# Module 20 Medical Imaging case, CAFCR illustration

logo
TBD

Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

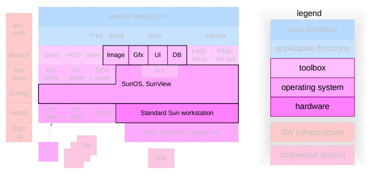
This module provides a complete illustration of the CAFCR based architecting method. The case is a Medical Imaging Workstation, created in the early nineties.

version: 0 status: finished September 1, 2020

# Contents

# Chapter 1

# Medical Imaging in Chronological Order



## 1.1 Project Context

Philips Medical Systems is a very old company, dating back to 1896 when the first X-ray tubes were manufactured. Many imaging modalities have been added to the portfolio later, such as Ultra Sound, Nuclear Medicaid, Computed Tomography and Magnetic Resonance Imaging. Since the late seventies the management was concerned by the growing effort to develop the viewing functionality of these systems. Many attempts have been made to create a shared implementation of the viewing functionality, with failures and partial successes.

In 1987 a new attempt was started by composing a team, that had the charter to create a *Common Viewing* platform to be used in all the modalities. This team had the vision that a well designed set of SW components running on standard workstation hardware would be the solution. In the beginning of 1991 many components had been built. For demonstration purposes a *Basic Application* was developed. The *Basic Application* makes all lower level functionality available via a rather technology-oriented graphical user interface. The case description starts at this moment, when the *Basic Application* is shown to stakeholders within Philips Medical Systems.

## 1.2 Introduction

The context of the first release of Medical Imaging is shown in Section 1.1. The chronological development of the first release of the medical imaging workstation is described in Section 1.3. Sections 1.4 and 1.5 zoom in on two specific problems encountered during this period.

## 1.3 Development of Easyvision RF

The new marketing manager of the *Common Viewing* group was impressed by the functionality and performance of the *Basic Application*. He thought that a stand alone product derived from the *Basic Application* would create a business opportunity. The derived product was called Easyvision, the first release of the product was called Easyvision R/F. This first release would serve the URF X-ray market. The *Common Viewing* management team decided to create Easyvision RF in the beginning of 1991.

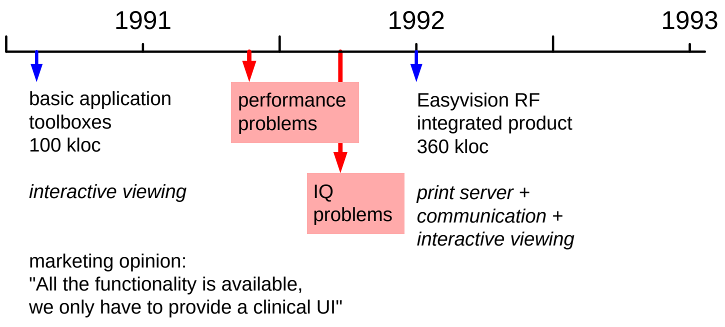


Figure 1.1: Chronological overview of the development of the first release of the Easyvision

The enthusiasm of the marketing people for the *Basic Application* was based on the wealth of functionality that was shown. It provided all necessary viewing functions and even more. Figure 1.1 shows the chronology, and the initial marketing opinion. Marketing also remarked: ”Normally we have to beg for more functionality, but now we have the luxury to throw out functionality”. The addition of viewing software to the conventional modality products[1] was difficult for many reasons, such as *legacy code and architecture*, and *safety and related testing requirements*. The Easyvision did not suffer from the legacy, and the self sustained product

[1]*Modality products* are products that use one imaging technique such as Ultra Sound, X-ray or Magnetic Resonance Imaging

provided a good means to separate the modality concerns from the image handling concerns.

This perception of a nearly finished product, which only needed some user interface tuning and some functionality reduction, proved to be a severe underestimation. The amount of code in the 1991 *Basic Application* was about 100 kloc (kloc = thousand lines of code, including comments and empty lines), while the product contained about 360 kloc.

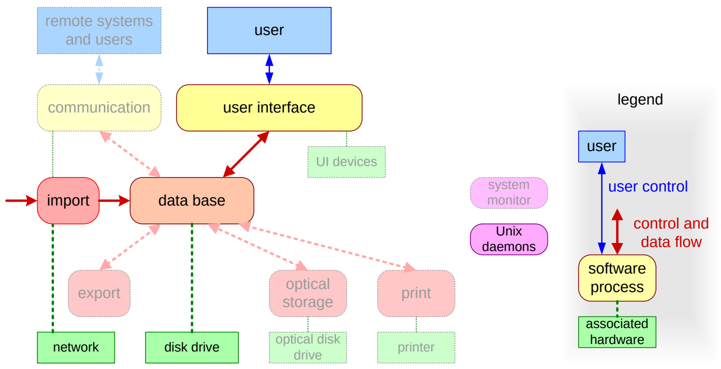


Figure 1.2: The functionality present in the Basic Application shown in the process decomposition. The light colored processes were added to create the Easyvision

The *Basic Application* provided a lot of viewing functionality, but the Easyvision as a product required much more functionality. The required additional functionality was needed to fit the product in the clinical context, such as:

- interfacing with modalities, including remote operation from the modality system
- storage on optical discs
- printing on film

Figure 1.2 shows in the process decomposition what was present and what was missing in the 1991 code. From this process decomposition it is clear that many more systems and devices had to be interfaced. Figures 1.2 and 1.3 are explained further in Chapter 3.

