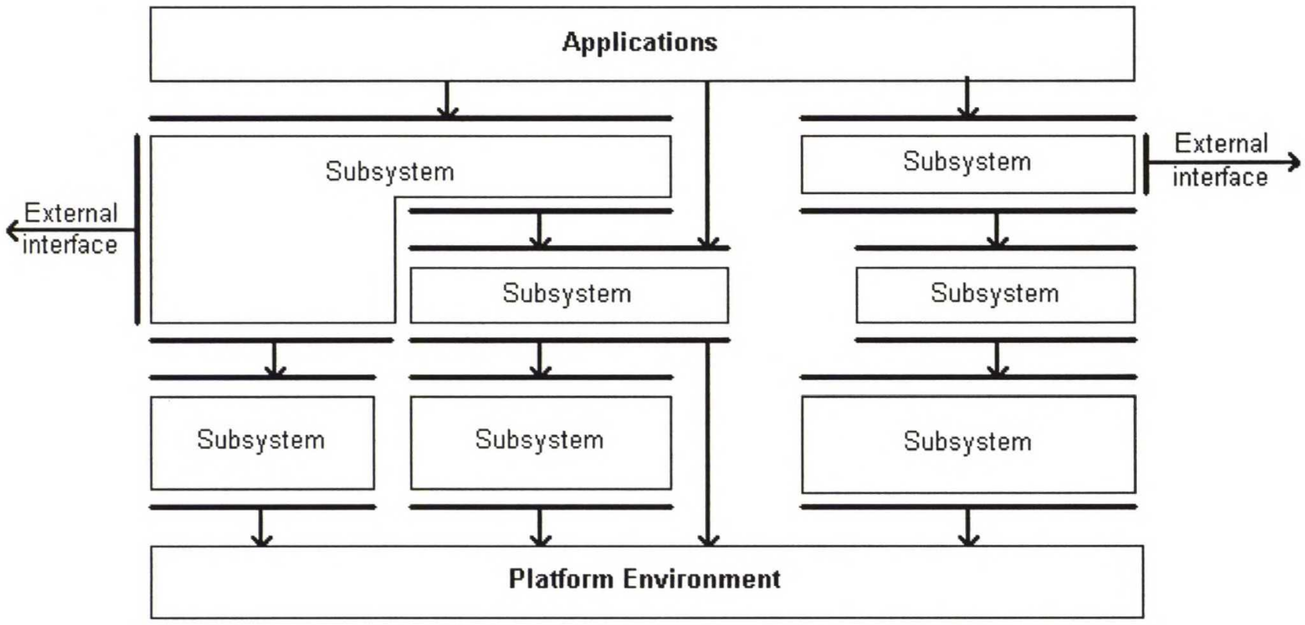

Figure 11: Modular element with subsystems and their interfaces (MITA, p 41).

Each subsystem in a product has a specific task and by combining them a higher level functionality is achieved. Therefore, Meyer and Lehnerd suggest that subsystems may also depend on each other in a way that changes in one subsystem require changes also to some others (Meyer & Lehnerd, p 40). This is an opposite viewpoint compared to the approach that the dependencies between modules should be minimal.

However, a company must be careful for not to use too many modules. A big number of modules gives a possibility to variety in products, but creates also more connections between modules and costs of integration to the company (O'Grady, pp 154-155). A big amount of possible configurations makes the maintenance of the system harder, which of course increases costs. For that reason, a well-defined basic part and a reasonable amount of alternative parts should be implemented (Vainio, pp 38-43).

4.1.4 Building a platform

Successful, robust platform products do not happen by accident. They must be managed and renewed all the time, in order to avoid the derivative products to be out of date. Literature in the platform area concentrates on manufacturing of physical products, but as Meyer and Lehnerd stated, physical and non-physical products share a number of similar conceptual elements when thinking of building a platform (Meyer & Lehnerd, p xii).

The common building blocks under a platform are (Meyer & Lehnerd, p 44):

1. Insights into the needs and processes of customers and competitive research and development to uncover those needs.
2. Product technologies in components, materials, subsystem interfaces and tools.
3. Manufacturing processes and technologies for competitive production.
4. Organisational capabilities for distribution, for customer support and for information systems used for control and to collect market feedback.

Manufacturing technologies are mostly related to the industries where the product is seen as a physical entity. They can be as critical as the chosen technology of the components. In process industry, the process itself actually is the product platform (Meyer & Lehnerd, p 47). By simplifying, it can be said that the production process and methods used to build a platform have also a great significance to its success. At least the process should not prevent the usage and building of the chosen technology and product types.

Organisational capabilities are the common building blocks related to distribution, customer support and information systems. Meyer and Lehnerd see these capabilities as building blocks for derivative products, which are included in the platform already. In any case, products must have an established distribution channel, either an own one of the company one established with its partners (Meyer & Lehnerd, p 48). One must keep in mind that the product can be, for example, a software build, which is distributed in an electronic form.

Meyer and Lehnerd suggest that managing of a product family evolution should be made in such an incremental model, where first derivative products are started to be build at the same time with the platform. Increments are then done first to the platform and then taken into the applications. In this kind of a model, concurrently one team is building new add-in modules and another is making enhancements to the platform. This way also the transition from a technological environment to a new one is handled by the platform team and application developers do not have to worry about it. The last point mentioned is one of the greatest challenges for the software companies in general (Meyer & Lehnerd, pp 191-193).

To accomplish the platform development task, there are five key steps to be taken (Meyer & Lehnerd, pp 235-236):

1. Define the platform strategy.
2. Determine the core building blocks of the new platform.
3. Build a composite design of the platform and attach key building blocks to it.
4. Develop a rollout plan for platform enhancements and for the derivative products.
5. Organise and empower the teams that will implement the platform and its derivative products.

One must note that in this horizontal business case there are no derivative products sold through the organisation unit in question, although the case company in wider sense produces them also.

Meyer and Lehnerd presented the Hewlett Packard DeskJet-printer platform development. It included a product family map. The map presents all the branches of the platform and the derivative products made from it. The platform itself must also be developed and enhanced. Hewlett Packard had made the enhancements and the development of a completely new version of the platform concurrently with the present platform and its derivative products. The enhancements included small feature extensions to the current platform without disturbing the existing functionalities (Meyer & Lehnerd, pp 28-29). Lessons from the Hewlett Packard strategy are (Meyer & Lehnerd, p 34):

1. Create derivative products from platform.
2. Enhance current platform to address new markets or to reduce costs.
3. Develop a completely new platform.
4. Do all above-mentioned three things at the same time.

Performance of a platform should be monitored. Company must gather data for it on a product family basis. A simple spreadsheet or database for performance analysis purposes should include knowledge on (Meyer & Lehnerd, pp 149-151):

- Engineering costs,
- Development times,
- Manufacturing costs,
- Market development costs,
- Sales data,
- Margins.

The platform performance depends largely on the success of the derivative products based on it. A simple formula for measuring the average platform efficiency is (Meyer & Lehnerd, pp 152-153):

$$\text{Efficiency} = \frac{\text{Average derivative product engineering costs}}{\text{Platform engineering costs}} \qquad \textbf{(Eq. 3)}$$

Efficiency here refers to the costs allocated for the products. In horizontal markets, this efficiency can have a tremendous meaning in marketing. A study of electronics firms made by Meyer and Lehnerd showed them that an efficiency value below 0.10 indicates a highly leveragable platform (Meyer & Lehnerd, p 156).

Besides costs, also the development times of derivative products are things that are supposed to become lower when efficient platforms are used. It takes probably a little longer time to build the initial platform than the first end product, but the next ones should be able to be developed in a lot shorter time. The cycle time efficiency for a platform can be expressed as (Meyer & Lehnerd, pp 159-160):

$$\textit{Cycle time efficiency} = \frac{\textit{Time to develop a derivative product}}{\textit{Time to develop the platform}} \qquad \textbf{(Eq. 4)}$$

Although the task of a product platform is to offer a common ground for the derivative product development, they cannot be left frozen in any stage. Instead, the platforms must be evolutionary entities. They must be developed further and managed in the way that new technologies are taken into the platform and to the processes used in developing them. The evolution of a platform can be seen to follow a three-stage model. First, there is the initial platform architecture with the initial subsystems and interfaces between them. First derivative products are based on this platform. In the next stage, some subsystems have been developed further and changed to newer versions, but the architecture remains the same. Some new derivative products can be built on this platform extension. A completely new product platform is introduced when a number of subsystems and interfaces between them are reordered, i.e. the architecture of the platform is changed (see Figure 12, Meyer & Lehnerd, pp 41-43).

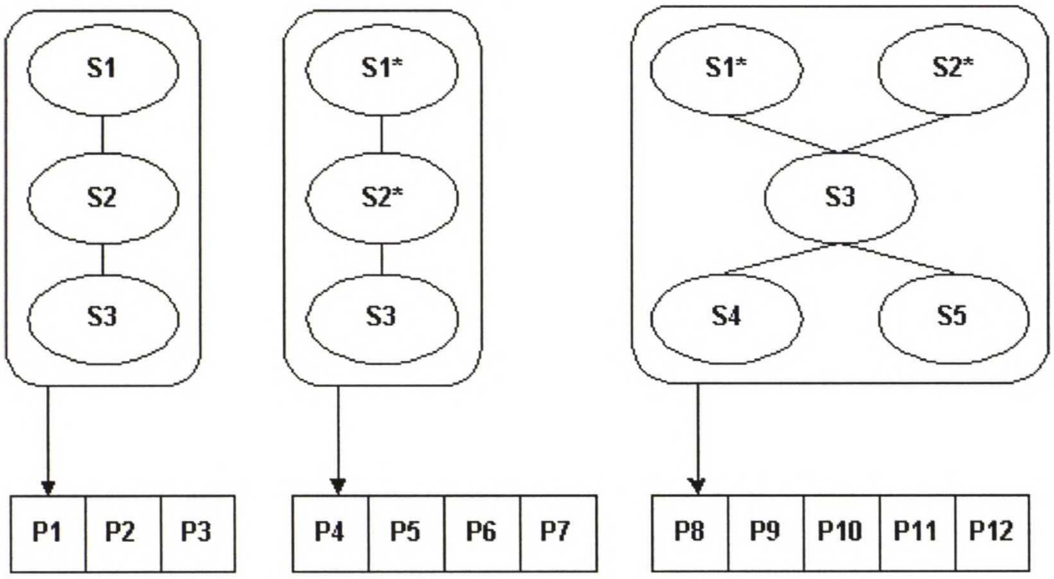

Figure 12: Defining changes to product platform and the products based on it (Meyer & Lehnerd, p 42).

A rollout plan for the platform should be made to have a forward-looking map for the product family. The plan includes schedules and contents for the initial platform and corrections made to it, the same things for coming platform extension versions and finally for the completely new, next platform version. Improvement plans of individual subsystems should be mapped to this whole platform rollout plan (Meyer & Lehnerd, pp 241-242).

Product technologies can be seen as implementations of the knowledge of a company, which are taken as components into the final product. Just as new derivative products are made through incremental improvements to the existing platform, the platform itself should be renewed by integrating complete component technology implementations. These components can be made by the firm itself or by external suppliers (Meyer & Lehnerd, p 46).

Product technologies include also the subsystem interfaces. This applies both to proprietary connections but more importantly to the de-facto or regulatory industry standards. Product technology work includes searching for new technologies, taking part in the industry standardisation and bringing competitive information in to the company (Meyer & Lehnerd, p 46). As a summary, the platform product technology should be in the leading edge of the industry all the time and bring new technologies for the use of the derivative products.

4.1.5 Interfaces

The real power of a platform is facilitated by its interfaces. There are three kinds of interfaces in a software platform (Meyer & Lehnerd, pp 180-181):

1. Internal interfaces between subsystems (see Figure 11 in Section 4.1.3). The clearer these interfaces are, the easier it is to replace individual subsystems. Internal interfaces do not interact over layer boundaries in a layered architecture model.
2. The user interface between the system and its user or a machine-to-machine interface between the system and, for example, databases or other similar systems (external interfaces and interfaces to applications in Figure 11).
3. Interfaces between the platform and the add-in modules in application layer that are attached to it.

One must notice, that in the case of a platform, the user is actually either an application or its developer, not the final user of a product. External interfaces are the ones that need to be industry standard compliant. This is quite well understood nowadays. If there does not exist a widely adopted standard, the companies involved come together and create it for the industry segment (Meyer & Lehnerd, p 180-181).

The modules in products are defined by their interfaces to each other. The interfaces reflect the way they interact with the other modules. Interfaces are composed of mechanisms, functions, constraints and attributes. A mechanism defines the nature of an interface; is it a mechanical, an electrical or a software interface, for example. A function describes what the module behind an interface does. Constraints give boundaries for the use of the module and attributes refer to the contents of the interface itself, for example inputs and outputs. Products are put together by linking modules to each other through internal interfaces of the product. Some modules contain an interface to outer world and thereby give an interface to the whole system. A combination of modules is a matched set of compatible interfaces. It has also two other attributes. The modules must meet the customer requirements or enable the product to meet them and operate within the given constraints or make the product to work within them (O'Grady, pp 43, 149-150).

Modularity and clearly defined interfaces increase the competition between several module suppliers and therefore speed up the development cycle. Clear interfaces also reduce the need of communication directly with the design teams of other modules and let the development personnel of a certain module concentrate on the module itself, not its environment. This improves the overall product evolution (O'Grady, p 31).

When building interworking application programming interfaces, successful companies take their partners early on to the development. By announcing APIs early, also the partners are able to build their solutions in a short time-to-market (Hoch et al., p 193).

4.1.6 Standardisation

The only way to maintain the required co-operation for building interworking products is standardisation. Before complete standards are available, the providers must come up with some solutions for their products. They can either produce a proprietary vertically integrated solution or co-operatively manage an infrastructure within a partnership of multiple participants. The former is closest to the way most of the vendors act nowadays, but those who adapt to the latter approach earlier, will gain competitive advantage by preparing for the universal, standards-based shared infrastructure that will emerge in the future (Wainewright, pp 7-8).

The need for well-defined external interfaces and interoperability of modules made by different companies drives the development of standardised interfaces. Standardisation is of high importance in the information industry related products. Besides the fact that the standards allow interaction between different bodies, it is nowadays also a customer requirement for an information industry product. That comes from the fact that standardisation increases competition and thereby lowers the prices and helps the customers to change the provider as the switching costs are lower (O'Grady, p 228).

Interfaces of a platform can be of strategic importance. According to Meyer and Lehnerd, experience has shown that specifying clear interfaces is critically important. They even suggest that the interfaces should have part numbers like components. Clear and standardised interfaces not only help to create derivative products faster but they also help a customer to use the systems better. Having an industry standard interface gives clarity and flexibility; the products of a company can be used in a variety of environments. Even more power from industry standards is gained when a company itself is active in creating the standards. This way the product of a company most likely fulfils the standards quickly and over the time also other vendors fulfil that industry standard with their products. Meyer and Lehnerd mention SUN Microsystems and its network file systems (NFS) -standard as an example of a widely adopted industry standard that was initially an interface definition of one company. SUN continued this by licensing its Java programming language to other vendors, which turned out to be a success (Meyer & Lehnerd, pp 40-41).

Widespread use of a multi-vendor, standards-based approach in system design helps to promote competition and stimulate technical innovation. An open standard interface encourages innovation by providing design flexibility, simplifying the work of the application developers and allowing the developers to focus on their core competencies and to create more and better applications. The end-users benefit from the continuous improvement of the services delivered (SA Forum, pp 8-9)

Advantages of standardisation are (Webb, p 105):

- Devices in different networks can talk to each other.
- The standardisation process is usually somewhat public, which brings more future-proof solutions to the market, as the viewpoints of different stakeholders are included.
- For the same reason, more vendors share the same vision of the future system architecture. This way the future solutions are implemented faster and can be taken into use quickly.
- Operators and their customers can buy their equipment from multiple providers, which reduces prices and speeds up the product development. It also gives them the feeling that they are not tight to a monopolistic supplier that might act in a self-interested way in the future.

Disadvantages of standardisation work include (Webb, p 106):

- Standardisation process takes time, mostly because of the publicity aspects presented above.
- Standardisation process produces sometimes an over-engineered solution, because all possible requirements are taken in. This adds costs and complexity to the vendors who want to thoroughly fulfil a certain standard.

Meyer and Lehnerd also talk about the disadvantages of standardisation. Long-term planning and research is essential in keeping a company in business, but it should not be the critical process in the development of the current product. Instead, building new products should mean integrating existing subsystems together, which takes a reasonable amount of time. Although this may seem to emphasise the meaning of the standardisation of subsystems and their interfaces, it is not a synonym for product platform. Robust product platforms yield standardisation but it does not work the opposite way. Too much of standardisation can cause the platforms to be heavy and inflexible, and there is no more room for customisation. It can also kill the means for competitors to differentiate from each other. The latter one drives only reduction of prices and forgets the idea of providing distinguished functional value for the user. For these reasons, standardisation should be selective, right elements for it should be selected. Interfaces are the ones that should be standardised, not the subsystem implementations. Internal structures of subsystems should leave the designers degrees of freedom to enhance them or to create completely new ones. This way some subsystems can turn out to be the differentiating factor for the company in the industry. Meyer and Lehnerd call this standardisation internal, and mention that those differentiating subsystems may become products themselves (Meyer & Lehnerd, pp 119-121).

Need for standard compliance is not completely dependent on customer acceptance, it is also a requirement for complex systems that integrate components of different vendors together. Module integrators are the ones who drive this requirement towards the module providers. By using standard compliant modules, an integrator has a possibility to be the first in the market. The hard task of an integrator is to assure the provider that the standard it wants to use will be the dominant one in the market (O'Grady, p 231).

Standards can be made by international standardisation organisations like ISO or ITU in the telecommunications sector. An advantage of the official standardisation organisations is that a wide variety of opinions and viewpoints is usually taken into account before setting a standard. For the same reason they are, however, considered to be slow and bureaucratic (O'Grady, p 229).

Another option is to use market driven standards. These so called de-facto standards are solutions that have gained powerful market dominance and other companies follow the market leader (O'Grady, pp 229-230). Nowadays companies have recognised the power of network effect and the disadvantages of competing standards in markets, and have therefore established alliances for creating a standard first and then started to compete for the market share.

4.2 Software platform products

A telecommunications platform is compounded of the server hardware, operating system, middleware software and the applications (see Section 2.2). Since the hardware and operating system are nowadays more or less standard components, the main target for productisation and differentiation of products is the middleware software layer. For this reason, the nature of software products is discussed here.

