Title

A Conceptual Framework for Development of Complex Systems

Summary

This PM describes a design support framework for developing high-quality teleco systems on time. Best design practice, black-box product structuring, delivery driven project planning, tailor made processes, organization of teams and information handling systems are combined into a unified whole.

The main benefit of the framework is that it helps to coordinate the efforts of many people working together. If it is efficiently communicated and endorsed by product- and system management, the framework can show how everyones particular work piece contribute to the whole product.

The framework is simple and based on established working practices. The ideas behind it can be applied directly in ongoing projects with a minimum of efforts.

Contents

1 Introduction

It is quite obvious that customer demands and new technologies will change the way we develop systems. One example is that changing customer requirements force us to carry out development incrementally. Another example is that the design processes must be adaptable to local needs at the design centres.

If this change is going to take place in an ordered way, we need some strategy to guide us through this transition. A concept, or framework serves this purpose. To be useful, this framework should have at least the following properties:

- It must be based on current working practices.
- It must be open to new ways of working.
- It must be simple enough to be communicated and understood.

In this PM we suggest a framework which we claim has these properties. It is based on the following principles:

- Guidelines for "Best design practice" shall be innate in the framework.
- The basic principles of abstraction and containment for developing complex systems should be part of the framework. This means that the "black-box" design paradigm of the AXE concept shall be supported. This is achieved by structuring the product information throughout as alternating layers of specifications and implementations together with their dependencies.
- The project planning should be based on integration and planned from behind.
- The framework shall not prescribe which methods, tools and models shall be used in a particular development project.
- The framework shall be is equally valid for SW, HW, mechanical and any other type of implementation technology.
- The design processes (methods, activities, tools, resources etc.) shall be structured to make it possible to shift the balance between local adaptation needs and global central control.
- Handling of design- and product information shall be done in the information system which is most appropriate for the task. An information system architecture shall ensure that the information systems can cooperate.
- It should be possible to organize teams along competence and projects.
- Design practice, product structure, project execution, processes, team organization and information handling shall make up a coherent whole.

2 Best Design Practice

Experience has shown that successful design of complex systems have some things in common:

- The system must be divided into parts that are uncoupled. Functions should as far as possible be implemented orthogonally, that is they should not interfere with each other. Another way of stating this is that the execution of one function should not affect any other function. These findings are part of a design practice known as "Axiomatic Design" [1].
- The project must equally well be as uncoupled as possible. The implementation of functions must be assigned to subprojects, or teams, in such a way that the subprojects do not interfere with each other. This is very important. There are even indications that some types of couplings may cause non-convergent loops between subprojects. In such a case the project should not even be started [2]. If couplings cannot be avoided, we must at least know where they are.
- The system design should make use of the black-box paradigm[1], thus supporting the principles of abstraction and containment. This means that we alternate between modelling a product from the outside and inside. In the external view, the part interacts with other parts in some local context. In doing so, we only model the externally visible properties of the part, disregarding its internals:

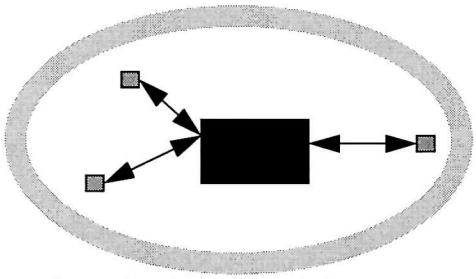

figure 1 A black box in its context

In the inside view, we model the internals of the product in such a way that its externals remain unchanged. Other interacting parts will then be used. These in turn are modelled from the outside:

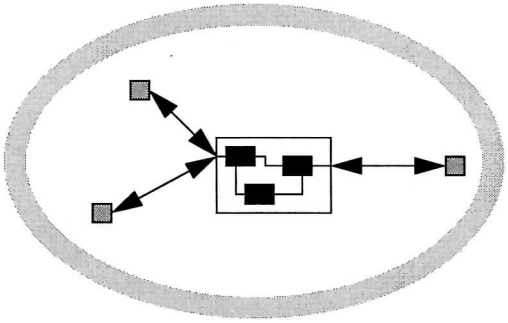

figure 2 An implementations of a black box

In this way we can split a complex syste into contexts that can be treated more or less independent of each other.

- The dynamics of the market- and technology situation makes it vital to consider the interactions of a certain context and what implications they have on the design paradigm [3].