Figure 1.3 also shows what was present and what was missing in the Basic Application, but now in the construction decomposition. Here it becomes clear that also the application-oriented functionality was missing. The *Basic Application* offered generic viewing functionality, exposing all functionality in a rather technical way to the user. The clinical RF user expects a very specific viewing interaction, that is based on knowledge of the RF application domain.

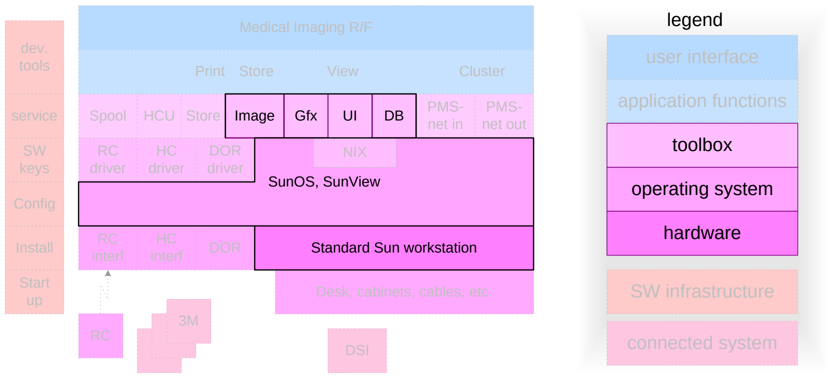


Figure 1.3: The functionality present in the Basic Application shown in the construction decomposition. The light colored components were added to create the Easyvision

The project phases from the conception of a new product to the introduction in the market is characterized by many architectural decisions. Architecting methods are valuable means in this period. Characteristic for an immature architecting process is that several crises occur in the integration. As shown in Figure 1.1 both a performance and a (image quality related) safety crisis happened in that period.

## 1.4 Performance Problem

The performance of the system at the end of 1991 was poor, below expectation. One of the causes was the extensive use of memory. Figure 1.4 shows the performance of the system as a function of the memory used. It is also indicates that a typically loaded system at that moment used about 200 MByte. Systems which use much more memory than the available physical memory decrease significantly in performance due to the paging and swapping to get data from the slow disk to the fast physical memory and vice versa.

The analysis of additional measurements resulted in a decomposition of the memory used. The decomposition and the measurements are later used to allocate memory budgets. Figure 1.5 shows how the problem of poor performance was tackled, which is explained in much more detail in Chapter 3. The largest gains were obtained by the use of shared libraries, and by implementing an anti-fragmentation strategy for bulk data. Smaller gains were obtained by tuning, and analyzing the specific memory use more critical.

Figure 1.6 shows the situation per process. Here the shared libraries are shown separate of the processes. The category *other* is the accumulation of a number of small processes. This figure shows that every individual process did fit in the

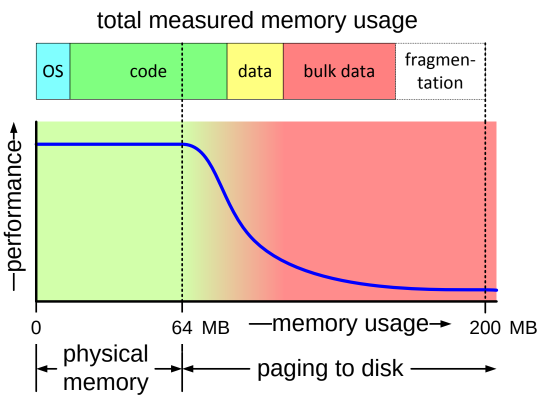


Figure 1.4: Memory usage half way R1

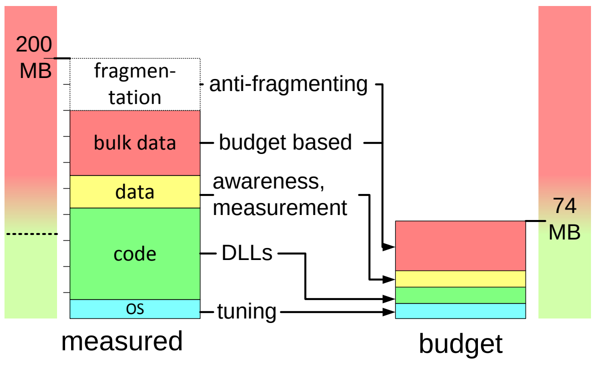


Figure 1.5: Solution of memory performance problem

available amount of memory. A typical developer tests one process at a time. The developers did not experience a decreased performance caused by paging, because the system is not paging if only one process is active. At the time of integration, however, the processes are running on the same hardware concurrently and then the performance is suddenly very poor.

Many other causes of performance problems have been found. All of these are shown in the annotated overlay on the software process structure in Figure 1.7.

Many of the performance problems are related to overhead, for instance for I/O and communication. A crucial set of design choices is related to granularity: a fine grain design causes a lot of overhead. Another related design choice is the mechanism to be used: high level mechanisms introduce invisible overheads. How aware should an application programmer be of the underlying design choices?