4.2.1 Nature of software products

A software product is a mixture of intangible components, which are integrated together and tested for compatibility and performance so that the buyer does not have to do that. The core product gives a customer the needed functionality whereas the supplementary functions enhance the image of its customer value. To point out the intangible functionality of a software product, strong marketing must be in place (Vitikainen, pp 42-43).

Companies can provide customers with complete products or partial solutions and the customers then choose their favourite combination of the providers for the final software solution (Sallinen & Seppänen, p 17). Customers may still want to buy complete applications, but more likely they want to have the possibility to change any component to a substitute that suites better for their purpose. For this reason, compatibility with third party components is a vital issue (Wainewright, pp 18-19). Packaged software products include software components, complete products and platforms. The degree of productisation in packaged software products varies from high via tailored to low. Things that vary between these are the amount of services for installation and maintenance, the novelty of the market and the novelty of the used technology (Sallinen & Seppänen, p 8):

To the production phase of a software product include compiling the software, copying it, printing of the documentation and packages and putting them all together and sending them to the distribution channel. Maintenance services are usually based on an agreement made between the seller and the customer for a certain period and for a certain price. The price includes correcting the faults that the customer has found and reported as well as the distribution of upgrades for the product (Junttila, p 20).

Customer requirements for a software product are often unclear, which results in complexity in the products. That is why incremental development should be used. It means first building a basic solution and then extensions to it. A problem is that only few standard solutions are usually available and even they might not produce competitive advantage for a company (Sallinen & Seppänen, p 16).

4.2.2 Required components of a software product

Junttila (1998) has a similar approach to software product components as was discussed in Section 3.1.1 for products overall. According to her, a software product has a core part, which is the program itself, and which is surrounded by the look and feel of the product. That means the packaging, the user interface and, for example, local features of the software product. In the outer section, there are the product services, the distribution channel, product management, maintenance support and, for example, user manuals and other guides (see Figure 13) (Junttila, p 14).

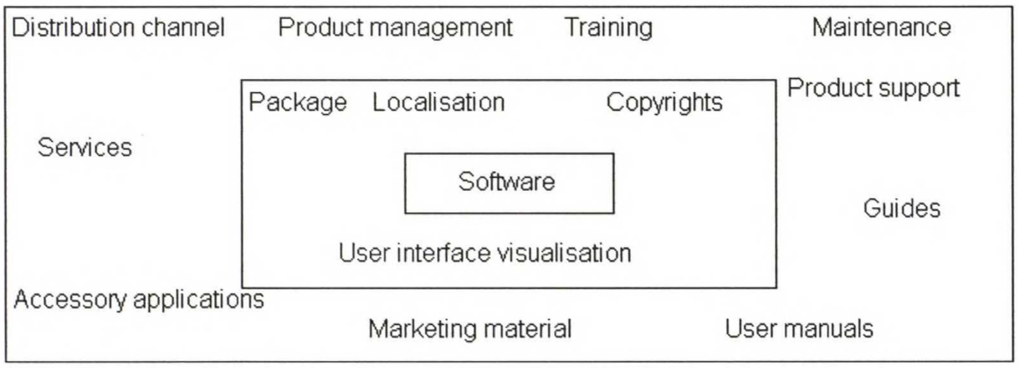

Figure 13: Parts of a software product (Junttila, p 14).

Delivering a software product to a distribution channel can be done by giving master disks of the software to it and that way save in the delivery costs. The distribution channel is then responsible of making the necessary copies of the product and reporting the correct number of sold items to the producer. The problem is just how to be sure what is the amount of sold items. An effective licensing practice is probably the only way to handle this. Another method nowadays is to handle the distribution via the Internet. It is especially useful in delivering corrections and updates to an already delivered software product or its documentation in the maintenance phase. The Internet is usually available 24 hours a day and is therefore immune for example for different time zones. It also gives a direct link from the customer back to the producer. In a tight competition nowadays, no software company can afford not to have a web page for its product (Junttila, pp 77-78).

Services related to a software product usually take place in the maintenance phase. The product support depends of course on the product itself and the already implemented support activities but also on the nature of the customer and its environment. Especially this is true in industrial business where customers buy expensive products and their number is relatively small (Junttila, p 80, originally Taylor 1992).

4.2.3 Software platform

Every software product has its internal architecture that is formed from subsystems and their interfaces. This architecture can be structured in such a way that it creates a robust platform for effective development of derivative products. Platform is the engine, and derivative products are created by developing add-in modules and plugging them into the platform. Variety and the feel of tailored products can be offered to customers by providing customised modules, which can be taken in or left out from the product with no additional integration cost. The platform, the engine, is a core set of programs that propel the whole system (Meyer & Lehnerd, pp 177-178). The approach is very similar to the ideas presented in Section 2.2 about the layers of a telecommunications network element.

Just as the physical products, the software subsystems must fulfil industry standards and their internal interfaces must be clearly defined in order to effectively integrate them to work together. The add-in modules integrated into a platform represent the incremental development for special user group needs. The whole power of platforms lies in the fact that the application developer does not have to start from scratch with every new product. The R&D efforts can be focused to integrating the applications and tools in platform level instead of the hardware or operating system level. At the same time as the existing platforms make the development of new derivative products faster, the increments of a platform open up new market opportunities. Improvements in the underlying software platform should still be hidden from the end-customer that uses the derivative products. This requires good architecture and interfaces so that the add-in modules do not have to be renewed every time when the platform develops (Meyer & Lehnerd, pp 179-180).

4.2.4 Software architecture, modularity and interfaces

Software architecture is harder to understand just by looking at it than physical products. The approach of product families and platforms is still fully adaptable to non-physical products like software and information products or services like insurance. Software is usually built on a layer model where upper layers use services from the lower ones. Meyer and Lehnerd introduce the layers to be hardware, operating system, software development tools and applications (see Figure 14). The development of systems has evolved over the time from proprietary to open systems. A common approach is to use client-server architectures. The networked computing has accelerated this development even further (Meyer & Lehnerd, pp 172-174).

Development tools form a key layer in platform thinking (see Figure 14). They contain things like programming languages and libraries of functions or classes, which can be used in building of specific types of end-user applications. Tools are made as modular and flexible as possible so that the application developers can use a combination of them according to their needs. Using widely adopted tools, the final applications can be developed by a completely different software company than the platform. The tools may be generic or meant for a special type of application development, like industrial, medical or telecommunications industry applications. See more about the development tools in Section 4.2.5 below (Meyer & Lehnerd, pp 172-174).

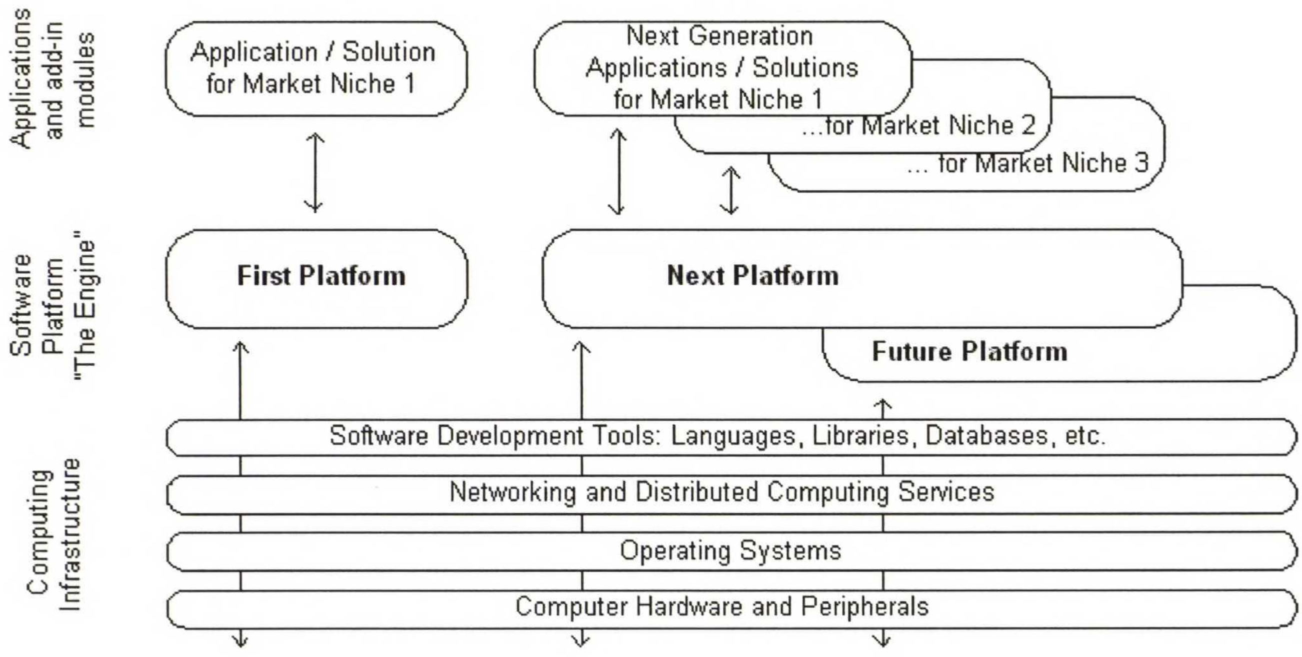
Applications and add-in modules
Software Platform "The Engine"
Computing Infrastructure
Application / Solution for Market Niche 1
Next Generation Applications / Solutions for Market Niche 1
...for Market Niche 2
... for Market Niche 3
First Platform
Next Platform
Future Platform
Software Development Tools: Languages, Libraries, Databases, etc.
Networking and Distributed Computing Services
Operating Systems
Computer Hardware and Peripherals

Figure 14: Managing the evolution of Software Product Family (Meyer & Lehnerd, p 193).

Different services and tools have been put on different layers of the architecture in the past and this phenomenon will be true also in the future. For example, local area networking was done before by specific products like Novell Netware, but today basic features of networking are included in the operating system (Meyer & Lehnerd, p 174).

The current and future network environments encourage a modular software componentisation. Technology trends in standards-based, multi-provider network infrastructure include (Wainewright, pp 8-9):

- Plug-in technology for third party functions,
- Remote infrastructure services providing behind-the-scenes functionality (user authentication, payment processing, etc.),
- Enterprise application integration, linking applications within an enterprise or across a value-chain.

The principle of using modularity in software consists of two things. First, modules must be cohesive, i.e. logically related modules should be bound together. Second, they should be loosely coupled, meaning that dependencies between the modules should be minimized (O'Grady, p 183).

Success and elegance in the design of software architectures is achieved through modularisation and minimising the number of interfaces in a system. Optimising each subsystem may result in sub-optimisation of the system as a whole, however. Minimising the number of interfaces is essential and it has to be ensured that those interfaces are not allowed to develop in an uncontrolled manner. One change in how the subsystems work together usually generates a chain of modifications, which can be described as a ripple effect to the whole system. This can be devastating to the whole platform system development. Managing the internal and external interfaces is as important as developing new functionalities to an existing system (Meyer & Lehnerd, pp 183-184). Ran reminded that complex systems are not built from simple components. The components are, wanted or not, still dependent on each other and must follow some system-wide policies for example for security, overload control and for fault detection (Ran, p 164).

4.2.5 Development environment

The need for a rapid product development has increased the importance of languages and tools in developing the applications. Approaches of reusable code and object oriented modelling are also consequences of this need (O'Grady, pp 183-187). In Section 4.1.1, effective application development was discussed. As a consequence for those thoughts, it is rather obvious that a platform must offer an environment with tools for the application development.

Sommerville (1995) uses the term computer aided software engineering (CASE) to denote the automated tool support the for software development. There are nowadays various different CASE toolsets available. Most of them are production-process support tools for specification, code generation and testing. A few of them are process management support tools for process modelling management and some are meta-CASE support toolsets for generating other CASE-tools and processes. Terms such as tools, toolsets, workbench and environment are used inconsistently to mean CASE-tools. Sommerville defines that the tools support individual process tasks like compiling a program, whereas a workbench supports a complete phase of the process like the specification or the implementation phase. Workbenches usually contain a set of tools integrated together. Environments support all or nearly all phases of the process and include several workbenches (Sommerville, pp 506-510).

The development environment tools can be classified according to which phase of the process they support or according to the extent of support in that particular field. There are tools for example for the following phases (Sommerville, pp 507-508):

- Product development management (estimation etc.),
- Editing,
- Configuration management (version and change management systems),
- Prototyping (high level languages, user interface generators),
- Method support (code generators, data dictionaries),
- Language processing (compilers, interpreters),
- Program analysis,
- Testing,
- Debugging,
- Documentation,
- Re-engineering (cross-reference or program restructuring systems).

Sommerville did not include in the list the design tools, which are also an essential part of the development cycle.

A software engineering environment (SEE) is a set of hardware and software tools integrated together and supporting the software development process from the initial specification to the module and system testing. What distinguishes the software engineering environments from CASE tools is that the facilities in them are integrated from platform, data, presentation, control and process integration point of view. The SEEs support teamwork-based development and configuration management. They also support a variety of different tasks related to the development process, from specification to testing. A part of the integration of tools is that all project related files are stored in one shared place, and several or all tools have an access to it (Sommerville, pp 548- 549).

Sommerville presents a SEE to contain three layers: platform services, framework services and workbench applications. Developers of software see the environment as a set of workbenches supporting their application development process. Note, that here again Sommerville has used the word 'platform' in a different meaning than elsewhere in this thesis (Sommerville, pp 549-550).

The platform services of a development environment depend on the hardware and operating system on which the development is done. This environment is called the host system. It is possible that the final software runs on an environment, which does not have development work facilities (see Figure 15). Embedded software has typically this kind of restrictions. The platform services are usually used in a development environment where several computers are connected to each other with a local area network. The environment must allow these computers to be of different types and to have different operating systems or different operating system versions. The common platform services for all developers include file and process management services, network and communication services as well as window management and print services (Sommerville, pp 550-551).

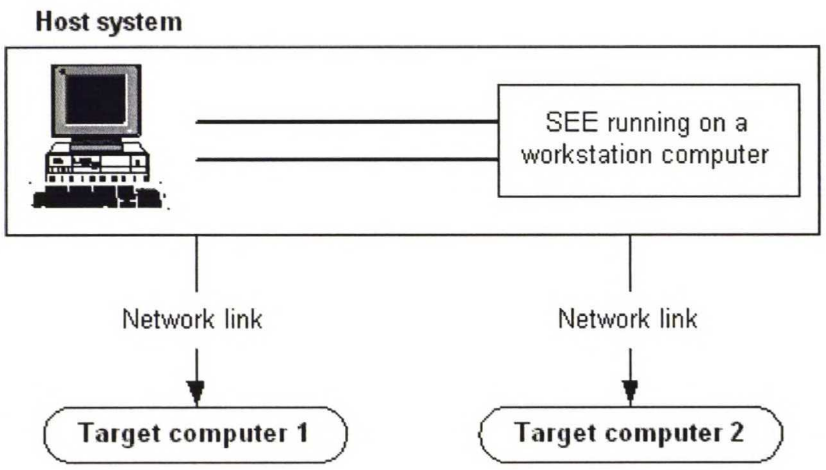

Figure 15: Host-target development (Sommerville, p 551).

The framework services are usually implemented by using individual platform services. They are specifically designed for a certain CASE workbench for a dedicated task. Sommerville identifies five services that he includes in a framework services reference model (Sommerville, pp 552-558):

- Data repository services, for naming and storing entities and creating dependencies between them,
- Data integration services, which supports software development by providing version and configuration management to entities,
- Task management services, which provide activity execution and event monitoring services in the environment,
- Message services for tools and frameworks to communicate with each others where some tools act as deliverers and the others as registered users for messages,
- User interface services, which are divided into layers and by which they support integration of presentation.

In the workbench applications layer of the development environment, Sommerville presents three levels of tools and types of integration for them (Sommerville, p 559, see Figure 16):

- Integrated tools which are totally inside a framework and use only services from it,
- Semi-detached tools, which manage also own data entities but still use services from a framework,
- Foreign tools that run on the same machine but do not use any services from the framework.

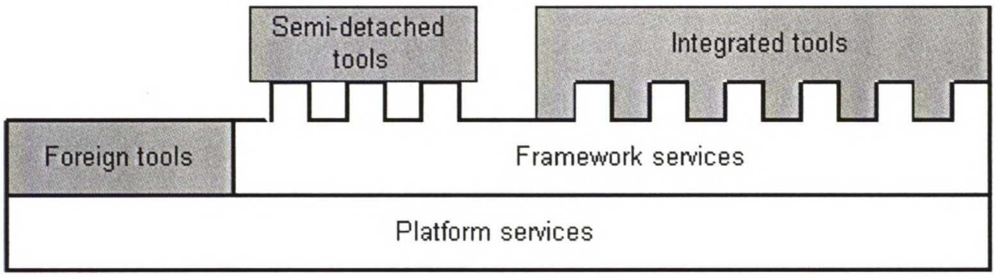

Figure 16: Tool integration with a software engineering environment (Sommerville, p 559).

A SEE should allow the usage of all kinds of tools and the possibility to integrate new ones to the environment as the needs for application development evolve. There is still a certain circulating problem with the integrated tools. Total power from a platform and a framework is got only with integrated tool environments, which over the time increase the market size for individual tools in it. However, the tool developers are not keen on developing tailored integrated tools until the size of the market is already high enough (Sommerville, p 559).

5. EXAMPLES OF PRODUCT SOLUTIONS IN INDUSTRY SEGMENT

In this chapter are introduced some product offerings that have similarities with the platform that the case company is productising. First one is the IBM WebSphere Application Server and the second one the Resilient Telco Platform of Fujitsu Siemens Computers. In the end of the chapter, a look is taken into the standardisation work done by the Service Availability Forum.

IBM WebSphere is a software platform for making and running e-business applications in several supported computing environments. It provides basic building blocks, tools and methods for adding user accessibility to services and ways to integrate solutions to other businesses. IBM promotes in its web page that WebSphere is a solution to dynamically integrate business strategy and information technology into e-business (IBM web (1), 14.10.2002).

The WebSphere product family consists of several products built on top of a software platform. In other words, IBM has used the approach of building a common platform and then dynamic derivative products on top of it. The family includes, for example, the following software packages (IBM web (1), 14.10.2002):

- WebSphere Business Integration,
- WebSphere Application Server,
- WebSphere Studio,
- WebSphere Portal for Multiplatforms.