1. This is the design paradigm of AXE, although it can at times be hard to spot in the documentation.

3 Product Parts Depend on Each Other

During the development of a complex system, we need an information model which is expressive enough for handling purposes, while still being general enough to comprise different modelling techniques and technologies. The "composed-of" product structure used in PRIM / GASK is not sufficient for this purpose. We need an information model with at least the following properties:

- It shall show the interdependencies between the functions and parts of the product.
- It shall be general enough to be applicable to all types of information(SW, HW, docware and mechanics, etc.)
- It shall be independent of the modelling structure of particular application domains.
- It shall be compatible with the product structure of PRIM

One model with these properties is SBDM, Specification Based Data Model [4]:

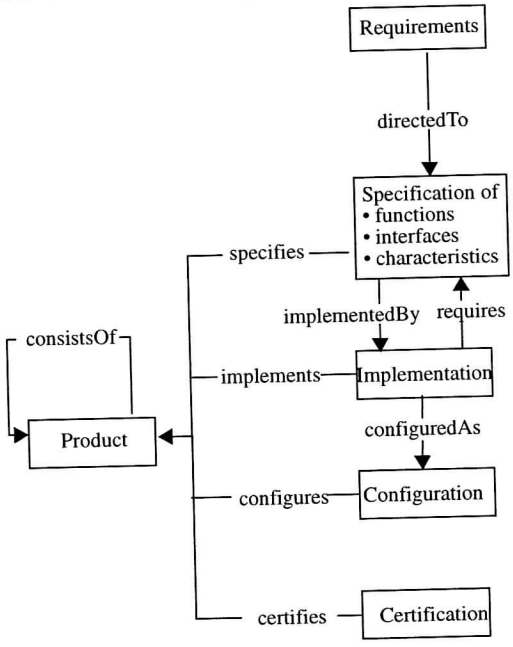

figure 3 The Specification Based Data Model

It has the following characteristics:

- It uses "Specifications" and "Implementations" iteratively, thus reflecting the black-box design paradigm (see chapter 2). Specifications specify functions, interfaces and other relevant characteristics of product parts on arbitrary levels. Functions are related to users of that functionality by the relation "requires"[1].
- Dependencies between product parts can be traced from requirements via the relations "implementedBy" and "requires" to any specification and implementation. Thus, if we make an update of, say a particular specification due to new requirements or a modification request, we always know which implementations and specifications are affected.
- The model does not show how a particular specification is modelled, or how the corresponding implementation is done. This must be documented elsewhere, for example by the mechanisms provided by the design tools. This is intentional, since we want the model to be just precise enough to handle complex design without being burdened with details. It should not be regarded as a disadvantage.
- Both specifications and implementations have versions and status, which makes it possible to assemble configurations and baselines.
- The model does not show the layering of the product, just the dependencies between product parts.

The basic constructs in this model is shown in the entity - relationship model in figure 3. It shows the ordinary "consistOf" structure of the product, together with the dependencies between product specifications and their implementations at arbitrary product levels.

1. The most appropriate way of modelling a function is as really as a relation between two parts (see chapter 11).

There are several advantages associated with SBDM. Here are some examples:

- It is possible to define independent design contexts, which is the key for successful development of complex systems (see chapter 6). The implementation work is contained within specifications to be implemented and specifications required for the implementation.
- Validation ("do we do the right thing?") of a specification is done in the context where the specification is to be used. In doing so, the part which is specified is regarded as a black box:

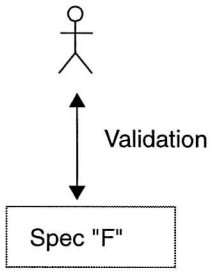

figure 4 Validation of a Specification

- Verification ("do we do the thing right?") of an implementation is done by comparing the specification with the implementation of that specification[1]:

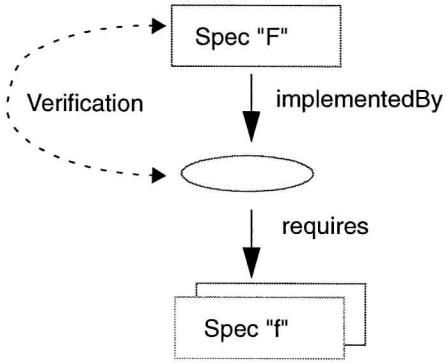

figure 5 Verification of an Implementation

1. Ellipsoids are used to symbolize implementations.

- Specification may have alternative implementations. This is advantageous if we want to quickly develop a prototype which will later be replaced. Another occasion is if we want to replace a hardware implementation in a certain technology with a more efficient one.