For example, accessing patient information might result in an implicit transaction and query on the database. Building a patient selection screen by repeatedly

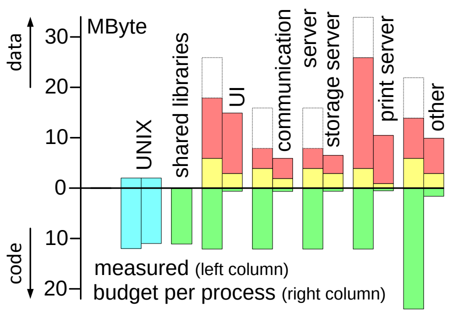


Figure 1.6: Visualization per process

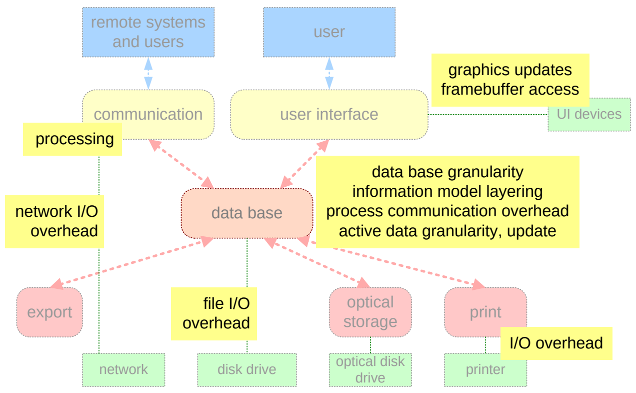


Figure 1.7: Causes of performance problems, other than memory use

calling such a function would cause tens to hundreds of transactions. With 25 ms per transaction this would result in seconds of overhead only to obtain the right information. The response becomes even worse if many layers of information have to be retrieved (patient, examination, study, series, image), resulting in even worse response time.

The rendering to the screen poses another set of challenges. The original *Basic Application* was built on Solaris 1, with the SunView windowing system. This system was very performance efficient. The product moved away from SunView, which was declared to be obsolete by the vendor, to the X-windowing system. The application and the windowing are running in separate processes. As a consequence all screen updates cause process communication overhead, including several copy operations of screen bitmaps. This problem was solved by implementing an integrated X-compatible screen manager running in the same process as the application, called Nix[2].

[2]A Dutch play on words: *niks* means nothing

Interactive graphics require a fast response. The original brute force method to regenerate always the entire graphics object was too slow. The graphics implementation had to be redesigned, using damage area techniques to obtain the required responsiveness.

## 1.5 Safety

The clinical image quality can only be assessed by clinical stakeholders. Clinical stakeholders start to use the system, when the performance, functionality and reliability of the system is at a reasonable level. This reasonable level is achieved after a lot of integration effort has been spent. the consequence is that image quality problems tend to be detected very late in the integration. Most image quality problems are not recognized by the technology-oriented designers. The technical image quality (resolution, brightness, contrast) is usually not the problem.

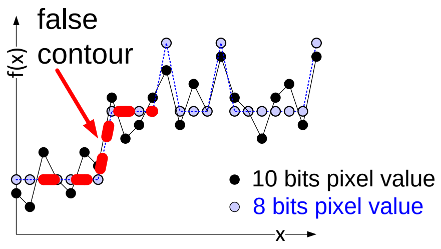


Figure 1.8: Image quality and safety problem: discretization of pixel values causes false contouring

Figure 1.8 shows a typical image quality problem that popped up during the integration phase. The pixel value $x$, corresponding to the amount of X-ray dose received in the detector, has to be transformed into a grey value $f(x)$ that is used to display the image on the screen. Due to discretization of the pixel values to 8 bits *false contours* become visible. For the human eye an artefact is visible between pixels that are mapped on a single grey value and neighboring pixels that are mapped on the next higher grey value. It is the levelling effect caused by the discretization that becomes visible as false contour. This artefact is invisible if the natural noise is still present. Concatenation of multiple processing steps can strongly increase this type of artifacts.

The original design of the viewing toolboxes provided scaling options for textual annotations, with the idea that the readability can be guaranteed for different viewport sizes. A viewport is a part of the screen, where an image and related information are shown. This implementation of the annotations on the X-ray system, however, conflicts in a dangerous way with this model of scalable annotations, see Figure 1.9.

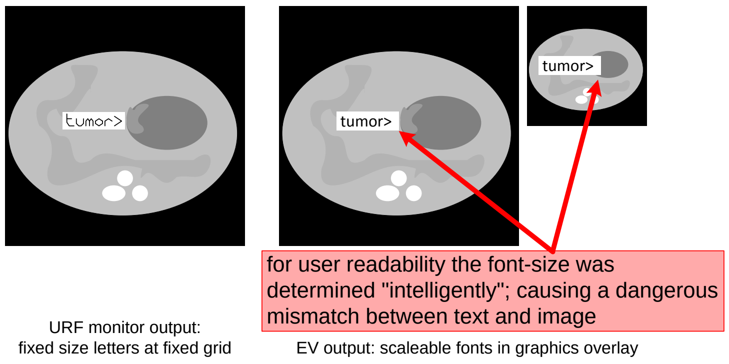

tumor>
tumor>
tumor>
for user readability the font-size was determined "intelligently"; causing a dangerous mismatch between text and image
URF monitor output: fixed size letters at fixed grid
EV output: scaleable fonts in graphics overlay

Figure 1.9: Safety problem caused by different text rendering mechanisms in the original system and in Easyvision

The annotations in the X-ray room are made on a fixed character grid. Sometimes the ’>’ and ’<’ characters are used as arrows, in the figure they point to the tumor. The text rendering in the medical imaging workstation is not based on a fixed character grid; often the texts will be rendered in variable-width characters. The combination of interface and variable-width characters is already quite difficult. The font scaling destroys the remaining part of the text-image relationship, with the immediate danger that the annotation is pointing to the wrong position.