The WebSphere Application Server is chosen for closer investigation in this study.

The Resilient Telco Platform of Fujitsu Siemens Computers is a software platform solution for building highly available telecommunications application servers. It uses standard hardware and a standard operating system under it and provides an application programming interface for applications.

5.1 IBM WebSphere product in general

IBM WebSphere Application Server is a scalable transaction engine for e-business applications. It provides a foundation that supports the on-demand business applications. It contains a Java 2 Enterprise Edition (J2EE) application server and a development environment for it (IBM-web (1)). IBM has defined five separate editions of the product (IBM-web (2), p 1):

- WebSphere Application Server-Express, an easy to use start-up for e-business applications,
- WebSphere Application Server for z/OS, J2EE qualified configuration optimised for z/OS services,
- WebSphere Application Server Network Deployment for distributed configurations using clustering, edge technologies and high availability,
- WebSphere Application Server, the core product of the family with J2EE and web technologies,
- WebSphere Application Server Enterprise to integrate business operations to application server functionalities.

Based on these editions, IBM has defined four different scenarios of the base products and development tools for different customer types. The scenarios are available for start-up companies, for small, growing companies, for mid-sized companies and for large, multinational companies (IBM web (3), 14.10.2002).

Besides the basic configurations, the provision contains also adapters, with which the customers can connect their business applications to the Application Server using industry standards like J2EE and XML (IBM- web (4), 14.10.2002).

WebSphere Application Server version 5.0 is J2EE -standard compliant, as it passes the SUN J2EE test suite. It implements a set of J2EE services and supports the SUN J2EE APIs but contains also additional, proprietary interfaces. Standard interfaces for Enterprise Java Beans (EJB) are defined in javax.ejb classes, which are described in the SUN web pages (Interview, B.K, 1.11.2002, and SUN-web, 4.11.2002).

The WebSphere Application Server is deployed to the target environment using a GUI-based installer or directly by decompressing an rpm-file. It then runs as a separate process in the target computer. Other applications use applications inside the application server process by first resolving the address for the service and then invoking calls directly to the application (Interview, B.K, 1.11.2002).

5.2 Components and services of the WebSphere Application Server offering

5.2.1 Documentation

In WebSphere Application Server, a hypertext documentation centre comes with the product. It is meant for both the application developers and the system administrators. A number of documents can be downloaded also in pdf-format from the IBM web pages. They contain, for example, red books about main topics for developers, like security in J2EE (Interview, B.K., 1.11.2002).

5.2.2 Support Services

Since IBM is a big company and its products have thousands of customers, support services for the products are also extensive. IBM provides a problem reporting system for the application developers in customer organisations. A customer unit has only one user account in the system and must therefore deliver the problem reports through one dedicated person (Interview, B.K., 1.11.2002).

There is also a feature request database in the IBM website. Again, there is only one user account for each customer company to this database. The customers can seek through the features that have earlier been requested to avoid requesting same features again. Customers can see the requests made by others, but they do not know who has made each request (Interview, B.K., 1.11.2002).

IBM has collected several customer support services under one service point in the Internet. It calls this point the Community. Besides feature requests, the Community website includes features for: (IBM-web (5) 14.10.2002):

- Submitting an idea or article to be published in their website,
- Attending to an event or conference related to WebSphere or submitting an own idea of an event that should be in the list,
- Sending questions to IBM WebSphere experts, which publish monthly a list of top questions with answers to them on the website,
- User groups for changing ideas and experience among other developers using WebSphere.

5.2.3 Education

IBM promotes education modules for its products in its web page. There are instructor lead courses for WebSphere in general, for the Application Server separately and for other WebSphere tools. There are also distance learning packages and online tutorials for different parts of the product family. IBM provides also personal training in the Internet with web broadcast modules. It has 2-3 days training courses, where the developers are taught how to make applications to the Application Server (IBM-web (6) 14.10.2002 and Interview, B.K 1.11.2002).

IBM provides newsletters and magazines for developers using the WebSphere product family. There are several different magazines or newsletters that the developers can subscribe (IBM-web (7) 14.10.2002).

Furthermore, IBM has a concept called the Software Services for WebSphere. It offers technical, product-specific services for the WebSphere software family products. Those include skills transfer, implementation, migration, architecture and design services, pilot workshops and education as well as training on how to install, configure and maintain an enterprise-scale WebSphere Application Server and assess the performance of applications built on it. There are different levels of service suites (overview, mini suite, full suite and so on) and they contain varying combinations of classroom training, self-study and hands-on –type of training. The classroom and practical trainings are lead by IBM's own application area experts and the company calls these education solutions the Centers of Excellence (IBM-web (8) 14.10.2002).

5.2.4 Development environment

As discussed earlier, a platform -type of product needs an efficient set of development tools in order to make suitable applications on top of it. IBM offers the WebSphere Studio Application Developer for WebSphere Application Server, which is a graphical development environment for building, testing and deploying applications in several operating systems and with multiple devices. The development environment includes command line scripts for making deployable EJBs from source code java classes. They are stored in the JAR-file format. After deploying applications with the WebSphere Studio tools to the Application Server, other applications can immediately start to use them. The tools and scripts are delivered in a compressed file format and they can be installed directly to the development environment (IBM-web (9) 14.10.2002 and interview, B.K., 1.11.2002).

There is also a Telecom Application Toolkit for developing telecommunications applications and integrating them with the Application Server or with other IBM WebSphere products. It is an extension to the WebSphere Studio Application Developer and it helps in integrating e-business applications to different types of telecommunications networks (wireline, wireless and internet). It is based on industry standards like Java, Parlay, OSA and 3GPP (IBM-web (10) 14.10.2002).

5.2.5 Consultation

IBM is one of the biggest services and consultation companies in the world. With its teams of consultants it can provide customers services for assessing, planning, designing and implementing e-business solutions based on the WebSphere product family. IBM uses its Global Services organisation resources for delivering services for the WebSphere Application Server customers. Help services are available, for example, for installation and application implementation (IBM web (8), 14.10.2002).

5.3 Fujitsu-Siemens Resilient Telco Platform (RTP)

Fujitsu Siemens Resilient Telco Platform (RTP) is a software solution that addresses users who need to achieve permanent availability of their applications. The RTP requires no modifications to the operating system or the database used. Instead, it works on top of a standard, off the shelf Solaris or Linux system in a standard cluster environment, offering additional layers of functionality. The single system image that the RTP provides makes the application independent of the complexity of the underlying hardware and software and especially of the current system configuration (see Figure 17). Thus, the application can be unaware that it is a part of a cluster, which makes the development of applications easier. The location of the resources within the cluster is completely transparent to the application and the RTP takes care that they are available all the time, no matter where they are actually located (FSC-web (1), 8.1.2003).

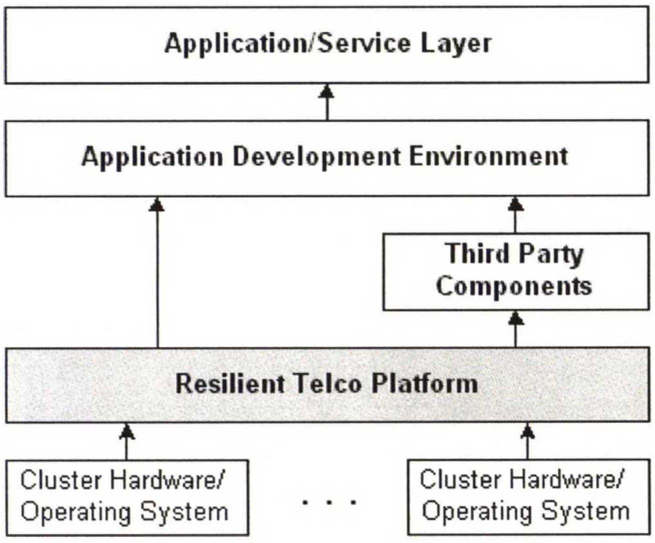
Application/Service Layer
Application Development Environment
Third Party Components
Resilient Telco Platform
Cluster Hardware/ Operating System
. . .
Cluster Hardware/ Operating System

Figure 17: Overview of Fujitsu Siemens Resilient Telco Platform4 Continuous Services (FSC (3)).

The Resilient Telco Platform as such is a software solution addressed by Fujitsu Siemens Computers. The product that currently is based on this solution is called Resilient Telco Platform^4 Continuous Services (RTP^4). The major building blocks of the RTP^4 platform are (FSC (1), p 2 and M.R. mail interview 21.1.2003):

- Off-the-shelf server hardware,
- Standard, open operating System (Linux),
- Standard Cluster Database (e.g. Oracle),
- Standard Cluster Framework,
- Resilient Telco Platform 4 Continuous Services Middleware.

5.3.1 Product and target group in general

The Resilient Telco Platform^4 Continuous Services (RTP^4) is a software layer for the implementation of continuously available communications services and E/M-commerce applications (FSC (3), p 1). It addresses customers, who want to implement true zero downtime applications with availability beyond 99.999% (less than five minutes downtime per year). This requirement can only be met by applications that operate in parallel and can respond to the failure of individual components with rapid switchover times. The switchover is done to additional redundant components. Writing and testing such applications is a very expensive business. It requires highly specialized knowledge from the application developers about the underlying layers (hardware, operating system, databases and protocols) (FSC (4), pp 5, 10).

The RTP^4 is targeted for network equipment manufacturers, for integrators and for ISVs, who are focusing on mobile business services and taking the advantage of the convergence of networks and media (FSC (4), pp 23-24). It is based on standard, open hardware and software products. High-availability is achieved through cluster technology. The middleware layer makes the cluster technology transparent to the applications and provides the generic functionality needed by continuously available carrier-grade applications and services. Carrier grade services are defined by supporting (FSC (3), p 2 and FSC (4), p 3):

- Scalability from a few thousand to millions of users,
- High parallelism with more than one million busy hour call attempts (BHCA),
- Service availability in the order of 99,999% or more,
- Very short response times to client requests (less than 500 ms).

When a solution provider develops an application based on RTP, the service can easily be deployed on several hardware platforms, because the RTP handles the interaction with the hardware (FSC (4), p 14).

FSC has grouped the actual RTP functions into three different areas (FSC (4), p 15):

- Scalability and availability area,
- Protocols,
- Operations, administration and maintenance area (OAM).

5.3.2 Suggested benefits and value of the RTP

Fujitsu Siemens Computers suggests that with systems based on the Carrier Grade Linux and RTP, the user benefits from a low cost of ownership and a low integration overhead through the consistent use of standard components. Compared to the traditional proprietary solutions, the user may utilize the latest state-of-the art processor and computing platform technology using off-the-shelf components. The well-known operating system environment and quick and easy learning curve for RTP assists developers to be productively working on RTP-enabled applications, speeds up the time-to-market as well as the customisation (FSC-web (2) 8.1.2003).

Benefits for application developers from using RTP are that they can concentrate on business aspects like application logic and user requirements, whereas the distribution model, hardware dependencies and recovery actions are addressed by the cluster experts (FSC (4), p 11). Issues that are handled by the platform provider are (FSC (4), pp 23-24):

- High scalability and availability,
- Cluster management, recovery handling and error treatment,
- All critical scalability and HA features tested on RTP level,
- Providing rolling software upgrade features.

The key benefits from using the RTP for application developers and integrators, network equipment manufacturers or ISV organisations are listed in Table 2.

Table 2: Benefits of using RTP for different groups of customers (FSC (1) p 4, FSC (3) p 2 and FSC (4), pp 23-24).

Benefits for application developers	Benefits for the equipment manufacturer organisation
Short time to market through straightforward hardware abstraction; developers see a consistent single system image, independent of hardware configuration.	It helps to focus on company's service and application core competencies.
Cost control of carrier-grade applications through the use of RTP functions.	Allows the use of standard development and deployment technologies.
Scalability from developer desktop to largest carrier-grade cluster installations.	Reduces implementation complexity due to well-designed application and administration interfaces.
Improved application portability through the separation of high availability functions from application features.	
Full, worldwide support by Fujitsu Siemens Computers.	
Minimal adaptation and training due to modular software structure.	
A standard based solution, as the RTP solutions have been proposed to the Service Availability Forum as an open industry standard.	

5.3.3 Parts of the product

The RTP platform product offering is divided into three layers: the Foundation Package, the Optional Add-ons Package and the Service Packages (see Figure 18) (FSC (3), p 2).

In the RTP Foundation Package, the key features are provided for the cluster wide, safe process communication as well as for the management of cluster nodes. In the optional RTP Add-ons, typical functions in high-availability communications systems are provided. Those are, for example, ticketing services for billing and payment systems, timers for reliable control of the system and application status and contexts for the system wide access to consistent state information. In addition, connectivity is supported through a wide set of telecommunications and IP protocol stacks (FSC (4), pp 23-24).

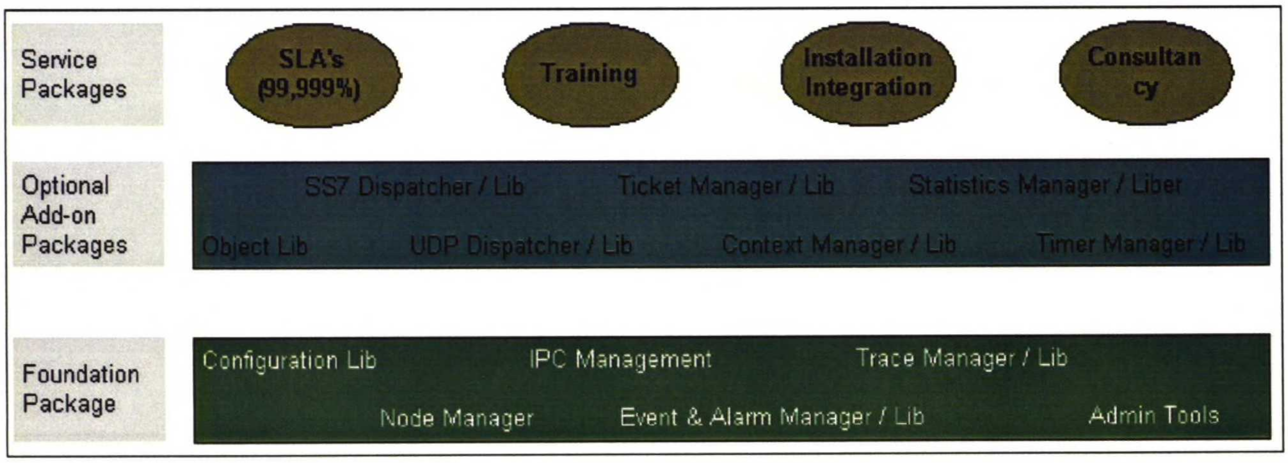

Figure 18: Packages forming the Resilient Telco Platform product (FSC (3)).

The service packages are provided for the RTP to help its users (FSC (4), pp 23-24):

- With the design and implementation of carrier-grade services and applications,
- To provide service level agreements as required by their customers,
- In the integration and installation of services and applications fulfilling the highest availability and scalability requirements and
- In the training of their customers.

Fujitsu Siemens Computers provides its customers with a change request procedure for entering requests of new features to the product. There is a defined structure how their implementation is handled and how different features are prioritised. The plan is discussed together with the customers. Error corrections for the platform software are usually downloaded from a special web page or they can be delivered through email. Maintenance services also include a new version of the software approximately once a year but error corrections and new features including upgrades can be delivered more often between the versions (M.R., mail interview 21.1.2003).

The development environment of the platform consists of several commercially available software development tools. These include Java 2 Software Development Kit, the standard GNU C++ tools, which are included in Linux and, for example, a commercial database software tool (M.R., mail interview 21.1.2003 and FSC (5), pp 2-3).

Together with the platform product, FSC delivers application programming manuals, administration manuals as well as installation and configuration manuals for the customers. Altogether there are several thousands of pages of them (M.R., mail interview 21.1.2003 and FSC (5), p 2).

5.3.4 Standards and interfaces

Fujitsu Siemens Computers is jointly with other companies, such as Intel and Nokia, participating in the development of the high availability API standards by working in the Service Availability Forum (SAF) (FSC-web (2) 8.1.2003).

FSC lets possible customers see the APIs after they have signed a non-disclosure agreement. APIs are currently designed by the FSC itself, but there is a plan to make them gradually SA Forum -compliant in the future. This shall be done in a systematic way and the timetable will be agreed with the current customers who already have implemented applications using the current RTP APIs (M.R., mail interview 21.1.2003).

The RTP functions are modular, and through the RTP standard interface they can easily be adapted to the reliability needs of the applications. They are running on standard Unix/Linux systems with a cluster support (FSC (2), p 1). Based on this modularity, some individual components are left optional and therefore variants from the basic product can be made. There are some obligatory basic components, but otherwise variants are made by applying different licensing methods to the optional modules (M.R., mail interview 21.1.2003).

5.4 Summary of product concepts

In this section, a summary of the two product concepts explained above is presented. Issues are first explained and then summarised in Table 3.

Both of the products fit well for the idea of horizontal markets, since they are not end products but meant to be sold to the vendors of the final application in the same industry area. The product offerings show that there is a certain target group for them defined, but room for variation is reserved. For example, the RTP^4 Continuous Services platform consists of several optional add-on packages on top of the basic package. It allows customers to make their needed configuration of the platform according to the applications they are building. The basic requirement for this kind of modularisation is that the interfaces between modules are well defined.

The application programming interfaces are also important. Both of the product offerings emphasize that their APIs will be based on relevant industry standards. The API of the IBM WebSphere Application Server is in line with the SUN J2EE API and it passes the SUN J2EE tests. There is also a group of non-standard APIs for special purposes, but those are not necessary for basic functionality (B.K. interview, 1.11.2002). RTP APIs of Fujitsu Siemens Computers are at the moment proprietary and not public, but they are planned to be compliant with the Service Availability Forum specification over the time, as the specifications become ready.

Regarding the APIs, for effective application development there must also be a useful set of documentation in place. FSC provides several thousands of pages documentation whereas IBM counts on electronic formats, mainly hypertext documentation. IBM also refers to the API documentation provided by SUN in its J2EE web pages.

Both of the products provide toolsets for the application development on top of them. The IBM WebSphere has its own special tools and scripts for deploying applications to the Application Server. Fujitsu Siemens Computers RTP on the other hand enables the use of standard commercial Java and C++ tools.