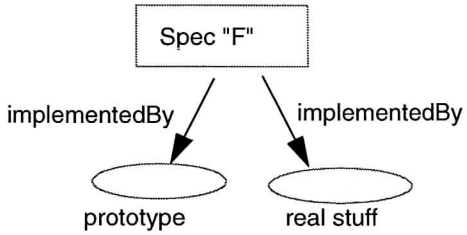

figure 6 Alternative implementations

- Sourced parts are modelled as specifications:

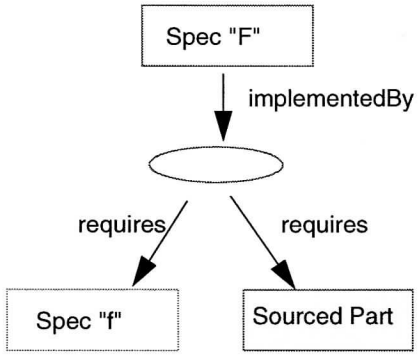

figure 7 Sourced parts

More examples can be given. Today, SBDM can be used as a conceptual framework to structure the vast amounts of information being produced in an AXE-project. It is directly applicable in the work with incremental development and improved AM-methods. Due to its simplicity, it is fairly straightforward to implement the model in a CM tool.

4 Project Planning from Behind

The order of deliveries within a project shall be based on end user functionality. What the customer wants first shall be delivered first, on time. To achieve this, we must know the dependencies between user functions and other functions needed to implement the user functions. In other words, we must know what has been called the "functional anatomy" of the product [5]:

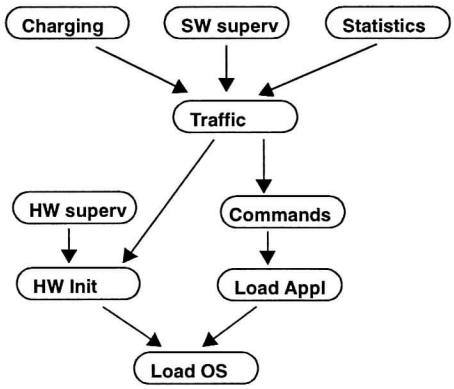

figure 8 Functional Anatomy

The anatomy can be represented in a dependency matrix:

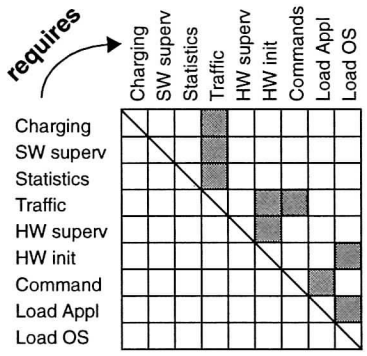

figure 9 Dependencies

Since functions are part of specifications, this matrix shows also the dependencies between specifications of products providing a function, and products required to provide that function. For example, a block implementing HW init requires a block providing Load OS. This means that SBDM (see chapter 3) is directly applicable. It should also be noted that the same functional anatomy can be implemented in different product structures, for example according to the structures in AXE 105 or AXE 106.

When the dependencies are laid down, it is a straightforward matter to lay out the project plan according to the priorities among functions, their delivery times and the amount of work to implement each function. Different strategies can be used. In incremental development, complete and testable user functionality are delivered in steps. In integration driven development, the system functionality is gradually built and tested from the bottom up. In both strategies, the step wise integration and testing of functions must of course be considered.

Since this procedure is based on dependencies between specifications, we propose to call it Specification Based Planning,SBP.

Today, the SBP procedure can be applied right on, and the project plan can be implemented in planning tool such as Autoplan. It is likely that something like SBP is necessary to handle incremental development, since the interactions between and testing of increments become even more important than in traditional design.

In the future, tools and method to support SBP should be developed. A prominent part of this will be the CM-tool.

Unless we develop the same product the same way over and over again, it is pointless to prescribe in advance how the coupling between the development project and PROPS shall be expressed. This must instead be done sometimes before TG2, presumably during the feasibility phase. This also means that the design process cannot state in advance which documents shall be delivered at which milestone. Instead, the design process must provide "connection points" where the progress according to the project plan can be coupled to the prescribed PROPS milestones and TG passages. These connection points are called Status Points in this framework (see chapter 5).

5 Tailored Processes

Most design processes in use today are monolithic in nature, which means that they become inflexible. At the same time there is a great need to adapt the design process to local needs. This contradiction can be solved by building adaptability into the design process from the outset. However, central control must persist of general features which are common to all development sites (for example the ABC product classification system). Thus the design process must exhibit both flexibility and stability. T achieve this we propose a **process architecture** for design processes, which is similar to the AM-architecture. It is comprised of three architectural entities [6]:

1 A **process core,** which provides the stability mechanism. It holds rules, principles, tools and other resources which are mandatory within a specified organization (which can be, but must not be, the entire Ericsson). The responsibility for this entity is central.