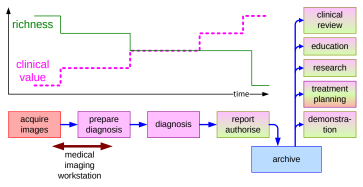
The solution that has been chosen is to define an encompassing rectangle at the interface level and to render the text in a best fit effort within this encompassing rectangle. This strategy maintains the image-text relationship.

## 1.6 Summary

The development of the Easyvision RF started in 1991, with the perception that most of the software was available. During the developement phase it became clear that a significant amount of functionality had to be added in the area of printing. Chapter 2 will show the importance of the printing fumctionality. Performance and safety problems popped up during the integration phase. Chapter 3 will show the design to cope with these problems.

# Chapter 2

# Medical Imaging Workstation: CAF Views



## 2.1 Introduction

This chapter discusses the *Customer Objectives*, *Application* and *Functional* views of the Medical Imaging Workstation. Section 2.2 describes the radiology context. Section 2.3 describes the typical application of the system. Section 2.4 shows the key driver graph, from customer key drivers to system requirements, of the Medical Imaging Workstation. Section 2.5 shows the development of functionality of the family of medical imaging workstations in time. Section 2.6 discusses the need for standardization of information to enable interoperability of systems within the department and the broader scope of the hospital. The conclusion is formulated in section 2.7.

## 2.2 Radiology Context

The medical imaging workstation is used in the radiology department as an add-on to URF X-ray systems. The main objective of the radiologist is to provide diagnostic information, based on imaging, to the referring physician. In case of gastrointestinal problems X-ray images are used, where the contrast is increased by digestion of barium meal.

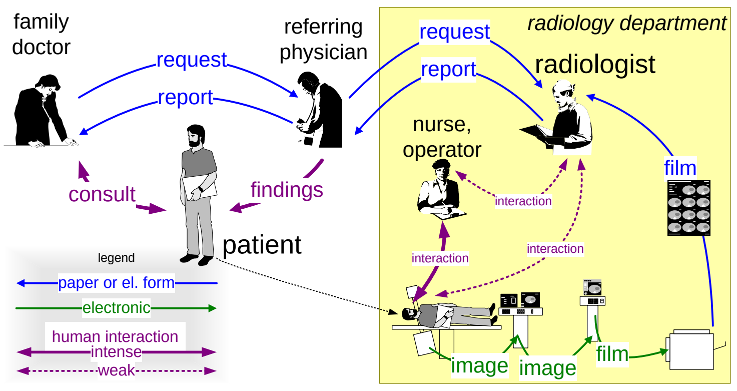


Figure 2.1: The clinical context of the radiology department, with its main stakeholders

The work of the radiologist fits in an overall clinical flow, see Figure 2.1. The starting point is the patient visiting the family doctor. The family doctor can refer to a consultant; for gastrointestinal problems the consultant is an internist. The family doctor writes a request to this consultant. In the end the family doctor receives a report from the consultant.

Next the patient makes an appointment with the consultant. The consultant will do his own examination of the patient. Some of the examinations are not done by the consultant. Imaging, for example, is done by radiologist. From the viewpoint of the radiologist the consultant is the referring physician. The referring physician uses a request form to indicate the examination that is needed.

The patient makes an appointment via the administration of the radiology department. The administration will schedule the examination. The examination is done by hospital personnel (nurses, operator) under supervision of the radiologist. Most contact is between nurse and patient; contact between radiologist and patient is minimal.

The outcome of the imaging session in the examination room is a set of films with all the images that have been made. The radiologist will view these films later that day. He will dictate his findings, which are captured in written format and sent to the referring physician. The referring physician performs the overall diagnosis and discusses the diagnosis and, if applicable, the treatment with the patient.

The radiology department fits in a complex financial context, see Figure 2.2. The patient is the main subject from a clinical point of view, but plays a rather limited role in the financial flow. The patient is paying for insurance, which decouples him from the rest of the financial context.

The insurance company and the government have a strong interest in cost

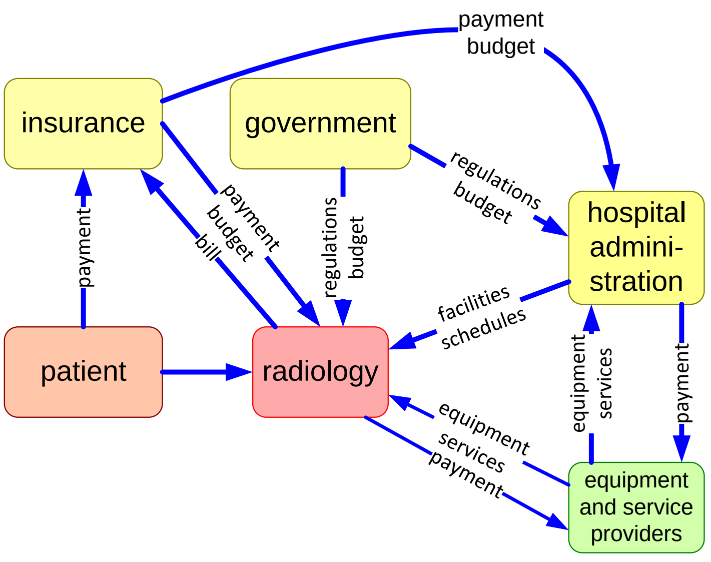


Figure 2.2: The financial context of the radiology department

control[1]. They try to implement this by means of regulations and budgets. Note that these regulations vary widely over the different countries. France, for instance, has stimulated digitalization of X-ray imaging by higher reimbursements for digital images. The United States regulation is much less concerned with cost control, here the insurance companies participate actively in the health care chain to control the cost.