Post-sales services are a thing that distinguishes a sellable industry product from a company internal one. Basic maintenance services like problem reporting and fixing as well as new feature request services are considered as axiomatic and not very much emphasis is put on them in the marketing of these products. Instead, the installation and consultation services are emphasised more. This is reasonable, as the products are special expert products and are mostly used to build new kind of applications on top of a new kind of technology. The products are by no means standard products when considering the start of their usage. The installation procedure depends on the infrastructure and the development environment used by the customer. Effective application development cannot be started in a short timeframe by only reading manuals and APIs. Therefore, the provider companies have recognised the need for consultation and training services.

Table 3: Summary of the introduced product concepts.

A feature included in the product offering	Special notices
Purpose and target group defined, but room for customisation left in.	Also predefined variants of the product defined.
Interfaces and APIs are documented and they are either public or available for a potential customer.	At least APIs that fulfil a standard are public.
Interfaces and APIs follow or are planned to follow an industry standard.	Some APIs may still be proprietary.
Emphasis is put on post sales services, especially to installation and consultation services.	The size of customer base affects to the nature of services.

5.5 Service Availability Forum

The Service Availability Forum (SAF) is a coalition of several leading communication and computing companies. They are jointly trying to develop open standard interface specifications for a carrier grade infrastructure. The SAF is supposed to offer standard interfaces between the hardware, the software blocks, the management middleware and the applications of packet-based networks (SAF-web (1) 31.1.2003).

The background for the work of the SAF is that currently the communications infrastructure products are implemented with proprietary interfaces. It limits the software innovation and produces high development costs and lengthy times to market for developers and manufacturers. The Service Availability concept of an always-on uninterrupted service is jeopardized from the outset when disparate, proprietary pieces are connected in the new packet-based multi-service network. The SAF drives the industry adoption of open interface specifications that will benefit independent software vendors and the equipment, platform and service providers. These specifications are designed to create standard building block solutions that perform as effectively as the 99.999% availability level set by the legacy, circuit-switched telephone networks (SAF-press release, 28.4.01).

The Service Availability solution starts with high availability. The devices and systems that make up the communication infrastructure must be available to provide services and applications to the end user. High availability is achieved via a combination of equipment reliability, reparability and redundancy. To the high availability, the Service Availability solution adds service continuity. It means that the end user sessions are maintained despite any failure in the individual systems or components. For example, in order to provide an uninterrupted service, the state of the application session for each user must be preserved during switchover scenarios (SA Forum, p 4).

To put it simpler, the Service Availability attributes include (SAF-press release, 28.4.01):

- On-demand service, meaning that the system is up and services are available to meet or exceed the customer requirements.
- Uninterrupted service, which means that the customer sessions and state are maintained and preserved without disruption to service.

Benefits of a standard Service Availability interface solution are that it reduces development times and costs, allows design flexibility and reduces risks. The development times can be reduced when commercial off-the-self components are used and a standard interface approach is used in building blocks. It makes the systems easier to deploy and maintain and facilitates also outsourcing of the development work (SA Forum, pp 8-9).

The costs are reduced, as there is no longer vertical proprietary equipment to be developed, tested or maintained. Deployment of the systems and training and education of the technical and operational people is easier and cheaper with a common interface solution (SA Forum, pp 8-9).

Flexibility and reduced risk comes true as the standards based approach with an open, building block – type of system architecture enables the equipment manufacturers to mix and match the best in class or at least the most suitable components from a variety of competing vendors. In addition, a standard interface and an open architecture create the manufacturers and system integrators architectural flexibility to more easily adapt to changing technologies (SA Forum, pp 8-9).

Initially, the Service Availability Forum will develop two interface specifications, the application interface and the hardware platform interface (see Figure 19). The application interface will be a programming interface between the applications and the Service Availability middleware. The hardware platform interface will be an interface between the operating system or hardware platform and the Service Availability middleware. The Service Availability Forum interface specifications will provide the ability to vertically integrate applications and systems to the Service Availability middleware without an access or modification to the source code (SA Forum, p 6).

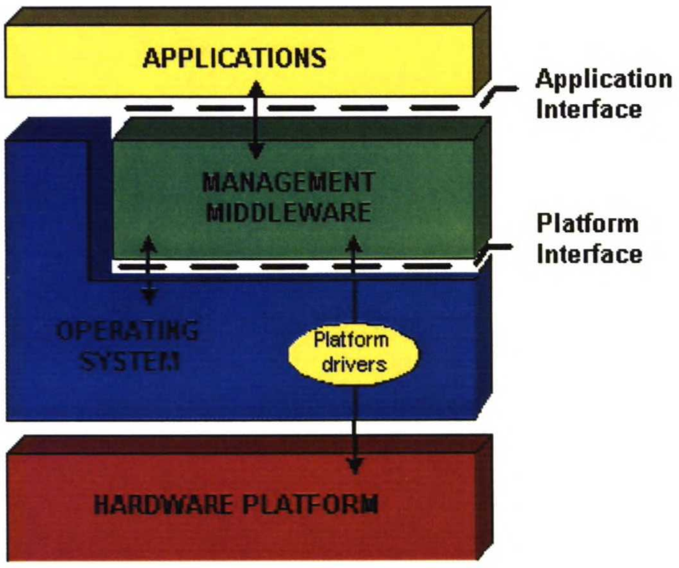

Figure 19: The Service Availability Forum Specifications and interfaces (SA Forum, p 7).

Open programming interface specifications will facilitate the development of equipment and applications that meet the demand of users for high availability and service continuity. The initial specifications will define programming interfaces to allow applications and platform resources, such as operating system facilities and hardware, to be integrated and coordinated by the Service Availability middleware (SAF-web (2) 8.8.2002)

6. THE CURRENT PLATFORM OF THE CASE COMPANY

6.1 Product

6.1.1 Overview

The telecommunications platform that the case company is developing is mainly intended for host control and service layer applications in All-IP networks. Examples of such applications are Connection Processing Server (CPS) and Home Subscriber Server (HSS). The platform is not intended for user plane processing. Other platforms both exist and are being developed for it. The server clusters, i.e. the network elements based on this platform will have at least 99.999% carrier grade availability. The elements shall be scalable and offer a high performance (Prod_descr., p 6).

The platform in question is a carrier grade open server platform for All-IP mobility systems. It leverages standard based hardware, an OSDL-specification compliant Carrier Grade Linux with high availability extensions, open application interfaces and standard based development tools (see Figure 20 for the overall architecture of the platform). The high availability extensions are planned to be Service Availability Forum (SAF) compliant as soon as the SAF-specifications are ready (Purpose_doc., slide 8).

The hardware architecture of the platform is based on loosely coupled computer units connected with a high capacity gigabit Ethernet. It allows high flexibility in configuring the platform for different applications and for different capacity and performance needs. With this approach the capacity figures achieved for the control plane elements are substantially higher than those that could be achieved with a shared bus or tightly coupled technologies (Prod_descr., p 7).

From the outside, the cluster is seen as a single server or a couple of servers with high performance and high availability figures. From the inside, the applications are mainly unaware of the cluster configuration. Same applications work in small and big configurations without modifications (Prod_descr., p 9).

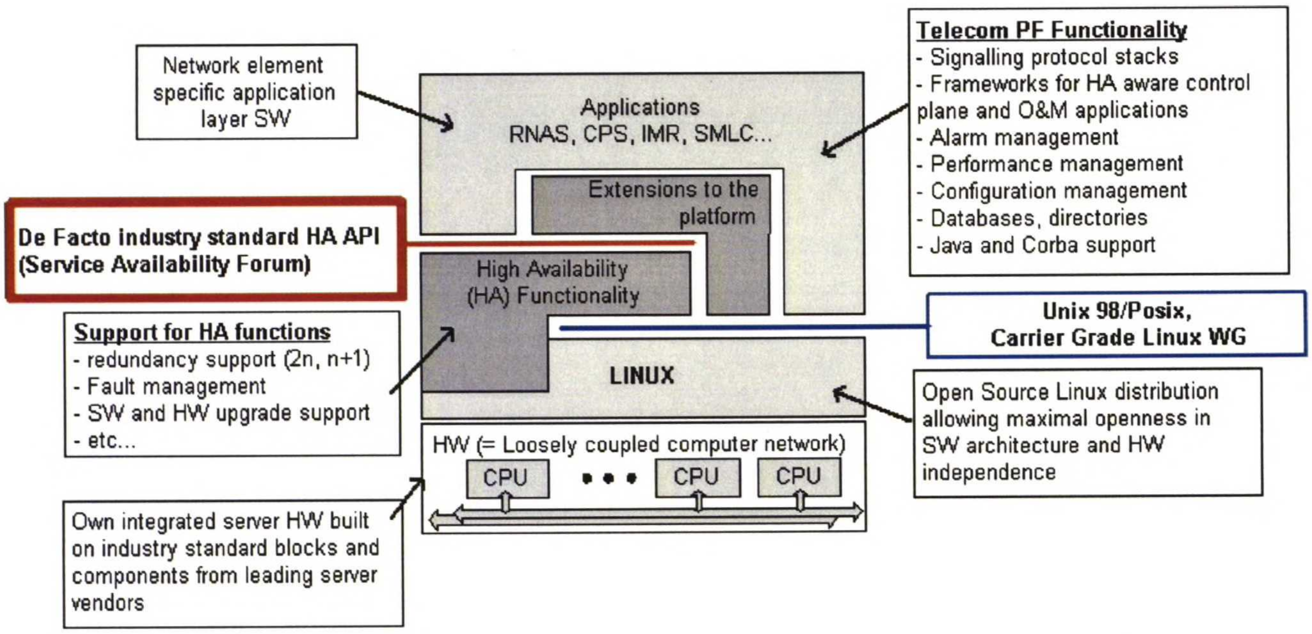
Network element specific application layer SW
Applications
RNAS, CPS, IMR, SMLC...
Telecom PF Functionality
- Signalling protocol stacks
- Frameworks for HA aware control plane and O&M applications
- Alarm management
- Performance management
- Configuration management
- Databases, directories
- Java and Corba support
Extensions to the platform
De Facto industry standard HA API (Service Availability Forum)
High Availability (HA) Functionality
Unix 98/Posix, Carrier Grade Linux WG
Support for HA functions
- redundancy support (2n, n+1)
- Fault management
- SW and HW upgrade support
- etc...
LINUX
Open Source Linux distribution allowing maximal openness in SW architecture and HW independence
HW (= Loosely coupled computer network)
CPU
CPU
CPU
Own integrated server HW built on industry standard blocks and components from leading server vendors

Figure 20: Overview of the current platform and applications on top of it (Prod_descr., p 6).

There are two alternatives for the hardware provided for the platform. The first is commercial off-the-shelf rack-mount hardware and the other one an own blade hardware solution of the case company. Because these are standard solutions as such, there are no modifications planned to them in productisation sense.

6.1.2 Modularity and configuration

The highest-level system breakdown in the architecture of the platform is done to system components, which are grouped according to their purpose to a layered system model (see Figure 21). A system component covers some functionality area of the system. They can be hierarchical up to two levels and are collections of related subsystems (Prod_descr., p 39).

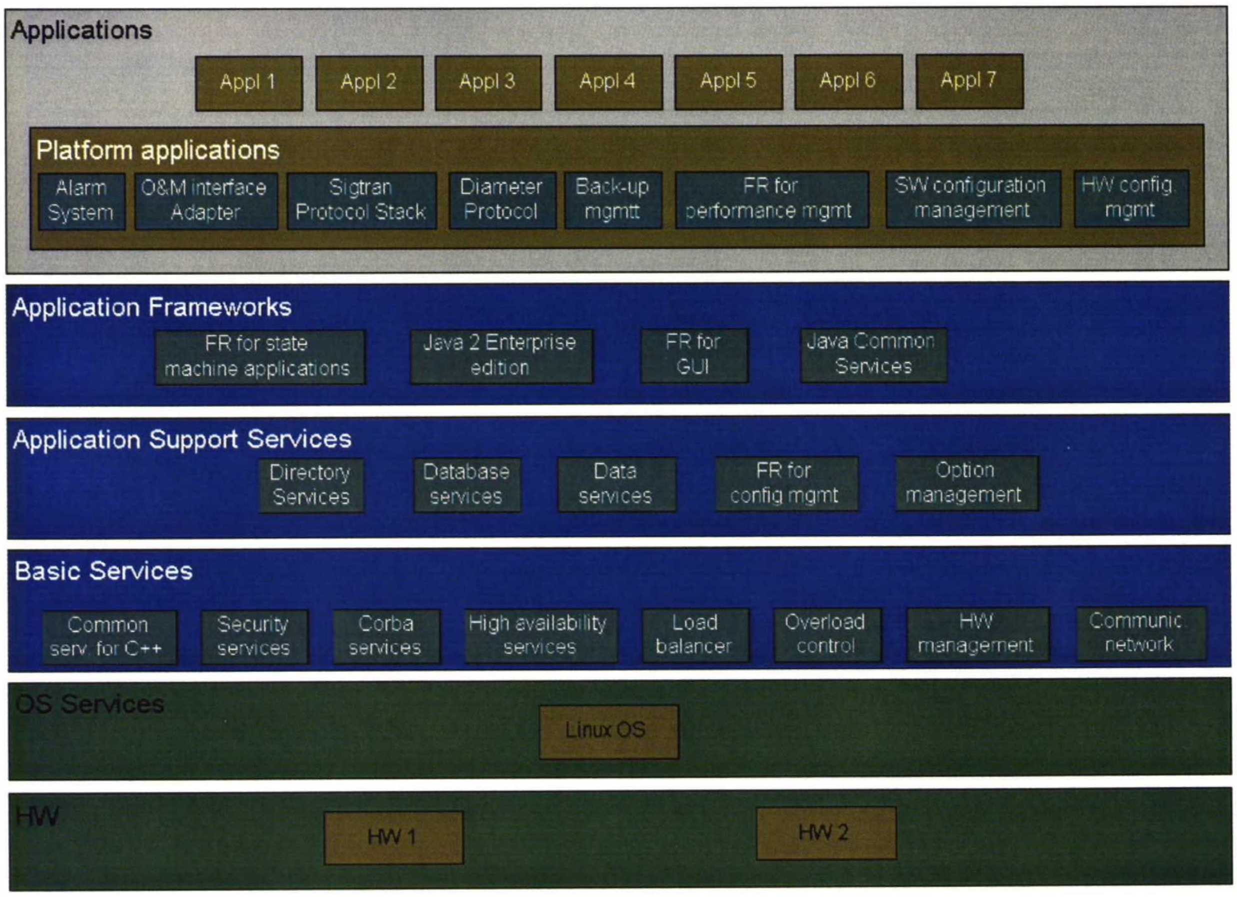

Figure 21: Layered system model of the server platform.

A subsystem is a large grain component that consists of components. A subsystem captures some related part of the functionality of a system component. A subsystem implements a number of integration interfaces, which is the only mechanism for other subsystems to use the services of the subsystem in question. A subsystem is independent of other subsystems during development time. It does depend on others but only indirectly via the integration interfaces. Both compile time and link time dependencies between subsystems are avoided to enable maximal parallelism in the development. Exceptions to this rule may exist. For example, a subsystem consisting of general-purpose services may be implemented as a library that is linked into other subsystems. A subsystem is also the unit of software that can be deployed, installed and upgraded separately into the system (Prod_descr., p 39 and Arch_princ., p 23).

A component is a process or a library. From the implementation point of view, it is a collection of classes and functions. From the software configuration management point of view, it is a list of source code files with versions that are linked together. The components can be used by several subsystems, for example as re-used libraries. Each component is still owned by one subsystem only (Prod_descr., p 39 and Func_and_arch., slide 8).

The Linux operating system, which is used in the platform, is itself an example of a modular system. Its key features are that it is free of charge and even more important, its source code is open. It means, that anyone can take its code and make modifications to it and thereby get the kind of system one wants. Linux is nowadays gaining some popularity also in embedded systems. Modifications can be done to it and then middleware and applications are built on top of it. Linux is not designed for a hard real-time use, which makes the modifications necessary (MITA, pp 89-91). As the case company is trying to establish an infrastructure around a platform using standard building blocks, it hopes that there will though not be modifications to the standard operating system versions.

The modularity of Linux also applies to its kernel modules. This is very useful for the designers of real-time embedded systems, since the kernel modules have a direct access to hardware devices. Open source and modularity gives the designers also the possibility to modify the kernel modules in order to get the best out of Linux in an embedded software environment. When implementing the modifications to Linux as patches to its kernel modules, their source code does not have to be published (MITA, p 91-93). The case organisation has used the possibility to modify the original kernel modules of Linux. It has made two additional kernel modules for the Linux version it is using. The first is a modification to the Stream Control Transmission Protocol (SCTP) implementation and it is delivered also to the original Linux distribution version manufacturer to be adapted as a part of their coming versions. The second one is an own IP-director kernel module, which the case company tries to protect by a patent, since it is an essential part of the product offering. One should notice that the former one is an open solution for other users of the Linux distribution also, whereas the latter one is a proprietary solution that is used to gain competitive advantage.

Right below the application layer in Figure 21 is the application frameworks layer. Frameworks do not exist as such in the final system, but applications can be created with them. A framework is a collection of libraries and implementation principles that can be used in the application development. A means of using frameworks is for example inheritance of the framework classes. The frameworks help in implementing the application basic logic and hide some of the data structures so that the developer does not have to make them him/herself. There are currently four application frameworks defined for the platform (Frameworks_and_tools, slides 3-4):

- State Machine Application Framework,
- J2EE Framework,
- Element Manager UI Framework,
- Java Common Services Framework.

A basic principle in a telecommunications system is that all updates to the system can be made with no or with minimal service disturbance. A software upgrade that does not cause disturbance requires that all internal and external interfaces of the upgraded entity are backwards compatible (Prod_descr., p 37). Applications based on the platform should be installed and commissioned with the same tools and following the same principles as far as possible. At the moment, all subsystems are delivered as binary files and are installed to the runtime environment of the system from rpm-packages (RedHat Package Manager) (Arch_princ., p 22).

The above-mentioned flexible architecture makes it possible to use only appropriate platform components in different business areas and therefore enables the implementation of various derivative products for different purposes (see Figure 22) (Prod_descr., p 38).