2 Reusable and adaptable **process components**, which provides reusable process products. They comprise all the process descriptions, work instructions, tools etc. which are needed to perform a limited and recurrent design task. The quality assurance is done at Status Points in the module, and these are also connection points to PROPS. The responsibility for these entities may be either central or local. Some examples are given below:

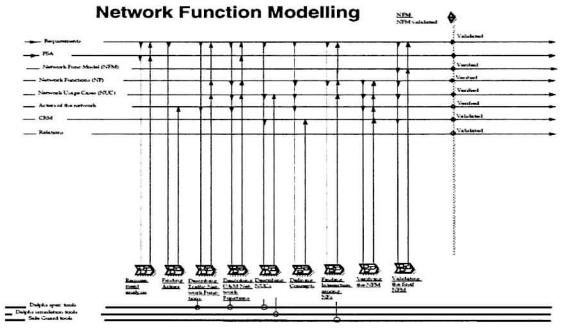

figure 10 A Process Component for Network Function Modelling

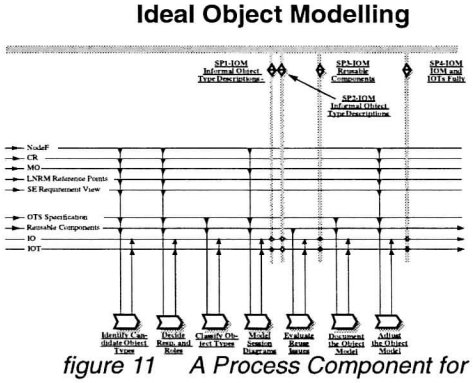

figure 11 A Process Component for Software Design

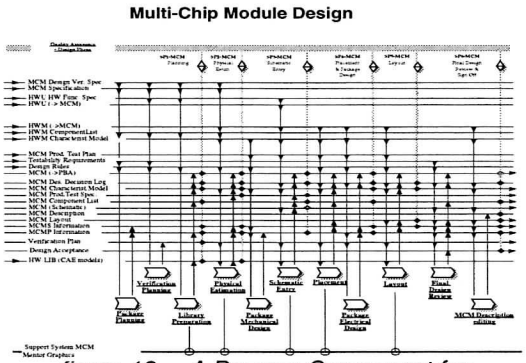

figure 12 A Process Component for MCM design

3 Tailor made **applied processes** adapted for a specific type of product development. The adaptability is achieved by picking a number of relevant process components and specialising them to local needs. The responsibility for this is local. Experiences found during a particular project can immediately be fed back to the process components without the need to involve central units.

Today, Multi Chip Modules have been designed using this model with good results. It is also used at ERA for system and SW design tasks.

In the future, the model should be formalised and implemented in some database, from which all necessary documentation and support can be generated.

6 Project- and Competence Teams

There are at least two aspects that should be considered when defining teams. One is the definition of project teams, and the other is how teams may be organized around competence areas.

Within this framework, the SBDM model (see chapter 3) makes it possible to define design competence teams for context bounded by specifications to be implemented and specifications needed for the implementation. The competence needed is determined by the nature of the context. For example, different competencies are needed to program blocks in PLEX than designing an Application Specific Circuit (ASIC).

These competencies are preferably organized in the line structure of the company, since they can be regarded as a pool of competencies which can be used in different projects. They should be responsible for all that is needed to perform a certain task. This comprises methods, tools, documentation, training, support and so on. In short, they should be responsible for and product owner of process components needed to perform the task.

When a project is organized, project teams may be set up to take responsibility of an entire user functionality. This function may be comprised of several design context, where different competencies are needed. For example, if the function needs new hardware to be designed and manufactured, competence from production units are needed in the team.

This is illustrated in figure 13 below, where two competencies "A" and "B" form a project team They are contributing with their competencies to implement a specification for a function "F", which in turn needs another function "f" to be implemented. This function in turn requires another function "g", which is already implemented.