The hospital provides facilities and services for the radiology department. The financial decomposition between radiology department and hospital is not always entirely clear. They are mutually dependent.

The financial context is modeled in Figure 2.2 in a way that looks like the Calculating with Concepts technique, described by Dijkman et al in [2]. The diagram as it is used here, however, is much less rigorous as the approach of Dijkman. In this type of development the main purpose of these diagrams is building insight in the broader context. The rigorous understanding, as proposed by Dijkman, requires more time and is not needed for the purpose here. Most elements in the diagram will not even have a formal interface with the product to be created. Note also that the diagram is a simplification of the reality: the exact roles and relations depend on the country, the culture and the type of department. For example a university hospital in France is different from a commercial imaging center in the USA. Whenever entities at this level are to be interfaced with the

[1]sometimes it even appears that that is the main interest, quality of health care appears than to be of secondary importance

medical imaging workstation then an analysis is needed of the greatest common denominator to be able to define a rigorous interface.

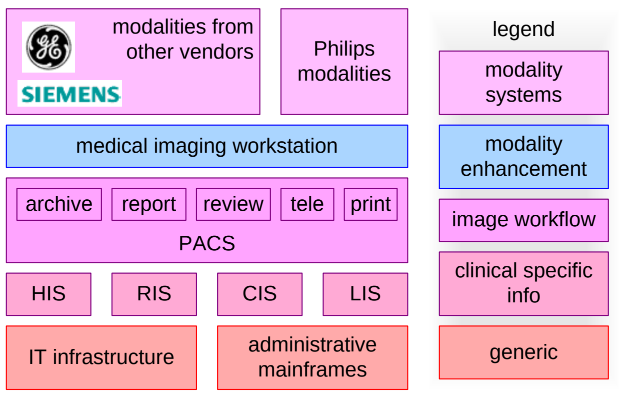


Figure 2.3: Application layering of IT systems

The medical imaging workstation is playing a role in the information flow in the hospital, it is part of the large collection of IT systems. Figure 2.3 shows a layered model of IT systems in the hospital, to position this product in the IT context. It is a layered model, where the lower layers provide the more generic functionality and the higher layers provide the more specific clinical imaging functionality.

In the hospital a normal generic IT infrastructure is present, consisting of networks, servers, PC's and mainframes. More specialized systems provide clinical information handling functions for different hospital departments (LIS for laboratory, CIS for cardio and RIS for radiology) and for the entire hospital (HIS Hospital Information System).

The generic imaging infrastructure is provided by the PACS (Picture Archiving and Communication System). This is a networked system, with more specialized nodes for specific functions, such as reporting, reviewing, demonstration, teaching and remote access.

The medical imaging workstation is positioned as a modality enhancer: an add-on to the modality product to enhance productivity and quality of the examination equipment. The output of the modality enhancer is an improved set of viewable images for the PACS.

Figure 2.4 shows a reworked copy of the reference model for image handling functions from the "PACS Assessment Final Report", September 1996 [1]. This reference model is classifying application areas on the basis of those characteristics that have a great impact on design decisions, such as the degree of distribution, the degree and the cause of variation and life-cycle.

*Imaging and treatment* functions are provided of modality systems with the focus on the patient. Safety plays an important role, in view of all kinds of hazards

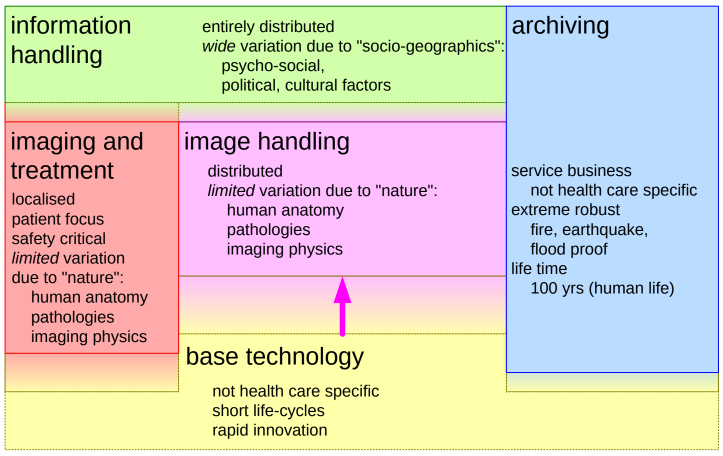


Figure 2.4: Reference model for health care automation

such as radiation, RF power, mechanical movements et cetera. The variation between systems is mostly determined by:

- the acquisition technology and its underlying physics principles.
- the anatomy to be imaged
- the pathology to be imaged

The complexity of these systems is mostly in the combination of many technologies at state-of-the-art level.

*Image handling* functions (where the medical imaging workstation belongs) are distributed over the hospital, with work-spots where needed. The safety related hazards are much more indirect (identification, left-right exchange). The variation is more or less the same as the modality systems: acquisition physics, anatomy and pathology.

The *information handling* systems are entirely distributed, information needs to be accessible from everywhere. A wide variation in functionality is caused by "social-geographic" factors:

- psycho-social factors
- political factors
- cultural factors
- language factors

These factors influence what information must be stored (liability), or must not be stored (privacy), how information is to be presented and exchanged, who may access that information, et cetera.