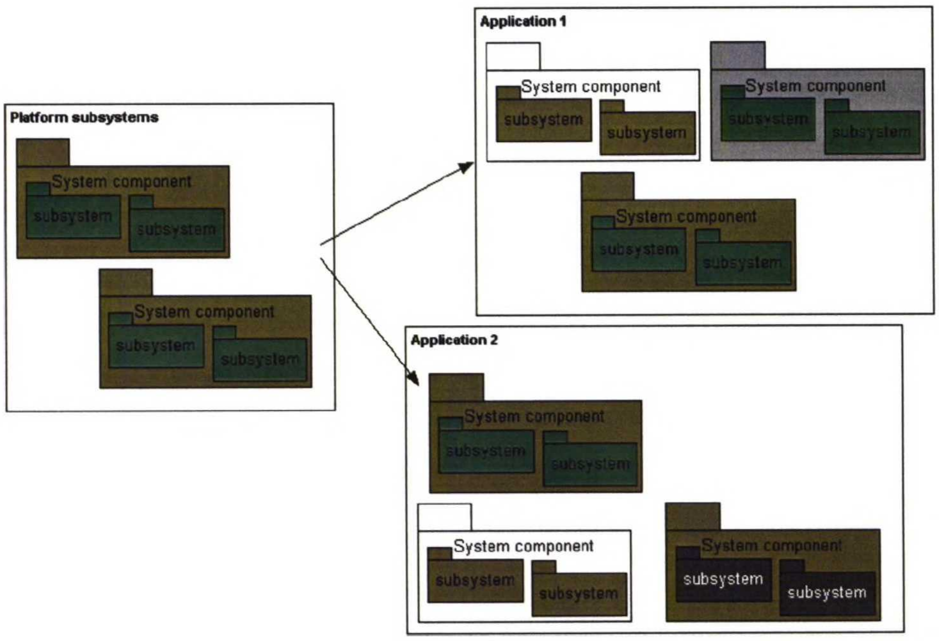

Figure 22: Product family approach from system components (Arch_princ., p 9).

With clear interfaces and a flexible structure, the components can evolve to be individual products themselves. The case organisation has so far defined three individual sellable components from the platform. The first one is the High Availability Services -system component, the second one is the application framework for state machine based applications and the third one is the implementation of the IP-director functionality. Dependencies of these entities to other system components are kept as minimal as possible. The high availability services are dependent only on the operating system and the CORBA services. The IP-director needs, besides the operating system, only the high availability services, the alarm system and a directory access. The framework for state machine based applications is implemented mostly using the basic operating system and CORBA services (Sell_comp., pp 10-32).

6.1.3 Interfaces

The subsystems of the platform in question have naturally interfaces to each other. They use services provided by others and offer their own services to them.

Dependencies between subsystems are defined as integration reference points (IRP) that are collections of related integration interfaces. An IRP has one or more subsystems as providers and one or more subsystems as users. A provider subsystem acts as the server and a user subsystem as the client in the service interface between them. Note that an IRP may contain interfaces where the server-client relationship is in both directions. In these situations, the user-provider relationship is defined by saying that the provider is the subsystem that acts mostly in the server side of the IRP (Prod_descr., p 39).

Compatibility of subsystems is defined via IRPs, which also have versions. Subsystems are compatible with each other, if they support at least one same version of an IRP that is used by both of them (Prod_descr., p 39).

6.1.4 Standards

Standardisation is an important issue in the productisation of a platform. The true value of a platform comes from its interfaces. The more familiar the users, i.e. the application developers, are with the existing standard interface, the more markets there are for a platform product.

In the server cluster implementation of the case company, each computer node using the platform is based on de-facto open industry standards and on components and building blocks provided by the leading players in the server industry to allow cost efficient and state-of-the-art performance. With open interfaces and components such as the Linux operating system, it is easier to integrate third party products into the platform (Prod_descr., pp 7,38).

An important standardisation party in the area of high availability platforms is the Service Availability Forum, whose task is to specify open high availability programming interfaces for carrier grade server platforms. The case company actively participates in the work of the forum. The forum has lots of promoters who are server, telecommunications and OEM vendors and users like Force, GoAhead, Hewlett Packard, IBM, Intel, Motorola, Nokia, Radisys and SUN. In addition, there are more than twenty contributing members like MontaVista, NEC, Samsung, Siemens and WindRiver (Means_doc., slide 3).

The APIs in the case organisation's platform are intended to be SA Forum –compliant as soon as the specifications are ready. At the moment, the case organisation is trying to get its solutions of the high availability platform in to the specification. The problem is that the specification is not ready yet, but the interfaces of the platform are already developed and internal customers have started to use them. The case company has a dual API -approach for the situation. The present APIs are supported as long as it is necessary for the customers and at the same time the SAF APIs are also supported.

The case company also drives the work of Open Source Development Lab (OSDL) Carrier Grade Linux Working Group (CGLWG), which specifies the carrier grade requirements for the Linux operating system. Several server vendors and users like Alcatel, Cisco, Hewlett Packard, IBM, Intel, MontaVista, Nokia, RedHat and SuSE take part in the work of the CGLWG.

The CGLWG defines a Linux Standard Base -compliant operating system with carrier grade extensions. Linux distributors are expected to include the CG Linux in their product offerings. (Func_and_arch., slide 21). The Linux distribution selected for the platform in the case company is an off-the-shelf Linux distribution, namely MontaVista Carrier Grade Edition, which is compliant with the OSDL Carrier Grade Linux Specification. The CGL-specification includes standards such as the Linux Standards Base (LSB), IPv6 (including IPSECv6 and MIPv6), SNMP support, and POSIX interfaces for timers, signals, message queues, semaphores, event logging, and threads. MontaVista

CGE Linux distribution is also compliant with the PCI Industrial Computer Manufacturers Group (PICMG) standard for hardware interfaces and shall be compliant with Service Availability Forum hardware platform API specification (CGLWG web page, 31.1.2003).

6.2 Internal customers of the platform

The platform in question is currently used only inside the case company. Although productisation of the platform has been an intention from the beginning of the development, it is still more a build of components integrated together and given to the internal customers to use. There is a similarity to the approach of Sipilä (1996) in the service sector, where he states that the products in expert organisations are base elements that are not even supposed to be sold. They are the building blocks, on top of which the actual product versions are customised case by case. Versions are results of the ability of the experts to quickly combine their own competencies and the established product model into an entity that provides superior benefits for the customer (Sipilä 1996, pp 46-47). That is in question also when a telecommunications platform is used in implementing network elements. The current organisation structure of the case company also supports the idea of Meyer and Lehnerd (1997), where the development of a platform and derivative products is clearly distinguished to minimise the interfaces between them (see Section 3.3.4). Then there is a minimal risk that changes in the platform automatically require changes in the derivative products, or vice versa (Meyer & Lehnerd, pp 126-127).

Recently, the case organisation has built up a new organisation unit for the horizontal OEM business. Its task is to act as the contact between the product development organisation and the horizontal customers. It has sales, marketing and customer support responsibilities. It contacts the potential customers and asks for their requirements for the platform. It also should act as the layer that makes the information anonymous between the horizontal customers and the platform development. Regarding the platform business, the new horizontal business organisation is seen just as a new internal customer for the platform (see Figure 23).

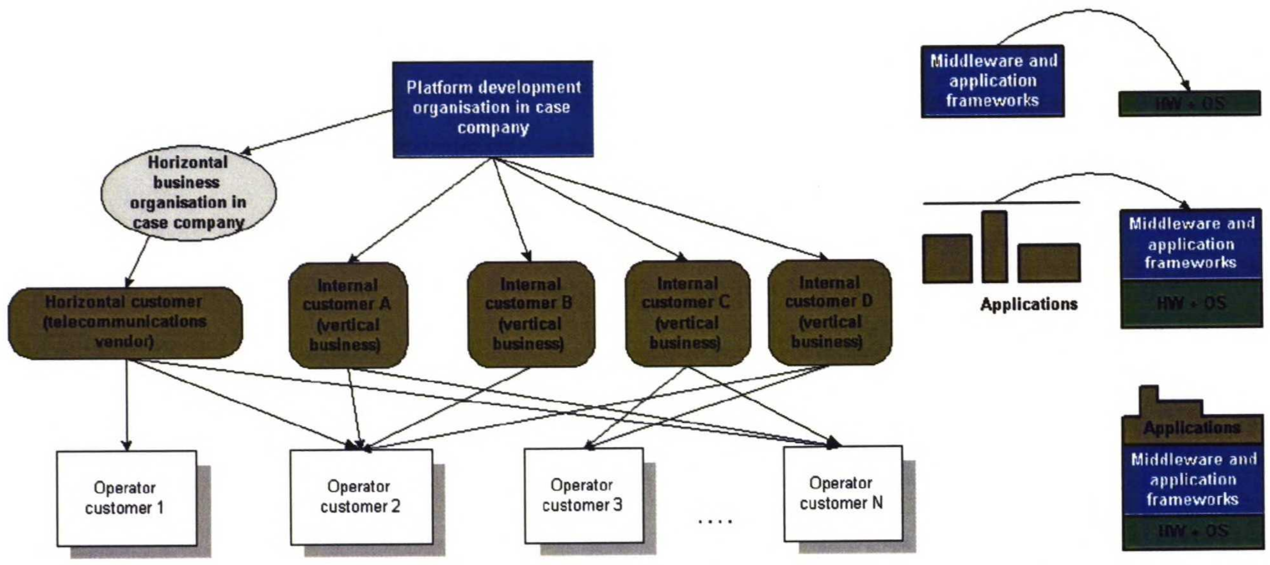
Platform development organisation in case company
Horizontal business organisation in case company
Horizontal customer (telecommunications vendor)
Internal customer A (vertical business)
Internal customer B (vertical business)
Internal customer C (vertical business)
Internal customer D (vertical business)
Operator customer 1
Operator customer 2
Operator customer 3
....
Operator customer N
Middleware and application frameworks
HW + OS
Applications
Middleware and application frameworks
HW + OS
Applications
Middleware and application frameworks
HW + OS

Figure 23: Platform development organisation, its internal customers and their customers.

Junttila stated that the tasks for the distribution organisation of a product include communication between the producer and the customer, negotiations regarding the sale, tasks related to the order process, taking the ownership and risk of the product during the delivery, physical distribution, billing and paying as well as transferring the ownership (Junttila, pp75-76).

The role of the new horizontal business organisation is to do the above-mentioned things, to represent the horizontal customers and deliver their requirements and thoughts to the platform development and to act as the sales and marketing organisation of the platform. Of course, the horizontal vendor is still a customer also for the platform development, like Junttila stated in Section 0 (Junttila, p 26).

Like said in Section 3.2.2, the internal business units of the case company already plan and develop derivative products on top of the platform in question. It can, in the marketing sense, point out the benefits of the platform to the horizontal customers. It requires that demo shows are arranged for the platform and that the internal business units are willing to share some of their experiences of the platform with the horizontal customers. Conflicts can occur if a successful application developed on top of the platform cannot be shown to a horizontal customer due to business reasons. This is of course opposite to the needs of the horizontal business unit.

It should be noticed, that the model described in Figure 23 is not symmetrical for horizontal and internal customers. The horizontal customers must be assured that the service given for them via the horizontal business unit is as good as is given to the internal customers. This relates, for example, to the feature request and problem reporting functions. Both the horizontal and the internal customers of the platform must be sure that no critical information of their application development goes through the platform development or the horizontal business organisation to their competitors. However, the internal customers of the platform have said that this is not of major concern. They stated that in the open architecture and standardisation -type of thinking, not any platform is something so special that its use cannot be revealed to a possible competitor (T.S., interview, 13.1.2003).

According to Seppänen, a product development process has two meanings. To produce quickly and economically new competitive products that fulfil the needs of the customers and to learn to do the above-mentioned things better and better every time (Seppänen, p 15). The latter is the fact in the idea of horizontal markets, which benefits also the internal customers that work in a vertical business environment. Current internal customers also think that the quality of the support services from the platform development organisation for them should increase as the platform is productised for external customers also (T.S. interview, 13.1.2003).

6.3 Current solutions for other than product functionality issues

In this section, an introduction is given to current solutions to various issues in the case organisation. The approach is to find and compare them to the factors that were presented in chapters 3, 4 and 5.

6.3.1 Documentation

Currently, documents related to the platform are generated in the platform R&D organisation. The target group for most of the documents is the application developers in internal customer organisations. Documents like installation and configuration guides are meant also for their laboratory personnel.

The platform development process defines that for every system component a developer's guide and an installation and configuration document should be written. Currently, depending on the subsystem, different above-mentioned issues may be implemented in different documents. For some subsystems, all of the mentioned documents are written separately while in some cases the information is combined to only one document.

The developer's guides describe the interface and the principles for how to use the services provided by the subsystem in the applications of the derivative products. They are rather simple documents, which are given to the application developers of the internal business units. Their customer documentation projects generate the final application documentation and use information from the developer's guides of the platform as a part of the documentation. The developer's guides are usually written in Microsoft Word –format and they are stored in an internal document database. The database is meant for the use of a platform product development program. It contains several different documents and all of them are not logically organised.

The designers write also subsystem software specification (SSS) –documents for the subsystems they are designing and implementing. The purpose of the documents is to clarify the internal architecture of a subsystem for problem solving and maintenance purposes. It also acts as a plan for the subsystem implementation work. The internal customers have an access to the SSS-documents, which helps their application development if the developer's guides are not ready yet or if they have flaws. This is often true, since the internal customers start their application development already when the platform subsystem implementation is ongoing and its developer's guide is not ready yet. One must also notice that the internal customers might currently have an access to the source code of the subsystems whereas only binary files of them shall be delivered to the horizontal customers.

In an upper level, there are also use case documents, which are produced with a special tool made for it and stored in Word-format to the database. The use cases are input documents for subsystem software specifications and they are also accessible for the application developers inside the company.

The presales marketing material of the platform currently consists of the product description and architecture documentation and a description of the sellable components. These documents have also been the basis for the introduction of the platform in this thesis (see Section 6.1 above). In addition, some new material for horizontal business has had to be written due to a customer demand. The presales documents are available for potential horizontal customers on a need-to-know basis. Public technical and marketing material of the platform is not yet available for example in the Internet.

Besides the fact that a different set of documents is produced from different subsystems, there are also a couple of other problems with the current documentation. The information regarding the IRP and subsystem dependencies and the services that can be used via a certain IRP is not documented in a consistent manner. In some cases, the information is in the developer's guides and in some cases it is in a separate document.

Another problem originates from references to company internal information. In several documents, there are references to case company internal documents, like the use case documentation and to internal contact persons. For example there can be certain architects or system designers mentioned by name. That kind of information cannot be left into the document versions that are given to the customers. The internal customers naturally have that information currently.

Similarly, the name of the case company and the platform is shown in many places in the documentation and the tools in customer documentation generate it automatically to some documents. There is a possibility that a horizontal customer uses parts of the documentation in its final customer documentation. As in those markets they might be competitors to the case company, having these kinds of references is a problem. Like discussed in Section 3.2.4, selecting and presenting a product name must be done carefully, especially when only a part of the product is sold to the horizontal markets.

Since the customers may want to use the documentation as a part of their final documentation, just as the platform is a part of their final products, there must also be clear disclaimers of the responsibilities of the case company. Current documents are produced for internal use and there is not a clear framing of the target group and not a disclaimer about responsibilities in them.

The documentation of the third party software and tools is somewhat more problematic. Currently they are not delivered to the vertical end customers, since they do not use the third party software for application development. Instead, only short guides are written if the software is such that the end customers use it directly. For example, some databases are only running on the background in the runtime system, the end users do not necessarily need any instructions on how to use them.

6.3.2 Interfaces and Application Programming Interfaces

The platform architecture is composed of four different layers and the system components in them (see Figure 21). The system components are compounded of subsystems, which are the smallest entities that can be deployed to a running system. The dependencies between subsystems are defined by the integration reference points (IRPs). An IRP is the way for the subsystems to communicate, i.e. to use the services of other subsystems and to provide services for them.

A principle is that a subsystem uses services only from its own layer or from the layers below it and provides services only to the own layer or the layers above. Because the dependencies are this way either horizontal or vertical, the IRPs can contain interaction that is internal for the platform or they can provide an interface also to the application layer. In this sense, the interfaces are either internal or public. The modularity and dependencies of the subsystems is discussed further in Section 6.3.6, where the product management function is introduced. The public interfaces are the Application Programming Interfaces (APIs) of the platform for the application developers. Some of the upper layer interfaces of the platform are the ones that shall be SAF-compliant gradually.

Currently, the API documentation of the platform is not public. Parts of it are given to potential horizontal customers on a need-to-know basis. The approach for publicity shall be redefined as the SAF-specifications are ready and the implementations of the interfaces are SAF-compliant. One must notice that the platform shall still contain interfaces that are not mentioned in the SAF-specifications.

6.3.3 Third party components and tools

There are several third party tools included and used in the platform in question. The idea behind using them has been the same as is the idea for other vendors to buy the platform. It can reduce own capital investments and development efforts in certain areas. However, although the horizontal markets productisation has been in mind from the early stages of the platform development, the model how the third party software is distributed to the horizontal markets has not been defined yet.

The problem is that the case company has currently OEM-type (original equipment manufacturer) contracts with the third party providers and those allow the case company only to sell them as parts of its own products, not to act as a reseller. That kind of a business would need contracts that allow the case company to act as a value added reseller (VAR).

There is also an observation related to the documents of the third party software. In a vertical business model, the end customers do not need most of the third party software documents. The platform development organisation has made simplified and summarised overview documents for the usage of these components, but not for development of software on top of them. The application developers of a horizontal customer may though need documentation that helps them in the application development, for example the original API-documentation.

The main third party components that are currently in use in the running system are:

- A Linux distribution version,
- A CORBA Object Request Broker (ORB),
- A database software solution,
- A J2EE application server,
- A signalling protocol stack.

6.3.4 Application development environment

The case organisation uses integrated software development environments for its product development. There are different environments meant for different types of platforms and different kinds of products, or parts of them. Mostly the division is done by the operating system used in the developed platform or product.

The tools in the integrated environments are made either especially for the case organisation or they are commercially available standard tools. The situation depends very much on the environment. In the platform in question, both the development and the target environments are running on the Linux operating system, but using different distribution versions of it. Partly due to this open operating system, the development environment uses commercially available standard tools. In the development of some earlier and more proprietary products, proprietary tools were used in the case company. In the current Linux development environment, the set of tools covers all phases of the software development process V-model. There are tools for architecture and system design, for the design and implementation of the software and for module testing (Frameworks_and_tools, slide 15).

The software development in the Linux environment is tightly integrated to the used software configuration management (SCM) in the case organisation. A widely used commercial SCM-tool has been selected and the developers of derivative products use it also, since they must have an interface to the platform software during development time.