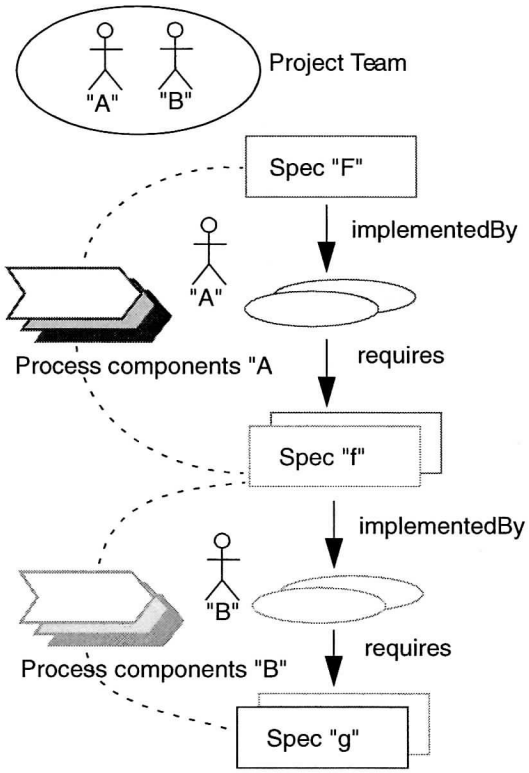

figure 13 Competence assignment

This concept has been used in the design of Multi Chip Modules with good results.

7 Handling the Information

A fundamental part of the framework is the information systems and what functions they provide. There is a natural tendency for the product owners of information systems to expand their domains into areas which the systems were not really designed for. This leads to many fruitless discussions of "which system is the best".

To avoid this, and to assign a role to each system, we need an architecture which shows the information systems and how they interact. The services and functions may be specified without any particular implementation / information system in mind. By doing so, we may change information systems while still maintaining the services.

The services can be grouped into three areas [7]:

- Design environment (DE); functions to support the daily design work such as file handling, build support etc.
- Management environment (ME); functions to control all the design information of a complex system during a project. This may be defining baselines and other configurations, exchanging design information between teams, supporting traceability, handling information status etc.
- Common Environment (CE); functions to support archiving and global exchange of information.

These environments appear in all areas which are involved in the life cycle of a product.

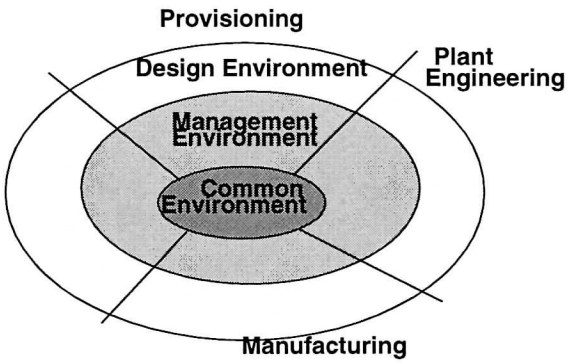

figure 14 Grouping of information handling functions

Examples of information systems in the Design Environment today are ClearCase for SW design and Mentor CAD systems for HW design. These systems will also host CAD libraries for specific implementation technologies. The component data base COMET does also belong to this domain.

Information systems in the Management Environment have been scarce. Most systems in the Management Environment are project unique. The Ericsson PDL-system is one possible implementation. PDL is based on a PDM (Product Data Management) system called Metaphase. Another possibility is to use ClearCase in this environment as well.

The APEX system from Rational is built on a concept which is simliar to SBDM. Therefore, it might very well be the best implementation of the DE and ME environments.

In the Common Environment we have today PRIM and GASK

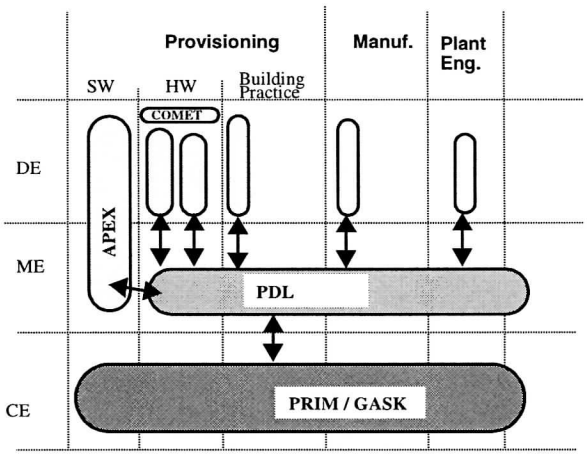

figure 15 An example of information systems implementing information handling functions