The *archiving* of images and information in a robust and reliable way is a highly specialized activity. The storage of information in such a way that it survives fires, floods, and earthquakes is not trivial[2]. Specialized service providers offer this kind of storage, where the service is location-independent thanks to the high-bandwidth networks.

All of these application functions build on top of readily available IT components: the *base technology*. These IT components are innovated rapidly, resulting in short component life-cycles. Economic pressure from other domains stimulate the rapid innovation of these technologies. The amount of domain-specific technology that has to be developed is decreasing, and is replaced by base technology.

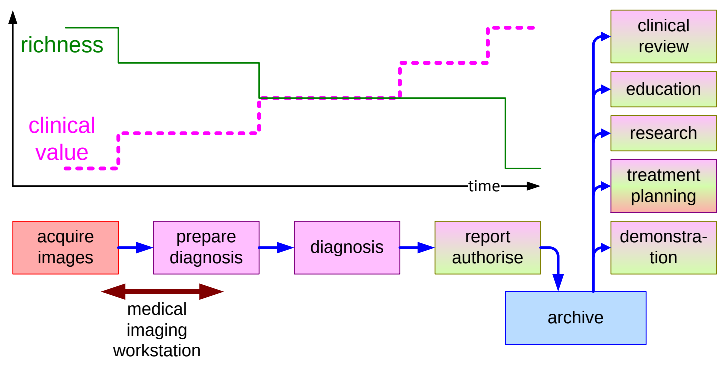


Figure 2.5: Clinical information flow

Figure 2.5 comes from the same report [1] showing the information flow within this reference model. During this flow the clinical value is increasing: annotations, comments, and anamnesis can be added during and right after the acquisition. The preparation for the diagnosis adds analysis results, optimizes layout and presentation settings, and pre-selects images. Finally the diagnosis is the required added value, to be delivered to the referring physician.

At the same time the richness of the image is decreasing. The richness of the image is how much can be done with the pixels in the image. The images after acquisition are very rich, all manipulation is still possible. When leaving the acquisition system the image is exported as a system independent image, where a certain trade-off between size, performance, image quality, and manipulation flexibility is made. This is an irreversible step in which some information is inherently lost. The results of the preparation for diagnosis are often frozen, so that no accidental

[2]Today terrorist attacks need to be included in this list full of disasters, and secure needs to be added to the required qualities.

changes can be made afterwards. Because this is the image used to diagnose, it is also archived to ensure liability. The archived result is similar to an electronic photo, only a limited set of manipulations can still be performed on it.

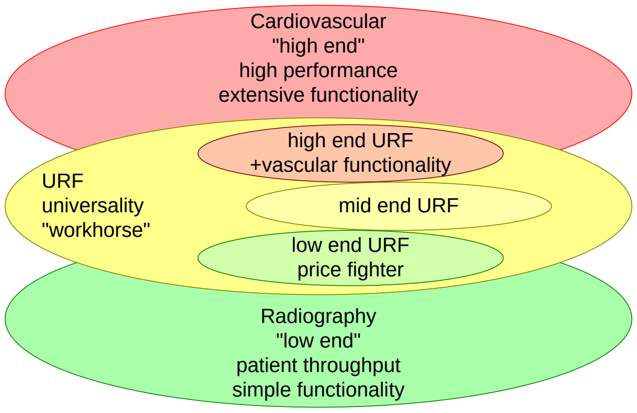


Figure 2.6: URF market segmentation

The first releases of the medical imaging workstation, as described in this case, are used in conjunction with URF (Universal Radiography Fluoroscopy) systems. This family of systems is a mid-end type of X-ray system, see Figure 2.6. At the high end cardiovascular systems are used, with high clinical added value and a corresponding price tag. At the low end “radiography” systems offer straight forward imaging functionality, oriented at patient throughput. Approximately 70% of all X-ray examinations are radiographic exposures.

The URF systems overlap with cardiovascular and radiography market segments: high end URF systems also offer vascular functionality. Low end URF systems must fit in radiography constraints. The key driver of URF systems is the universality, providing logistic flexibility in the hospital.

## 2.3 Typical Case

The specification and design of the medical imaging workstation was based on “typical” cases. Figure 2.7 shows the typical case for URF examinations. Three examination rooms are sharing one medical imaging workstation. Every examination room has an average throughput of 4 patients per hour (patient examinations are interleaved, as explained below for Figure 2.8).

The average image production per examination is 20 images, each of $1024^2$ pixels of 8 bits. The images are printed on large film sheets with a size of approximately $24 * 30 cm^2$. One film sheet consists of 4k by 5k pixels. The images must be sufficiently large to be easily viewed on the lightbox. These images are typically

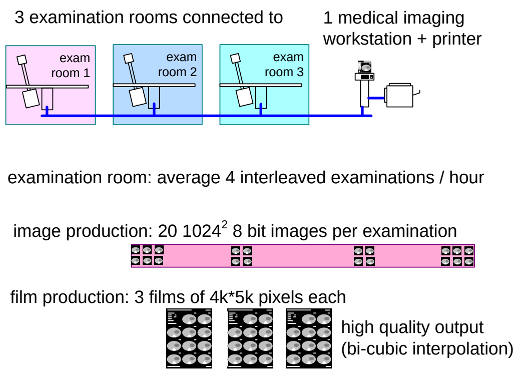

3 examination rooms connected to
1 medical imaging workstation + printer
exam room 1
exam room 2
exam room 3
examination room: average 4 interleaved examinations / hour
image production: 20 1024² 8 bit images per examination
film production: 3 films of 4k*5k pixels each
high quality output (bi-cubic interpolation)

Figure 2.7: Typical case URF examination

printed on 3 film sheets. Image quality of the film sheets is crucial, which translates into the use of bi-cubic interpolation.