Current solution for the development environment is that the tools and the managed software entities are installed to remote servers that the developers access through view servers (see Figure 24). The problem related to SCM originates just from this situation. The SCM assumes that certain files are found in certain locations. Removing the configuration of the environment attributes being dependent of the current infrastructure should not though require too much work from the environment support organisation (E.K, discussion 9.12.2002).

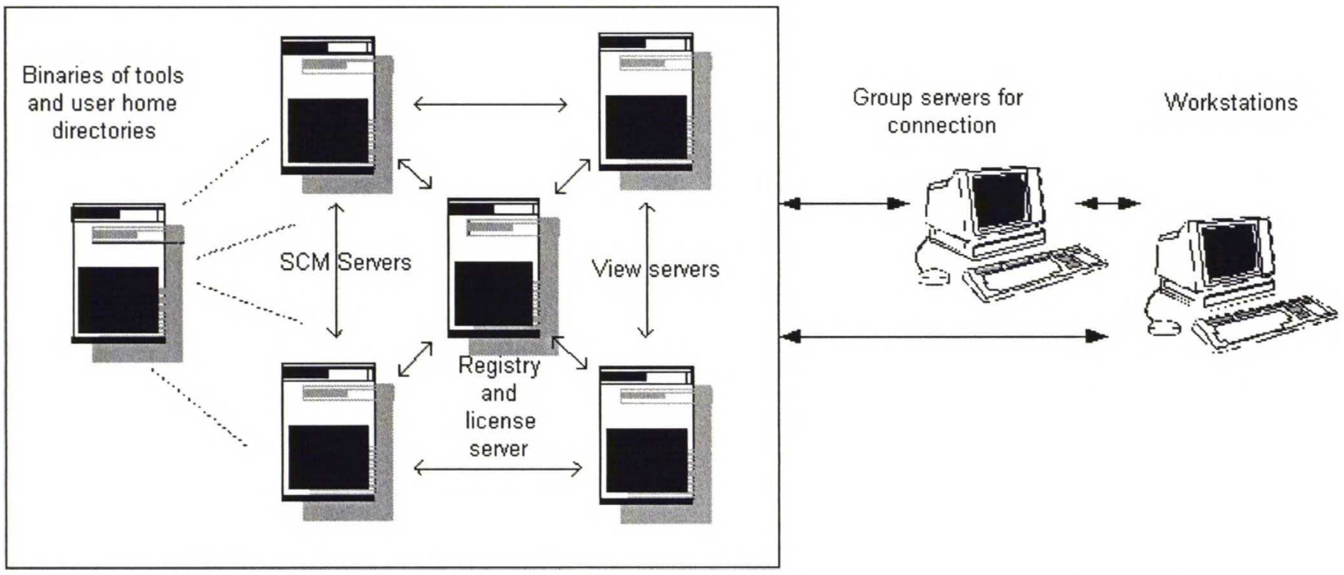
Binaries of tools and user home directories
SCM Servers
View servers
Registry and license server
Group servers for connection
Workstations

Figure 24: Deployment of the current software development environment in the case organisation.

The fact that there are some tools directly meant for certain third party software harms the idea of open environments and minimised dependencies between modules. Like we noticed in Chapter 5, providers of middleware components do include tools that help the usage of their software as a part of final implementations. This kind of a problem occurs with five tools in the current Linux development environment. Most obvious one relates to the cross compilers and other tools of the used Linux version itself, but the four other ones originate from real third party software in the runtime platform. Two of them are tools for database software, one is a set of tools for a Java application server and the fourth one is for implementing online-help textures in the graphical user interface of Java-applications in the final product.

Another observation is that there is actually not any decent presentation documentation of the development environment in the case company, except for some overview marketing slide sets. Reasons for this might be that the environment is integrated for internal use and there has been no need to market it outside the company. Internally, the developers learn to use the tools and the environment by doing or by asking from their colleagues or so-called key users. The horizontal business unit has noticed this and presented a requirement that there should be a description and an installation guide of the development environment, which can then be given to the customers (Horiz_req., p 3).

The same issue as with the third party components in the actual product occurs also with the third party tools in the development environment. The case company has licensed the use of those tools only inside its own organisation; it does not have the right to sell them to its customers. Being able to do that requires that the contracts are renegotiated and the case company can start to act as a value added reseller (VAR) for those tools. Selling tools provided by third party providers is though not any of the core business of the case company and it would be difficult to get any profits from selling the third party tools separately. There are two reasons for this. First, the horizontal customers probably already have a similar set of basic software development tools themselves and are not eager to learn new ones. Secondly, if they would like to change to third party tools that the case company offers, they could probably negotiate a similar contract for using them directly with the third party provider.

6.3.5 Services

As became clear in Section 6.2, the case company has existing telecommunications systems businesses in the vertical markets. The customers in those businesses are mainly telecommunications operators. The case company produces several different network elements and it has concentrated the customer services for them under one organisational unit.

The services that are provided via the customer care organisation include (Online_services_doc):

- Network implementation planning.
- Network deployment services, including project management, installation, multivendor integration and site acquisition.
- Maintain services, such as help desk, emergency support, software updates and upgrades, software installation and hardware spare parts support.
- Training services, including e-learning and customised learning solutions.
- Network operating and optimisation services.

There are special technical assistant teams worldwide that act as the first contact to the local customers and propagate their questions and requests to the right place in the case company.

The problem reports are managed in the organisation with a specific database tool made for it. Both the internal and external problem reports are handled with it. The technical assistance teams have an own tool for communicating with the final customers about the problems. They use the tool to report to the internal system (Means_doc, slide 66). The flow how a problem report and correction is managed is presented in Figure 25 on Page 83.

Like said in Section 4.1.4, a whole product can be a software build, which is distributed in electronic form. For that reason, it is natural that also the services for that kind of a product are delivered electronically. There is an online extranet solution in the case company for contacting with the customers. The tool requires a password for the users and it has a possibility to limit the access only to certain services, depending on the customer.

The services included in the customer care system are (Online_services_doc, slide 16):

- Product catalogues and electronic ordering,
- Product and technology introductions,
- Delivering software updates and upgrades,
- Documentation download system,
- Failure reporting interface,
- Technical note delivery,
- Help desk contact and browsing of earlier help requests,
- E-learning courses,
- E-mail notification service for individuals and whole customer company representative groups.

The benefit from the centralised online maintenance and documentation services is that what ever is put in there can be updated right after there is a need for it. This is not the case, for example, with a CD-ROM delivery (Online_services_doc, slides 9-10).

Installation and commissioning are important services in vertical telecommunications businesses, where big and complex systems are installed into use. In this horizontal case, the installation services would concern only putting up the software development environment to the infrastructure of the horizontal customer. It is also possible that they do not need any help with the installation, since they are probably capable of doing it themselves (see Section 6.3.4 above).

Training for the customers is currently arranged in many forms. Via the online services described above, the customers can attend e-learning courses. Traditional classroom trainings are also arranged in the regional training centres of the case company worldwide. For the platform development organisation, a more important mode of giving customer training is transferring the knowledge of the platform to the internal customers. There are so called product competence transfer solutions for these actions. Their main task is to ensure that the knowledge of the platform is transferred inside the case company. In simple words, it means delivering of all product related knowledge from the designers to the users. Teaching modes that are used in these competence transfers contain both classroom trainings and e-learning solutions. An external subcontractor is used in the production of the trainings but they naturally need consultation help from the product architects and developers to do the course materials (H.H. interview, 31.1.2003).

The horizontal business organisation has not yet implemented any customer care functions regarding the platform business, but it is expecting the results of this thesis also before taking any final actions. In a similar way, the platform development organisation has not any customer care functions in it, since the final customer support regarding the platforms nowadays is filtered through their internal customers (the product lines) and also through the centralised customer care organisation (see Figure 25).

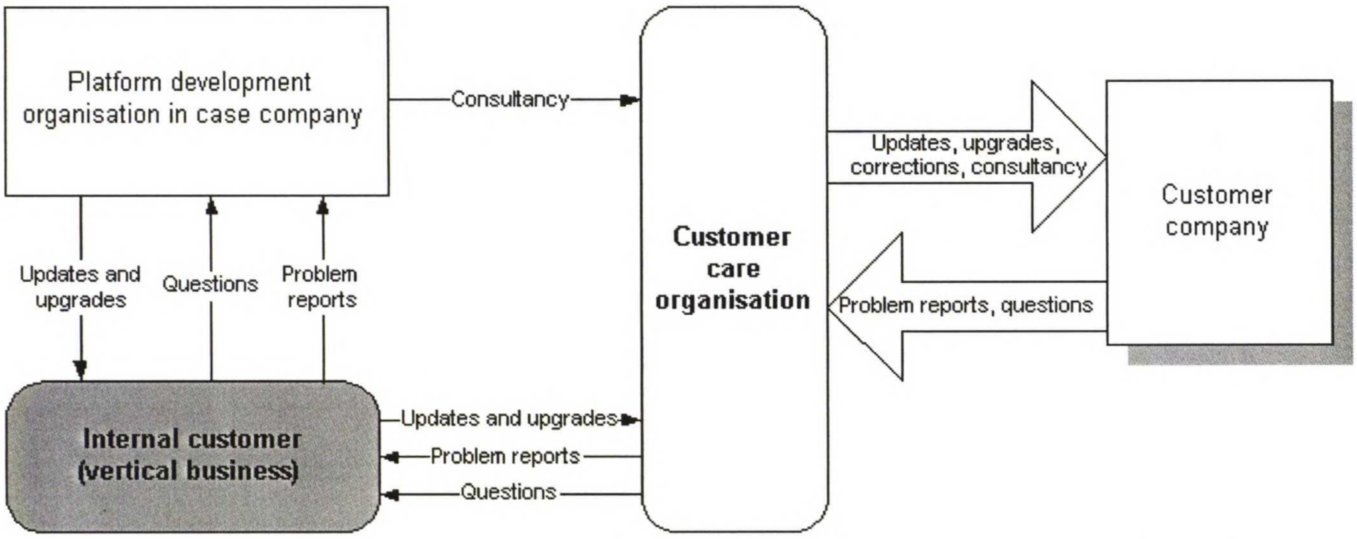
Platform development organisation in case company
Consultancy
Customer care organisation
Updates, upgrades, corrections, consultancy
Customer company
Problem reports, questions
Updates and upgrades
Questions
Problem reports
Internal customer (vertical business)
Updates and upgrades
Problem reports
Questions

Figure 25: Routes of customer support between platform development organisation and end customers in vertical model.

There are negotiations ongoing on how the customer care of the platform system is organised in the case of horizontal markets. Suggestions for that are given in Section 7.7.

6.3.6 Product management

Besides the product development, the platform development organisation in the case company contains also a product management function for different platforms. It is in charge of defining the platform requirements and functionality, as well as of roadmaps for the releases. The product management function has several product managers who do these things and act as account managers for the internal customers. For the horizontal markets case, one account manager shall take the responsibility for the relationship with the horizontal business unit (case company intranet, 24.1.2003).

The product management function makes suggestions and plans for the releases and the requirements that shall be implemented. The actual development work is then done in the product development departments and their product development projects.

Related to requirements, there have been some discussions of making separate variants of the platform for different application types, although any decisions about this have not been made. The way to achieve this customisation, and to make the variants, is to select a combination of system components and subsystems that are taken in to a configuration. The dependencies of subsystems from one another are defined via the integration reference points (IRP) that they use. According to some product managers, the documentation of these relationships is not yet as good as it should be. The subsystems cannot be defined to be sales items before their dependencies from other subsystems are clear. The information of which IRPs they use and which subsystems provide the services for them is inside the design document or the developer's guide of each subsystem. Exceptions are the three sellable components discussed in Section 6.1.2, where their dependencies were already explained. They are documented in the description document of the sellable components (Sell_comp, pp 10-32). However, the product management function in the case organisation has made plans about the sales items that can be left optional in the configuration. The problem is with the dependencies, which are hard to find out due to bad documentation (T.L and J.H, e-mail discussion, 23-27.1.2003).

One problem with the requirement management is that there are many requirements for the platform and that they are not properly categorised. Though there is a requirement database, there is no clear division between internal and customer originated requirements, for example. This phenomenon probably originates from the fact that the internal customers are not seen as real customers and their requirements are thought to be own discoveries of the case organisation. In addition, the application domain, the network control and management plane server platforms in all-IP networks, is such a new one that the requirements and especially their relative priorities are hard to categorise.

7. SUGGESTIONS

7.1 Productisation process

In Chapter 3 models for a productisation process were discussed. The things including into a general productisation process and that were presented in the summary of the chapter were:

- Creating product descriptions and clear user documentation,
- Creating documentation for further developers,
- Defining modularisation and versions of a product, also from internationalisation and localisation point of view,
- Defining the quality goals of the product, implementing quality assurance and planning its further development,
- Testing the existing product and its boundary values and performing integration testing,
- Creating copyright systems and license terms for a product,
- Creating services organisations and systems for
    - Receiving problem reports,
    - Distributing updates of the product,
    - Training of users and
    - Consultancy.
- Creating a product management organisation for handling the product configurations, requirements and services related to the product.

In the case company, a project manager should be nominated to the platform development organisation to take responsibility of the platform horizontal productisation project. The project manager should together with the product managements of the horizontal business and platform development organisations do a project plan about the activities related to the productisation. One should notice that the feature, capacity and other product functionality issues should not be taken in; they are handled in the normal product development projects.

Considering the problematic issues presented in Chapter 6, the parts that should be taken into the productisation project are:

- Documentation modifications and documents that have to be created,
- Redefining and renegotiating the contracts of the third party software and tools in the platform,
- Creating a software development environment that can be delivered to horizontal markets,
- Examining and documenting the APIs and making a roadmap for standard compliancy,
- Examining the module dependencies and defining sales items for different configurations,
- Defining and agreeing the model for the customer services for horizontal customers.

These are the main issues related to the productisation regarding the platform development organisation. Suggestions for handling them are given in following sections. To the productisation belong also other, more marketing related issues like pricing. Those are left out here, since they are business decision and should be made by the horizontal business unit. In addition, this is still not the optimal way for productisation, since here again an existing platform is being productised (see Section 3.2.1). However, after the above-mentioned issues are done, in the further development projects the platform can be treated as a complete product that is sold to both internal and external customers.

7.2 Documentation

Based on the current state of the documentation and the references from literature in Section 3.2.6, actions for the documentation part of the productisation project are presented below.

First, there must be made a complete list of what documents are produced and by whom. The list shall include a map of what documents are produced now and to which documents they have references. The most important documents for successful application development were asked from the internal customers of the platform. They identified the most relevant documents to be:

- An installation guide for the whole platform,
- An installation guide for the applications on top of the platform,
- A description of the runtime environment (the directory structure and the processes running in the system),
- The API documents and developer's guides of all IRPs and subsystems.

The documents that should be produced as a part of the productisation are:

- Pre-sales material for horizontal business organisation, which includes:
    - Product overview,
    - Detailed product descriptions,
    - Descriptions of separately sellable components,
    - Description of the development environment.
- Installation and configuration guides for the system and each subsystem,
- Developer’s guides for all subsystems,
- API descriptions for all IRPs,
- Development environment installation guide.

In the presales material, the material is generated to a general form from the internal architecture documents and product descriptions. The generalisation is done by the platform development organisation product management and the material is then given for finalisation to the horizontal business organisation. It must be kept in mind that the target group for the documentation is the application developers, not the final customers or users. Otherwise, the qualities for a good product description presented by Sipilä in Section 3.2.4 should be kept in mind.

For the developer’s guides the current template shall first be reviewed and then a modified version shall be made. The technical writers should then edit the current developer’s guides and make them compliant with the new template. The designers in the platform development shall review the changes. Things that should be included in the modification are:

- References for case company internal documents (such as system architecture designs or use cases) and persons are removed, either by removing them completely or by writing out the text that is behind the reference.
- The API descriptions (IRPs) are removed from the developer's guides and put to own documents.
- References to API description documents are put in developer's guides to replace the former IRP descriptions.

As a result, there shall be only one document containing the API description for a certain IRP.

Information sharing process for instructing the designers to start to use the new template shall be planned and organised. The internal customers benefit also from the unified documentation mode and the resulting new clear structure.

The installation and configuration guides shall be produced in the same way than the developer's guides. Technical writers modify the current documents made by the designers. The information in those documents shall answer to the following questions:

- How each subsystem is installed to the runtime environment?
- What are the related interfaces and components in the development environment?
- What other subsystems shall be installed in order for a subsystem to function right?
- What are the configurable parameters of the subsystem and how they can be changed?

The end user documentation is created by the horizontal customers themselves, though some help to them is given. Some end user documentation modules are currently done for the internal business units, who use them as parts of their own customer documentation. The format of these modules is XML, since they are easy to edit for the business units. They can leave things about the platform out or take them in and add their own documentation on top of it. The same raw modules should be given to the horizontal customers also. Before that, the case company name and logos must be removed from them as well as the links to documents that are not available for the horizontal customers. This task is rather mechanic and should not require too much resources from the technical writer group.

The format of other than end user documentation components should preferably be XML and they should be delivered on a CD-ROM and through the electronic customer care system to the customers. The documents should also be converted to PDF-format, since the PDFs can be delivered as read-only documents and they are more suitable for printing. The existing production processes for both formats of documentation shall be used.

The need of the horizontal customers for third party software documentation depends on whether the software is used only in the runtime environment or in the development environment also. In the latter case, the developer documentation has to be delivered to them. Depending on whether the case company delivers the software or if the customers buy those themselves (see Section 7.3 below), the documentation is also delivered through the same routes. If the case company delivers the software and there is very much documentation that an application developer probably does not need, the case company should create a compressed developer documentation in the same manner that it does for the end users in vertical cases.

7.3 Third party components and tools

Problems related to the third party software in the platform and the third party tools in the development environment were introduced in Section 6.3.3. The first action that has to be taken is that the most essential third party contracts shall be renegotiated and other ones left out of the product offering. If there are contracts that cannot be changed into VAR-type of contracts, a list of them with correct version information and with the contacts points, where they can be obtained shall be provided to the customers. This shall be done in any case, since the horizontal customers probably want to buy at least some of the development tools themselves, or they have them in use already.

The essential set of software, whose contracts have to be renegotiated or the version and contact information be given are:

- The two database software solutions of the platform,
- The Java application server software,
- The tool for making online help for Java applications.