8 Fitting the Parts Together

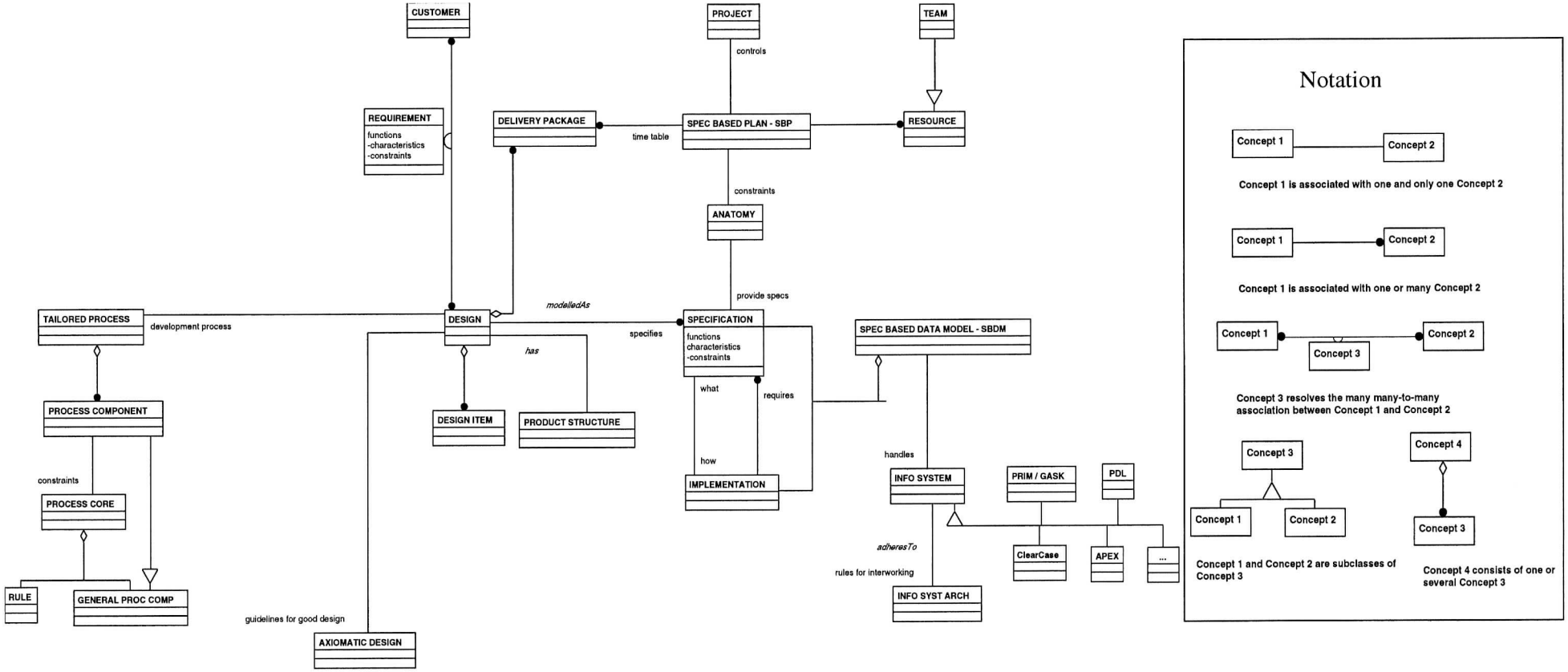

figure 16 The parts and their role in the framework

9 A scenario

Suppose we are going to develop a fictitious application using the AXE 106 AM architecture. Here is an example of how that would fit in the framework.

In the prestudy, the customer requirements are analysed and a specification of the application is developed. This is validated with the customer, and thus the first context is formed in which the AXE network is modelled as black box (see chapter 2).

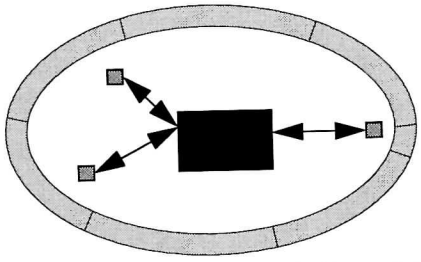

figure 17 Specification of AXE Network Functions

In doing so we may use an adapted process module from FSAD10 (see chapter 5). This module consists of use-case modelling supported by the Objectory tools. The specification is stored in the information system implementing the ME environment (see chapter 7) using the SBDM model (see chapter 3).

In the feasibility phase, it is decided that the design base is an existing version of System Modules XSS, PU's and an AM, AM1. A new AM, AM2, is to be developed. It is also decided that a modernized version of a printed circuit board (PCB) is to be developed, without changing the functional specification of the PCB.

This give us a first sketch of the architecture to be used. The specifications of the SM's are fetched from the archive. The specification of AM2 is developed, and the implementation, that is the interaction between the SM's, is elaborated using another process module from FSAD10: distribution of functionality. The implementation is stored together with the specifications of the SM's and the PU's.