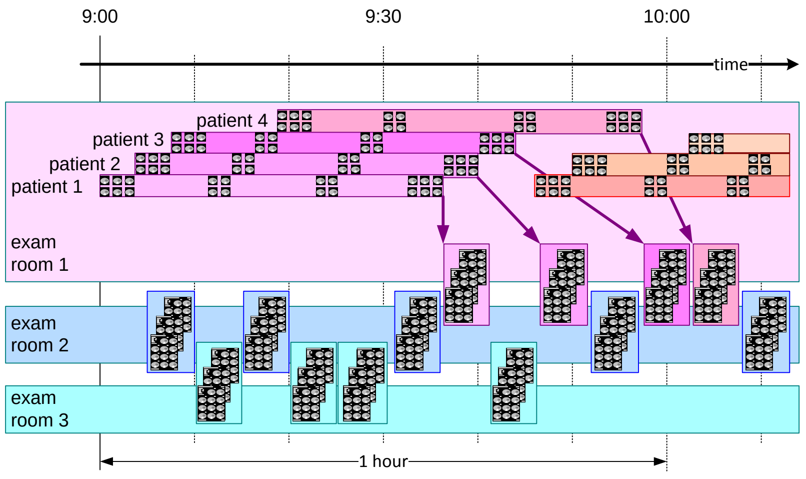


Figure 2.8: Timing of typical URF examination rooms

Figure 2.8 shows how patient examinations are interleaved. The patient is examined over a period of about one hour. This time is needed because the barium meal progresses through the intestines during this period. A few exposures are made during the passage of clinical relevant positions. The interleaving of patients in a single examination room optimizes the use of expensive resources. At the level of the medical imaging workstation the examinations of the different examination rooms are imported concurrently. The workstation must be capable of serving all three acquisition rooms with the specified typical load. The latency between the end of the examination and the availability of processed film sheets is not very critical.

## 2.4 Key Driver Graph

Figure 2.9 shows the key drivers from the radiologist point of view, with the derived application drivers and the related requirements, as described in Section **??**. The graph is only visualized for the key drivers and the derived application drivers. The graph from application drivers to requirements is a many-to-many relationship, that becomes too complex to show in a single graph.

The key drivers are discussed in Subsections 2.4.1 to 2.4.5.

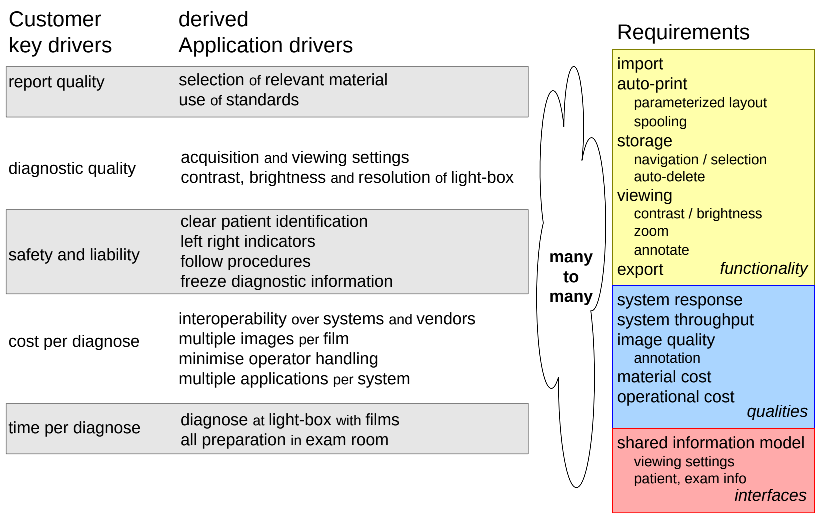


Figure 2.9: Key drivers, application drivers and requirements

### 2.4.1 Report Quality

The report quality determines the satisfaction of the referring physician, who is the customer of the radiologist. The layout, accessibility, and all these kind of factors determine the overall report quality. The radiologist achieves the report quality by:

**selection of relevant material** The selection of the material to be reported to the referring physician determines to a large degree the report quality.

**use of standards** The use of standard conventions, for instance pathology classification, improves the report quality.

### 2.4.2 Diagnostic Quality

The diagnostic quality is the core of the radiologist's work. The diagnostic quality is achieved by:

**acquisition and viewing settings** The actual acquisition settings and the related viewing settings have a great impact on the visibility of the pathology and anatomy.

**contrast, brightness and resolution of lightbox** The lightbox has a very good diagnostic image quality: high brightness, high resolution, and many images can be shown simultaneously.

### 2.4.3 Safety and Liability

Erroneous diagnoses are dangerous for the patient; the radiologist might be sued for mistakes. Also mistakes in the related annotations (wrong patient name, wrong position) are a safety risk for the patient and hence a liability risk for the radiologist. The derived application drivers for safety and liability are:

**clear patient identification** Erroneous patient identification is a safety risk.

**left right indicators** Erroneous positioning information is a safety risk. Left-right exchanges are notoriously dangerous.