Those are also the same, whose toolsets belong to the core set of the development environment. In addition to those, the core set contains also the cross compilers and other tools for the runtime environment of the used Linux distribution version.

The case company does not have to concern about other development environment tools in the first phase, unless the customers want to pay for the case company acting as a value added reseller for those also. That is not in the core strategies of the case company. In pure platform software, VAR-contracts for third party software are better than asking the customers to buy those themselves, since the customers can this way be provided a turnkey solution from a single point.

If the case company is able to renegotiate the contracts or if a horizontal customer buys them itself, the problems related to the third party software documentation are solved automatically. However, if the nature of the software were such that the horizontal customer probably needs to access only a limited functionality, a simplified compact user guide would still be useful.

7.4 Development environment

The software development environment is an important issue if not the most important one in this productisation case. The platform is planned to be sold to horizontal customers that develop applications on top of it. If there are no efficient tools for it, there is no use for the platform either.

As stated in Section 6.3.4, the case company uses a fully integrated software development environment. The environment consists of some obligatory tools that need to be installed in order to use certain third party software in the platform and to compile the applications to the selected hardware and operating system environment. The development environment needs to contain also the interface definitions for the platform services.

The first action to be taken is to define the core set of tools, i.e. the obligatory ones. Then there are two alternatives, depending on the selected third party software strategy (see Section 7.3 above):

1. If the third party tools can be distributed under a VAR-agreement, the core set of tools must be included into a separate development environment package.
2. If the third party tools are bought by the horizontal customer themselves, only a list of needed tools and their versions shall be delivered to the customers.

The second action related to the development environment is to provide the APIs to the developers. In practice, this means including all the IRP-files to the development environment package presented above. The easiest way to provide the development environment is a CD-ROM that contains all the needed components and an installation program or script for them.

To ensure customer satisfaction, successful installation of the development environment must be proved by testing. The installation shall be tested in a similar infrastructure that the customer has. If this is not possible due to a very special infrastructure, installation and configuration support services shall be offered separately.

The development environment package shall also include the documents of the tools and APIs that it contains. Developer's guides do not have to be there, but the API documentation and all installation, configuration and usage guides of the development environment shall be included.

7.5 Interfaces and standardisation

In order to offer an efficient application development environment for both the horizontal and the internal customers, the platform development must manage the interfaces of the platform. The subsystems in the platform interact through the integration reference points (IRP). The IRPs define the compatibility of subsystems. Applications in the application layer also access the platform subsystems via the IRPs.

Keeping the above-mentioned things in mind, the IRP relationships of subsystems shall be examined and documented. A hypertext documentation of them is already available. However, a responsible person has to be nominated to keep it up to date and a process build up for designers to report there any changes in IRPs or in the subsystems that use them.

The case company must also make a decision about the publicity of the APIs. After the APIs are documented in the above-described way, the selected customers can be given an access through the electronic customer care system to see the API documentation. The selected customers can include parties with whom there are negotiations ongoing and who have signed a non-disclosure agreement.

Managing of the interfaces relates to the ongoing Service Availability Forum and other specification processes and to the fact that the case company has announced its platform to fulfil those specifications gradually. When an interface management system has been implemented, it is easier to start the work of making them standard-compliant.

The case company naturally follows and drives the specification work in SAF and other forums. As the schedules of the specification work are known, a roadmap plan for the current interfaces evolution to be compliant with the specifications shall be done. The roadmap shall naturally also be taken into consideration when planning the product development projects, since it probably causes some extra work.

Part of the commitment to the SA Forum and other standards is clarifying it to the customers. As a part of the product descriptions and marketing materials, the customers shall be explained the technical and business reasons why each standard is fulfilled and what is the benefit for them to also commit to it.

7.6 Modularity, configurations and customisation

Ran said in his article that the reason why componentware does not mostly succeed is that the individual subsystems must still take the overall system requirements and restrictions into account (Ran, p 164). It should be noticed here that the real benefit for the customers from buying a productised platform is that the case company has done the integration of the standard based off-the-shelf components into a functioning entity. The complexities that Ran talks about have already been solved for them.

However, as the subsystem dependencies from IRPs are clarified (see Section 7.5 above), also the services that each subsystem provides for others via the IRP shall be identified, documented and managed. The reason for this is the customisation that shall be possible to do via modularisation. The customers should be offered with the possibility to leave some subsystems out of the configuration. The sales items shall be formed from the system components taking into account the dependencies to other system components.

The product data management (PDM) responsible people in the platform development organisation product management shall be responsible for keeping the dependencies up to date. This is possible with the help of well-documented and managed relationships of subsystems (see Section 7.5 above). The definition work of the dependencies and sales items relates to the intention of the case company to define sellable components from certain system components. Like explained in Section 6.1.2, the dependencies of these kinds of entities must be kept minimal.

7.7 Customer services

The customer services are an important, though not very well recognised, part of a product offering. They are usually taken for granted, but providing those is in reality a tremendous effort made in addition to the actual product functionality development and delivery.

In general, the case company has already an established and functioning customer support organisation and systems for it. As said in Section 6.3.5, there are ongoing discussions on whether it should be used or not in this case. It would be reasonable to use the existing resources, while still taking into account the special needs of the horizontal business.

The electronic customer care system in the Internet should be used as the contact point for the horizontal customers to the case company. Routine issues like delivering documents, handling of the problem reports, customer questions and feature requests can be handled there. It can also be used to delivering software upgrades and updates according to the selected release policy (see Section 3.3.1). E-learning courses can also be distributed via the system. The customer care organisation should nominate a team that familiarises with the needs of horizontal business. The most important issue for them is to manage the access of horizontal customers to only selected parts of the extranet system. In Figure 26, a suggestion for the overall arrangements for the customer support is presented.

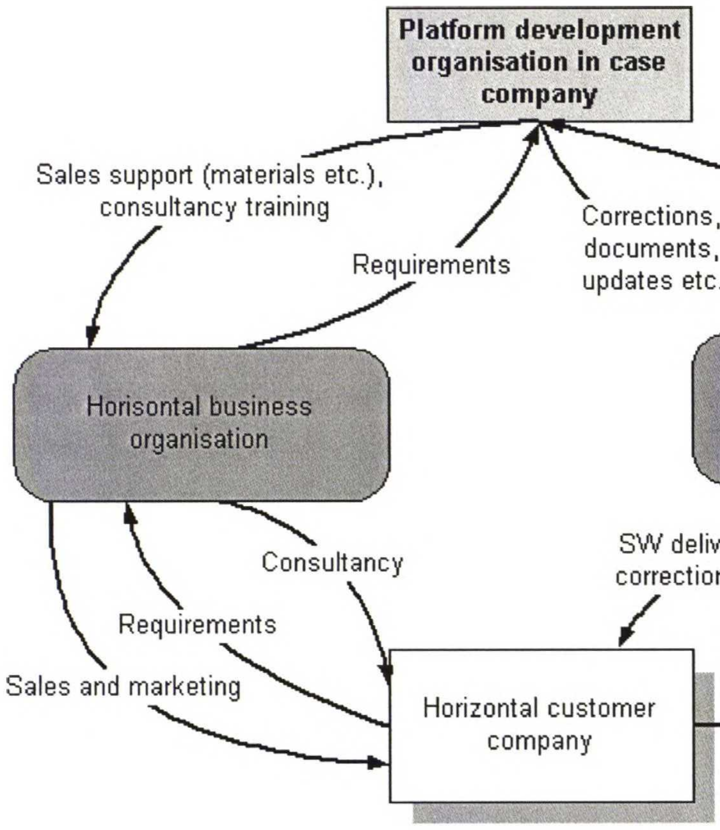
Platform development organisation in case company
Sales support (materials etc.), consultancy training
Requirements
Corrections, documents, updates etc.
Problem reports, questions
Horisontal business organisation
Customer care organisation and the electronic system
Consultancy
SW deliveries, corrections etc.
Requirements
Sales and marketing
Horizontal customer company
Problem reports, questions etc.

Figure 26: Organising customer support with existing organisations and electronic systems.

The technical assistance teams in the customer care organisation can have a direct contact to the platform development in routine issues like problem reports or document and software updates. A special horizontal business account manager should still be nominated to the platform development organisation to handle, for example, the requirement management process and consultancy training for the horizontal business organisation. In addition, the horizontal business organisation should have an account manager for each separate customer to oversee the quality of the customer care service.

The role of the horizontal business unit and the customer care organisation is to act as the layer that makes the information shared between the platform development and the horizontal customers anonymous. That way there is no problem of people working with the issues of a horizontal customer to work also with problems of the internal customers. Therefore, there is no need for distinguishing the platform developers according to different customers, a method that is often called the Chinese wall – principle.

7.7.1 Training

In training issues, the focus is in the product competence training that is currently offered inside the case company. The existing set of training should be adequate, since the target group is similar than inside the company, the developers of applications on top of the platform. The provider shall still be the platform development environment, since it has the knowledge of the area. The actions that they should take of are:

- Generalisation of current courses and their material (removing references to company internal sources and information),
- Making the courses available to the customers through the existing systems and processes of the customer care organisation.

One must notice that compared for example to the IBM WebSphere, the customer base of a telecommunications platform will probably be a lot smaller. The amount of potential customer companies is less than ten, at least in the first phase, and the number of developers inside each of them is only a few hundreds or a couple of thousands.

A part of the training is to continue the architecture and functionality overview courses given to the distribution and the customer care organisations. Namely, to the horizontal business organisation and the technical assistance teams in the customer care organisation.

7.7.2 Consultation services

The hardest part in arranging the support is the consultation. Installation services for the development environment and help in setting up the runtime environment of the platform can be provided by the horizontal business organisation after a technical consultancy group has been nominated there and they have been trained properly. However, consultation related to efficient application development is problematic, since it needs special technical and architectural knowledge about the platform. In addition, having only one or two consultants would violate the Chinese wall principle between different horizontal customers. Therefore, a separated consultant or a consultant group should be nominated for each customer.

A solution for this situation is that the consultation support is divided into three phases (see Figure 27). In the first phase, for the first customers, a technical project or product manager is nominated to the platform development organisation. He or she gives the consultation support for the customer and at the same time trains the technical consultancy teams in the horizontal business organisation.

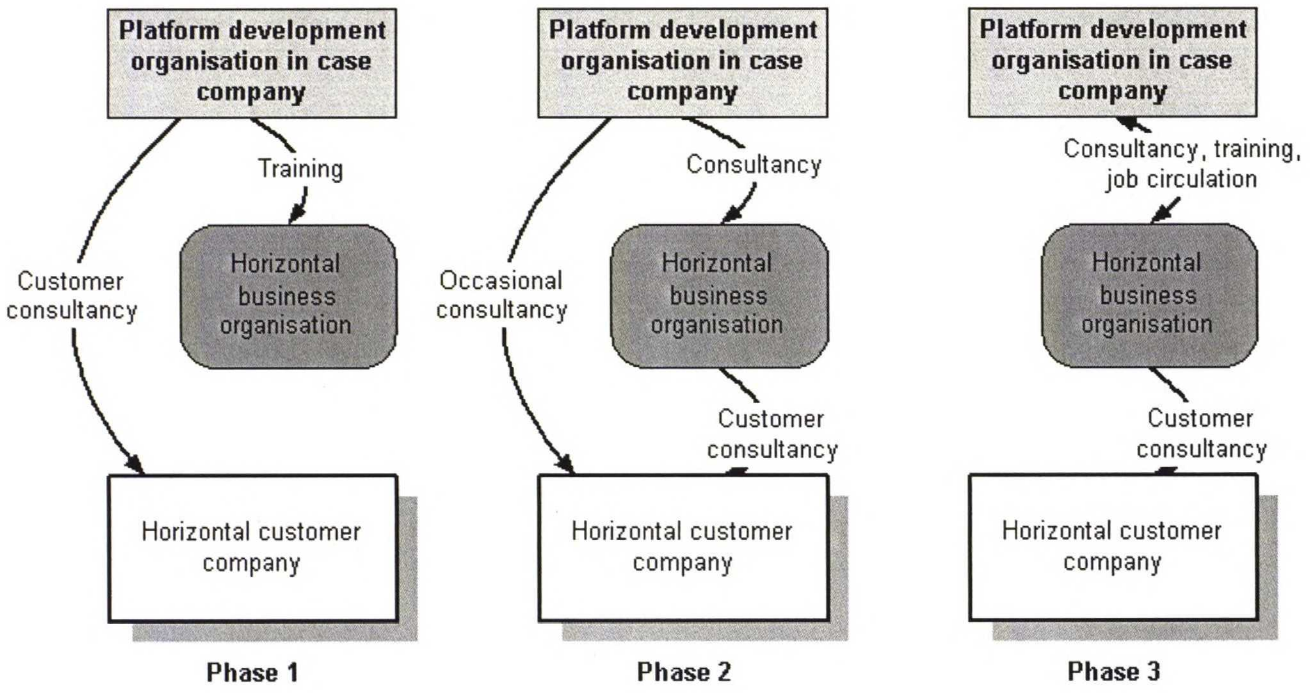
Platform development organisation in case company
Training
Customer consultancy
Horizontal business organisation
Horizontal customer company
Phase 1
Platform development organisation in case company
Consultancy
Occasional consultancy
Horizontal business organisation
Customer consultancy
Horizontal customer company
Phase 2
Platform development organisation in case company
Consultancy, training, job circulation
Horizontal business organisation
Customer consultancy
Horizontal customer company
Phase 3

Figure 27: Phasing of consultation services provision for horizontal customers.

In the second phase, the consultation is provided by the technical consultant teams themselves, and the technical product manager helps only if they cannot solve some problems. Gradually, in the third phase, all consultation is given by the technical consultant teams and the product manager propagates and trains only the new features to them. In this phase, it would also be useful to occasionally circulate the technical consultants in system testing and platform product development positions in order for them to fully understand the technology of the platform.

7.8 Summary

A summary of the suggestions presented in previous Sections 7.2-7.7 is gathered in Table 4 below. The suggested actions, the main benefits and reasons for them are listed.

Table 4: Summary of the presented suggestion for the case company.

Suggested action	Main reasons and benefits
Modifications to the documentation	Current documentation contains too much internal information.
Third party software renegotiations	Case company shall be able to act as a value added reseller.
Development environment modification	A requirement for effective application development. The current environment shall not be delivered as such to an external customer.
A roadmap for the API definitions and standardisation	Documented APIs are needed for effective application development. Standard-compliancy is a basic requirement in the platform business.
Module dependency definition	Requirement for modularity. Enables building of product variants and separately sellable components.
Organising customer services	Basic need for a product sold to external customers.

8. APPLICABILITY ANALYSIS

This thesis was done as a feasibility study related to the productisation of the telecommunications platform in the case company. In parallel with the study, there were discussions ongoing with the horizontal business unit about the requirements for the product offering. The horizontal business unit has derived them by filtering the information they have got from the potential horizontal customers. This study combines that information with the internal thoughts of the platform development organisation and the horizontal business unit. Appendix A lists the requirements that the horizontal business unit has presented to the platform development organisation.

According to Jokinen, after a solution or solutions are selected, it must be tested against given conditions. There are three different analyses that should be done (Jokinen, pp 86-87):

1. Inconveniences analysis. Evaluating if presented solution has some negative or positive side effects that are not yet taken into account.
2. Sensitivity analysis. It is estimated how easily the preference order of solutions is changed if some weights or grades are changed a little. In addition, factors that most probably will affect the order are presented.
3. Potential problems analysis considers problems that are currently not probable but would have considerable effect if they mature.

Chosen solutions must also be approved by some party and after that implemented. Approval contains two issues. First, the right people must approve the proposal before it can be implemented. Secondly, the developer and presenter of the solution must himself be sure of its viability (Jokinen, p 87).

The applicability of the suggestions and their expected resource need presented in Chapter 6 was asked from a group of managers and experts in organisations involved. The group consisted of:

- The general manager of the horizontal business unit,
- The Head of Platform Product Management in the horizontal business unit,
- The Head of Platform Business program in the platform development organisation,
- The Director of Product Business Management in the platform development organisation,
- A product manager in the platform development organisation,
- An Information Design Technology expert in the platform development organisation,
- The Head of platform the product development organisation and
- A Chief R&D Engineer in the software engineering environment development organisation.

The group was asked the following questions:

1. Are the presented suggestions reasonable and relevant?
2. Do the presented suggestions contain side effects that were not taken into account?

The relevancy of the suggestions was perceived good. The horizontal business organisation reminded that most of the suggestions are already on their action lists. The documentation people said that the suggested formats and processes for the documentation are preferable for them.

The inspectors mentioned a couple of issues that were not taken into consideration well enough in the suggestions:

- The removal of case company logos from documentation is not a problem to the document writers, since the logos come from the tool that converts the source files into final documentation.
- The possible conflicts between vertical in-house customers and horizontal business is not analysed much. The horizontal business unit expects that there will be different viewpoints regarding requirements on features, for example.
- The confidentiality between vertical in-house customers and horizontal customers as well as between different horizontal customers is not analysed in the thesis in detail.
- The development environment is hard to test in the case company, if the infrastructure of the horizontal customer is much different. Especially this concerns situations, where the horizontal customer buys and installs the third party components itself.
- It is still unclear, how independent the platform and the development environment are, for example, from the selected database software. It has not been studied whether a substitute can be used.

From the results it can be seen, that the suggested actions are relevant but some details are still problematic. There are alternatives for the implementation of the actions, but not any severe problems have been pointed out. The findings are questions that have to be answered, but they do not prevent the platform productisation.

9. CONCLUSIONS AND FURTHER STUDY

In this thesis, there were three main themes and keywords: productisation, telecommunications platforms and horizontal markets. A summary and a short description of them each is presented below.

Productisation means packaging the knowledge and technology of a company in to sellable entities. Productisation should be in mind from the beginning of product definition and requirement analysis. It includes creating of the product descriptions, user documentation and documentation for further developers. It also defines modularisation and versions of the product, defining the quality goals of the product, implementing quality assurance and testing the existing product and its boundary values. Furthermore, creating copyright systems and license terms for a product as well as organisations and systems for services are defined as a part of productisation. It also includes creating of a product management organisation for handling the product requirements and configurations.

A telecommunications platform contains the hardware and operating system and a collection of subsystems that take care of the basic functionality related to high availability, data management and communication protocols. It offers application programming interfaces to application developers of a variety of products and makes the underlying hardware solutions, clustering principles and high availability functionality invisible for the applications.