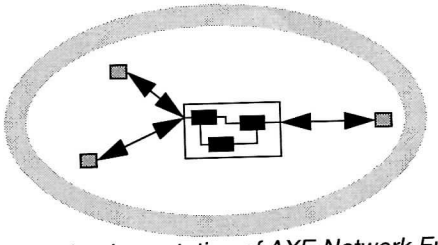

figure 18 Implementation of AXE Network Functions

Parallel to that, the implementation of the PCB is started, using an adapted process module for PCB's supported by the Mentor CAD system.

Now, we may plan the project using the SBP-method (see chapter 4). Functions are assigned to subsystems (ANT), blocks (CNT) and units (CAA). Increments and the order of deliverables are defined using the specifications and implementations needed. Teams are assigned according to contexts (see chapter 6). Configuration of specifications and implementations are used to maintain consistence between increments in different progress. This is communicated through the mechanisms in the information system. The specifications and products are stored in PRIM. The resources needed and the deliverables are collected in the project plan, whose milestone definitions are coupled to tollgates in PROPS.

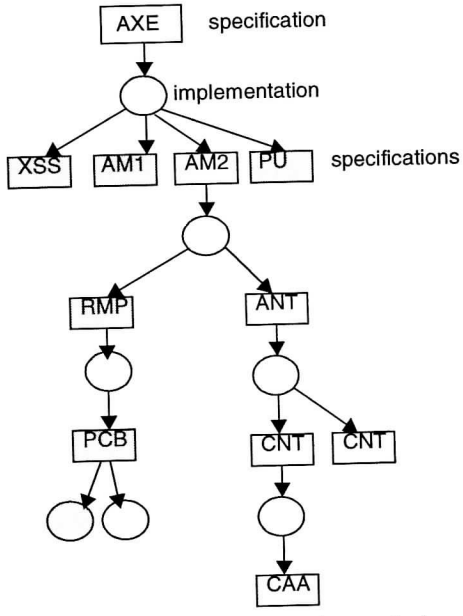

figure 19 The dependency chain

10 Start now

The framework can be applied right now. Some examples:

- The framework supports directly several of the principles for structuring the unit UAB/K, AXE 10 Methods, Tools & Training:
  - "Process Management on high level". Supported by the design process architecture (see chapter 5).
  - "Methods generically described to make it adaptable on local sites (BU/MLC)". This is also supported by the design process architecture.
  - "Structuring according to Product Areas with method, tools and training kept together". This is supported by the team organization (see chapter 6).
- The Rational product APEX CMVC (configuration managagement - version control) is (as far as can be judged from a short presentation) directly compatible with the Specification Based Data Model model, SBDM (see chapter 3). This means that there exist a tool support for the model. This must however be investigated.
- Incremental design will also benefit from SBDM. This model gives a simple interpretation of baselines and configuration in genera. Furthermore, since the development cycle will be transversed once for each increment, it makes sense to organise the design process according to the modular approach (see chapter 5).
- The work being done in FRED-FSAD10 needs to be aligned with the MEDAX products. This can be achieved by identifying specifications and implementations according to the SBDM. For example, an implementation of network services using FSAD10 will contain specifications of node functions, which are the same as Application Module Service Specifications in MEDAX.
- The AXE 106 product structure has a number of specification products associated to the ordinary product structure. These specification products correspond to specifications in SBDM (see chapter 3). Thus, document classes already exist for these in PRIM.
- The EDPI (Early Design Process Improvement) structure is essentially an attempt to introduce something with the same purpose as SBDM (my interpretation).
- The VVC concept can easily be included in the design process, since validation, verification and certification can be mapped on SDBM (see chapter 3).
- Project execution (see chapter 4) is directly applicable in ongoing projects.

The framework can be applied today with very simple means. It does not need any sophisticated tools. Here are some examples:

- The Specification Based Data Model can be applied by simply grouping the existing design information in "specifications" and "implementations" and illustrating their dependencies on a wall chart.
- Spread sheets can be used to specify design object status vs. mile-stones. These can in turn be coupled to project delivery times.
- Simple templates and rules can be used for quality assurance, for example the 1, 2 and 3 classification used by the "bilbesiktningen" (1 = remark, 2 = renewed inspection, 3 = not approved).

- The Specification Based Planning has more or less been used in CMS-30 [5]. The dependencies between functions can be graphically illustrated with "functional anatomy charts". Colours can be used to illustrate "readiness" (green for ready, yellow for on-time, red for delayed).