The horizontal markets approach means that a company can buy and sell parts of its product portfolio from or to its peers in the industry environment. Namely, it means that it can sell its technology to be a part of products made by its possible competitors. In this type of business, the company does not have an interface to the final customer of the product. The type of business also allows smaller infrastructure and application providers to enter the formerly vertical business, which used only proprietary systems. An essential part of this kind of a business is that the physical and logical interfaces of components and platforms are compliant with industry-wide standards.

From the insight into two platform-type of products in the IT and telecommunications segment, an overview of the required issues related to these products was provided in Chapter 5. The main things found out there were:

- The purpose and target group of a product is defined, but room for customisation is left in.
- The interfaces and APIs are defined and they are either public or available for a potential customer.
- The interfaces and APIs follow or are planned to follow an industry standard.
- Usage of APIs is efficiently documented.
- Emphasis is put on post sales services, especially on installation and consultation services.

The current platform that is used to develop end products inside the case company was presented in Chapter 6. The main problems in the current situation were found out there and suggestions for corrective actions were presented in Chapter 7. The suggestions on how to solve the gap between the present situation and the requirements for a telecommunications platform product were:

- Documentation modifications and documents that have to be created,
- Redefining and –negotiating the contracts of the third party software and tools,
- Creating a development environment that can be delivered to horizontal customers,
- Examining and documenting the APIs and making a roadmap for standard compliancy,
- Examining module dependencies and defining sales items for different configurations,
- Defining and agreeing on the model of customer services for horizontal customers.

The applicability analysis in Chapter 8 showed that the suggestions given in this thesis are applicable to the case company. Some things that are not clear yet and need still additional decisions were also noticed. Partly based on the suggestions given, the horizontal business organisation has generated a list of requirements for a productisation project (see Appendix A). The customer for the project is the horizontal business organisation.

As this thesis studied a special case related to three problem fields (horizontal markets, productisation and telecommunications platforms), further studies could be made related to these issues separately and in more detail. Especially interesting would be studies about the intellectual property rights (IPRs) and branding in the horizontal business and about the standardisation work of telecommunications platforms in general (Service Availability Forum interfaces and others). A further study should also be made on the horizontal business value chain and the process in it. The individual tasks described in this thesis could be used as the building blocks of the process phases if the relationships of them are explained in the process. A thing that was on purpose left out from this thesis, analysing possible new customer groups, is also worth studying. New customer groups could be found, for example, from the final customers, i.e. the operators that have their own application development as well as from application service providers and Internet service providers.

REFERENCES

Literature

Cooper, Robert G. Edgett, Scott J. 1999. Product Development for the Service Sector, Lessons from market leaders. Perseus Books, Cambridge, Massachusetts, USA.

European Commission. 1997. Competition in telecommunications: Why and how? Office for Official Publications. Luxembourg.

Fabricius, Frank. 1998. The Future Scenario for Telecommunications Vendors. Dataquest Vendor Analysis

Gartner Dataquest. 2001. Server market predictions for 2002. Gartner, Inc. 11.12.2001

Haikala, Ilkka. Märijärvi, Jukka. 2000. Ohjelmistotuotanto. Satku – Kauppakaari Oyj, Helsinki

Hirsjärvi, Sirkka. Remes, Pirkko. Sajavaara, Paula. 1998. Tutki ja kirjoita. Kirjayhtymä Oy, Helsinki.

Hoch, Detlev J. Roeding, Cyriac R. Purkert, Gert. Lindner, Sandro K. Müller, Ralph. 2000. Secrets of Software Success. Management Insights from 100 Software Firms around the World. Harvard Business School Press. Boston, Massachusetts.

Jokinen, Tapani. 2001. Tuotekehitys. Otatieto. Oy Yliopistokustannus University Press, Helsinki.

Junttila, Merja. 1998. Kansainvälistyvän ohjelmistoyrityksen tuotteistamiskäsikirja. Pro Gradu tutkielma. Oulun Yliopisto, Tietojenkäsittelyopin laitos. Oulun yliopisto, Täydennyskoulutuskeskuksen julkaisuja 19.

Kitcho, Catherine. 1998. High Tech Product Launch. Pele publications, Mountain View, California, USA

Kotler, Philip. Armstrong, Gary. Saunders, John. Wong, Veronica. 1999. Principles of Marketing. Second European Edition. Prentice Hall Europe.

Kuusikoski, Jari. 1996. Verkonhallintapalvelun tuotteistaminen ICL Data Oy:ssä. Master thesis. Tampere University of Technology.

Lahtinen Yrjö (ed.) 1995. Tuotteistus ja kustannuslaskenta. Suomen Kuntaliitto. Helsinki.

Lillrank, Paul and Forssén, Minna (ed.) 1998. Managing for knowledge, Perspectives and prospects, Helsinki University of Technology. Industrial Management and Work and Organizational Psychology. Working Paper No 17.

Mattila, Kirsi. 2000. Tuotteistaminen asiantuntijaorganisaatiossa, Tapaus Sedecon Consulting ja valmennuksen mittaus tuotteena. Pro gradu. Helsingin yliopisto, Valtiotieteellinen tiedekunta, yleisen valtio-opin laitos.

Meyer, Marc H. Lehnerd, Alvin P. 1997. The power of product platforms: building value and cost leadership. Free Press, New York.

Natis, Y. 2002. Sun's Software Strategy: A High-Stakes Balancing Act. Research Note 6 June 2002. Gartner Research.

MITA. 2001. Mobile Internet Technical Architecture (MITA). IT Press. Finland

Nybom Carita, Orkoneva Marjaana and Siltala Soile. 1996. Tietopalvelun tuotteistaminen. Helsinki University of Technology, Lifelong Learning Institute Dipoli. Info 1996:8.

O'Grady, Peter. 1999. The Age of Modularity, Using the new world of modular products to revolutionize your corporation. Adams and Steele Publishers, USA

Olkkonen, Tauno. 1993. Johdatus teollisuustalouden tutkimustyöhön. Helsinki University of Technology, Industrial Economics and Work and Organizational Psychology, Report No 152.

Ran, Alexander. 1999. Software Isn't Build Form Lego Blocks. ACM, Symbosium on Software Reusability SSR '99, pp 164-169, Los Angeles, USA.

SA Forum. 2001. What is the Service Availability Solution? Service Availability Forum white paper.

Sallinen, Sari and Seppänen, Veikko. 2002. Secrets of Software Success. About the book and software industry in general. Ohjelmistoliiketoiminnan case-kurssi, Oulun Yliopisto/TOL. http://www.tol.oulu.fi/opiskelijat/sw_success_kalvot_2002.pdf on 28.6.2002

Seppänen Anssi. 2002. Tuotekehitysprosessin yhtenäistäminen yritysfuusion jälkeen. Master thesis. Helsinki University of Technology

Tuominen, Kari. 1999. Muutoshallinnan mestari, Kuinka toteuttaa strategiset suunnitelmat kilpailijoita nopeammin. Laatukeskus.

Sipilä, Jorma. 1992. Asiantuntijapalvelun markkinointi. Weilin+Göös. Espoo.

Sipilä, Jorma. 1996. Asiantuntijapalvelujen tuotteistaminen. WSOY. Porvoo.

Sommerville, Ian. 1995. Software Engineering. Fifth edition. Addison Wesley Publishers Ltd. Harlow, Essex, England.

Tiihonen I.: Tiihonen, Juha et al. 1998. Concepts for Modelling Configurable Products. TAI Research Centre and Laboratory of Information Processing Science, Helsinki University of Technology. TKO-B139.

Tiihonen II:. Tiihonen, Juha et al. 1998. Towards a General Ontology of Configuration. TAI Research Centre and Laboratory of Information Processing Science. Product Data Management Group. Helsinki University of Technology. TKO-B141.

Vaattovaara, Matti. 1999. Transforming services into products. Doctoral dissertation. Helsinki University of Technology, Industrial Economics and Work and Organizational Psychology, Report No 9.

Vainio Markku. 1999. DX200 MTX-Ci - matkapuhelinkeskuksen mallintaminen, tuotteistaminen, laajennusten ohjeistaminen sekä testaussuunnittelu. Master thesis. Helsinki University of Technology.

Vitikainen, Katriina. 1995. Ohjelmistotuotteiden tuotteistaminen. Master thesis. Lappeenranta University of Technology.

Wainewright, Phil. 2002. Web Services Infrastructure. The global utility for real-time business. White paper by Phil Wainewright. http://www.philwainewright.com/pubs/wp/WSIpaper.pdf 2.9.2002.

Webb, William. 2001. The Future of Wireless Communications. Artech House, Norwood.

Weiss G. 2002. How to Make Sense of a Multi-OS, Heterogeneous World. Gartner Research.

FSC (1), 2002. Resilient Telco platform (RTP): Continuous Service through highly available Applications. http://extranet2.fujitsu-siemens.com/vil/oec/vil/us000000/i12138/i12139/i19849.pdf 8.1.2003.

FSC (2), 2002. RTP- Resilient Telco Platform, Customer Value. Material got from a contact in FSC: Flyer.customer Value v02.pdf. 8.1.2003

FSC (3), 2002. RTP- Resilient Telco Platform, Continuous Application Availability. Material got from a contact in FSC: Flyer.advert.v02.pdf. 8.1.2003.

FSC (4), 2001. RTP: Resilient Telco Platform, the optimal way to implement zero downtime applications. White paper, Fujitsu Siemens Computers GmbH, 2001. http://www.fujitsu-siemens.com/rl/products/software/rtp-wp-010309.pdf 8.1.2003

FSC (5), 2002. RTP – Resilient Telco Platform V2.0 on Linux. Technical Backgrounder. Material got from a contact in FSC: Flyer.dataSheet.v06l.orginal.pdf. 19.1.2003.

Internet sources

Clarke Roger. 1997. Electronic Commerce Definitions.
http://www.anu.edu.au/people/Roger.Clarke/EC/ECDefns.hmtl, 24.06.2002.

PDMA-glossary Product Development and Management Association glossary
www.pdma.org/glossary.html, 15.11.2002

IBM-web (1) IBM WebSphere Application Server, Foundation and tools description
http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv, 14.10.2002

IBM-web (2) IBM WebSphere Application Server, Version 5.0
ftp://ftp.software.ibm.com/software/webserver/appserv/v5/g325-2047-003.pdf 16.1.2003

IBM-web (3) Definition of different scenarios for WebSphere usage
http://www-3.ibm.com/software/webservers/appserv/scenario.html 14.10.2002

IBM-web (4) Adapters
http://www-3.ibm.com/software/ts/mqseries/adapter/ws/ 14.10.2002

IBM-web (5) Community
http://www7b.software.ibm.com/wsdd/community/ 14.10.2002

IBM-web (6) Education
http://www7b.software.ibm.com/wsdd/education/ 14.10.2002

IBM-web (7) Partner world and newsletters
http://www.developer.ibm.com/ 14.10.2002

IBM-web (8) Services
http://www7b.software.ibm.com/wsdd/services/services.html#appsvr 9.1.2003

IBM-web (9) IBM WebSphere Studio
http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/studio 22.10.2002

IBM-web (10) Telecom application server toolkit
http://www-3.ibm.com/software/devcon/devcon/docs/wta120wu.htm 22.10.2002

SUN-web SUN Microsystems J2EE API documentation
http://java.sun.com/apis.html#j2ee 4.11.2002

FSC-web (1) Fujitsu Siemens Computers web page
http://www.fujitsu-siemens.com/rl/products/software/rtp4.html 8.1.2003

FSC-web (2) Press release (28.5.2002) in Fujitsu Siemens Computers web page
http://www.fujitsu-siemens.com/rl/products/software/rtp4.html 8.1.2003

MontaVista-web page. MontaVista Software, News and Events –web page
http://www.mvista.com/news/2003/cge3.html 15.1.2003

SAF-press release. Premier Communications and Computing Companies Mobilize to Eliminate Service Interruptions.
http://www.saforum.org 28.4.2001

SAF-web (1). Service Availability Forum web page
http://www.saforum.org/about_the_forum.html 31.1.2003

SAF-web (2). Open Programming Interfaces, Service Availability Forum web page
http://www.saforum.org/about_the_forum.html 8.8.2002

CGLWG web page Carrier Grade Linux Working Group web page
http://www.osdl.org/projects/cgl/technical.html 31.1.2003

Case company internal material

Prod_descr. 2002. Product description document of the platform.

Purpose_doc. 2002. Purpose of the platform. Slide set in an internal training 20.9.2002.

Func_and_arch. 2002. Product Functionality and Architecture. Slide set in an internal training 20.9.2002.

Arch_princ. 2002. Architectural Principles –document of the platform.

Means_doc. 2002. Product development means. Slide set in an internal training 20.9.2002.

Sell_comp. 2002. Product platform Sellable Components description –document.

Horiz_req. 2002. Horizontal business unit's requirements for the platform –document

Frameworks_and_tools. 2002. Platform frameworks and tools. Company internal slide set.

Online_services_doc. Case company online customer services system. Slide set material of company internal training 28.1.2003

Interviews and discussions

B.K., Software Engineer Discussion about IBM WebSphere Application Server	1.11.2002
E.K., Chief R&D Engineer Discussion about development environment	9.12.2002
T.S., Technology Manager Internal customer interview	13.1.2003
O.M., Specialist Internal customer interview	13.1.2003
M.U. Chief Engineer Internal customer interview	14.1.2003
M.N. Program Manager Internal customer interview	15.1.2003
T.M. Information Design Technology Discussion about documentation	17.1.2003
M.R. Business Development Director in Fujitsu-Siemens Computers E-mail interview and discussion	21.1.2003
T.L Product Manager and J.H Principal Engineer E-mail discussion	23-27.1.2003
H.H. Senior R&D Manager Interview	31.1.2003

APPENDIX A

In the table below are presented the requirements for the platform that were defined in parallel with this thesis. The requirements were presented by the horizontal business unit to the platform development organisation product management.

Requirement	Description
There shall be a development documentation kit.	Materials like developer's guides, development environment installation guide, API documentation and architecture principles with examples delivered on CD-ROM and through extranet in PDF-format PDF.
Development documentation kit shall not include any references to components or other documents that are not available for the horizontal customer.	References to case company internal databases and documents.
There shall be a runtime documentation kit.	Documents, which can be used as a part of customers end product documentation.
All references to case company must be removed from runtime documentation kit.	Customer probably does not want to include case company references to its documentation.
There shall be means to support E1/T1 and ATM interfaces and protocols.	Customer wants to know how the platform can be connected with PCM and ATM networks.
Case company's own blade hardware shall be available for the platform.	Own hardware is essential for the first releases.
Case company logos shall not be visible in runtime environment.	Horizontal customer probably does not want to sell products with case company's logos on them. They want to use their own logos both in hardware and in software. License disclaimers are exception to this.
Hardware shall be supported from five to ten years.	Horizontal customers are telecommunications manufactures, and typical product lifetime in that segment is five to ten years.
There shall be defined sets of software sales items for runtime environment.	Customer shall be able test and deliver applications build on top of the platform. Delivery Media is CD-ROM and extranet.
There shall be defined sets of software sales items for development environment.	Customer shall be able test and deliver applications build on top of the platform. Delivery Media is CD-ROM and extranet.
Supported environments for each software sales items shall be documented.	Basic customer requirement.
Software sales items shall be verified in all supported environments.	Customer wants to have evidence that software item is really working as promised and where promised.
Software for runtime environment is delivered only as binary files.	Source codes are not delivered to horizontal customers.
Software for development environment is delivered only as binary files.	Customer does not need to have source codes to be able to develop applications.
All software deliveries shall include a list of included components with version numbers.	Delivered software shall be identified to enable correct maintenance.
Runtime software sales items shall be protected with proper license key or other method.	Licence keys prohibit misuse of software. They can also be used as a reference related to maintenance contracts etc.
Third party software versions needed in the platform shall be commercially available.	Customer shall be able to purchase needed third party software components if the case company cannot deliver them. This covers both runtime and development environment.
The case company shall be able to act as value added reseller for all third party software needed in runtime environment.	The horizontal business unit shall be able to sell complete runtime sales packages to horizontal customers.

The platform's first commercial release should include as complete high availability services as possible.	SAF compliant interfaces (both Application Interface and Platform Interface), have been built in such a way that high availability services component has clear boundaries and it is not interrelated with other software. This is also a requirement for selling the high availability services system component as a separate component.
The platform development organisation shall have dedicated account manager for horizontal business unit.	The platform development is responsible to support horizontal business unit's business according to agreed responsibility sharing. Account manager is the key contact towards it. Horizontal business unit is responsible for helpdesk and project management services for horizontal customers.
The platform development organisation shall provide a demo laboratory with resources and equipments.	Customer demos are crucial for horizontal business. Demos are needed e.g. for functionality, performance, scalability and development productivity.
The platform development organisation shall provide consultancy services for horizontal business unit.	The platform development organisation is responsible to support horizontal business unit business according to agreed responsibility sharing. Consultants are needed for supporting horizontal business unit to solve demanding customer problems.
The platform development organisation shall provide software maintenance for horizontal customers.	Software maintenance covers software bug fixes within agreed correction timeframe for all software sales items.
There shall be proper product presentation material available.	Presentation material is needed in marketing and sales activities. It includes e.g. product description, sales items, architecture principles and subsystem descriptions Presentation slide sets related to these materials shall be updated on release basis.
There shall be roadmap available for each key sales item.	Roadmaps are needed in marketing and sales activities. It is needed also for performance and scalability for at least three years of visibility.
The platform shall have a well-defined performance measurement set and regularly measured results.	Customers want to compare the performance of the platform to its competitors. This set shall also show how performance improves in relation to scalability. Measurement set shall include metrics showing typical control plane application performance (e.g. messaging capacity).
The platform shall have a roadmap for performance evolution.	Customer need to know how the platform's performance will evolve compared to its competitors.
There shall be planned demos of the platform.	Customers want to know how the platform can be used and how it works. Old demos shall be updated for each release.
Non-perpetual licensing shall be supported.	A perpetual licence gives you the right to use the software forever, while a non-perpetual licence lasts for a specific period. Third party components included in the platform shall also support non-perpetual licensing.
Perpetual licensing shall be supported for runtime environment.	Perpetual licensing shall be available for the runtime environment and for the third party components included in it.
Basic platform pricing shall be competitive.	Prices of hardware, operating system, high availability services and basic functionality shall be based on market prices.
Case company's extensions to the platform shall support value based pricing.	Extensions shall be charged according their real value.