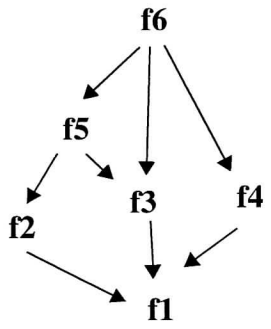

figure 20 Functional Anatomy

- Process components can be illustrated with for example FrameMaker, and placed on the wall in the project room to give a common view for the project participants:

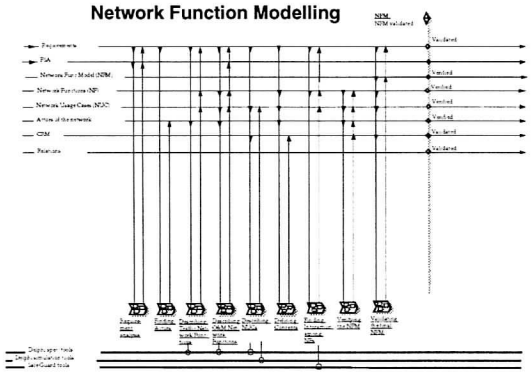

figure 21 A Process Component

11 Some Axioms

Any concept about how design is to be carried out, needs to be based on some assumptions. These might be called axioms, since they cannot be proven true or false. They will play the role of pathfinders for elaborating the concept, and can only be validated in practical design work.

Some axioms for this framework are:

Functions are associations between parts in a context. Referring to functions this way means that:

- there is always at least two parties involved in a function description.
- it is the characteristics of the providing part of the function that determines the set of possible functions.
- it is the context that determines which of the possible functions is valid.

In our daily experience, this is quite obvious. A book can be read or placed under a tilted table. A torch can light a room or be a symbol of the olympic games. When designing artefacts like AXE with particular functions in mind, this is not very obvious. However, regarding functions as relations has some very definite advantages:

- We have to consider all parts which the function relate. The tendency to "zoom in" on the design and forget about the part using the function will be less.
- We have to model all relevant characteristics of these parts in the context. No more "we forgot about the capacity..."
- We have to define the context. The chance that we will forget some important aspect will then be less.

Requirements are not specifications. Very often we mix up requirements with specifications. W may very well require that the outlet in the wall will provide 12 Volts, but the specification of the outlet is nevertheless 220 Volts. Requirements may be directed to parts having different specifications, and matched for compliances. When a part exist, its specification exists as long as the part exist, regardless of what requirements were put on it.

Flexibility is grounded in stability. In all flexible systems, there is some very stable structure. This goes for living organisms based on DNA as well as market economies based on regulations. In the AXE system, we have adopted this viewpoint[1]. The same thinking should be applied to processes and tools. The proposed process architecture in this PM is one attempt to do that.

The model of a system is determined by interactions with its context. Every model of a system we come up with is an abstraction of reality. It starts somewhere and ends somewhere. The interactions between the system and its relevant context determine which aspects are modelled and which are not, both in the system and the context. For example, requirements will influence the system, but the system will also influence the requirements, as when a customer changes his mind when he sees the first prototype of the system.

Methods and tools out of their natural contexts are useless. We may use a hammer on nails, but it is useless when it comes to writing. The same is valid for methods and tools. We should only use them in their natural context. For example, it is not appropriate to use one information handling system for all purposes.

Design support should focus on the needs of the designer This means that tools, methods, training, information handling support etc. should be delivered in packages, optimized to provide a solution to a certain, well defined design task. This is the idea behind the process component concept. Simply dumping a pile of tools and a process manual on the designers desk is not sufficient any more.

1. The ABC classification is one example of a very stable structure in AXE.

References

[1] N. P. Suh: The Principels of Design, Oxford University Press, 1990.

[2] Killander A.J. "Concurrent Engineering Requires UncoupledConcepts and Projects", Proceedings of the ISPE International Conference on Concurrent Engineering, CE95, August 22-25, 1995, McLean, Virginia, USA.

[3] Taxén L. F., "The Dialectical Approach to System Design", Proceedings of Integrated Design and Process Technology, Austin 1995.

[4] Gandhi, and Robertson, 1995,"SDBM as a Model for Codesign Data" in "Computer Aided Software/Hardware Engineering," IEEE Press, Piscataway.

[5] J. Järkvik, L. Kylberg et al"On succeeding", EN/LZT 123 1987

[6] H95 1639: System Design Process Basics

[7] F95 2073: Arkitektur och principer för informationssystem i AXE-N