**follow procedures** Clinical procedures reduce the chance of human errors. Following these procedures lowers the liability for the radiologist.

**freeze diagnostic information** Changing image information after the diagnosis is a liability risk: different interpretations are possible, based on the changes.

### 2.4.4 Cost per Diagnosis

Insurance and government generate a lot of cost pressure. Cost efficiency can be expressed in cost per diagnosis. The cost per diagnosis is reduced in the following ways:

**interoperability over systems and vendors** Mix and match of systems, not constrained by vendor or system lock-ins, allow the radiology department to optimize the mix of acquisition systems to the local needs.

**multiple images per film** Film is a costly resource (based on silver). Efficiency of film real estate is immediately cost efficient. A positive side effect is that film efficiency is also beneficial for viewing on the lightbox, because the images are then put closer together.

**minimize operator handling** Automation of repeated actions will reduce the amount of personnel needed, which again is a cost reduction. An example is the use of predefined and propagated settings that streamline the flow of information. This is a cost reduction, but most of all it improves the convenience for the users.

**multiple applications per system** Universality of acquisition system and workstation provides logistics flexibility in the radiology department. This will in the end result in lower cost.

### 2.4.5 Time per Diagnosis

Time efficiency is partially a cost factor, see 2.4.4, but it is also a personal satisfaction issue for the radiologist. The time per diagnosis is reduced by the following means:

**diagnose at lightbox with films** This allows a very fast interaction: zooming is done by a single head movement, and the next patient is reached by one button, that exchanges the films mechanically in a single move.

**all preparation in exam room** The personnel operating the examination room also does the preparation for the diagnosis. This work is done on the fly, interleaved with the examination work.

### 2.4.6 Functional Requirements

The functionality that is needed for to realize the derived application drivers is:

**import** The capability to import data into the workstation data store in a meaningful way.

**autoprint** The capability to print the image set without operator intervention:

- **parametrized layout** Film layout under control of the remote acquisition system.
- **spooling** Support for concurrent import streams, which have to be printed by a single printer.

**storage** The capability to store about one day of examinations at the workstation, both as a buffer and to enable later review:

- **navigation/selection** The capability to find and select the patient, examination and images.

**autodelete** The capability to delete images when they are printed and no longer needed. This function allows the workstation to be used in an operator free server. The import, print and auto-delete run continuously as a standard sequence.

**viewing** All functions to show and manipulate images, the most frequently used subset:

**contrast/brightness** Very commonly used grey-level user interface.

**zoom** Enlarge part of the image.

**annotate** Add textual or graphic annotations to the image.

**export** Transfer of images to other systems.

Note that the *import*, *storage* and *autoprint* functionality are core to satisfy the key drivers, while the viewing and export functionality is only *nice to have*.

### 2.4.7 Quality Requirements

The following qualities need to be specified quantitatively:

**system response** Determines the speed and satisfaction of preparing the diagnosis by means of the workstation.

**system throughput** As defined by the typical case.

**image quality** Required for preparation of the diagnosis on screen and for diagnosis from film. Specific quality requirements exists for the relation between image and annotation:

**annotation** The relation between annotation and image is clinically relevant and must be reproducible.

**material cost** The cost price of the system must fit in the cost target.

**operational cost** The operational cost (cost of consumables, energy, et cetera) must fit in the operational target.

### 2.4.8 Interface Requirements

Key part of the external interfaces is the shared information model that facilitates interoperability between different systems. The cooperating systems must adhere to a shared information model. Elements of such an information model are:

**viewing settings** Sharing the same presentation model to guarantee the same displayed image at both systems.

**patient, exam info** Sharing the same meta information for navigation and identification.

## 2.5 Functionality

Figure 2.10 shows a retrospective overview of the development of functionality over time. The case described here focuses on the period 1992, and 1993. However the vision of the product group was to design a platform that could serve many applications and modalities. The relevance of this retrospective overview is to show the expected (and realized!) increase of functionality.

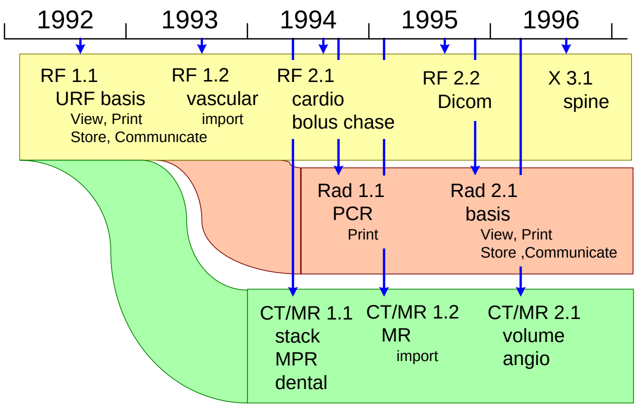


Figure 2.10: Retrospective functionality roadmap

The first release of the product served the URF market and provided the so-called view-print-store-communicate functionality. We already saw in figure 2.9 that a lot of functionality is hidden in this simple quartet.

Release 1.2 added import from vascular systems to the functionality. Cardio import and functionality and bolus chase reconstruction were added in release 2.1. Cardio functionality in this release consisted mostly of analysis functions, such as cardiac volume and wall motion analysis. The bolus chase reconstruction takes a series of exposures as input an fuses them together into a single large overview, typically used to follow the bolus chase through the legs.

Release 2.2 introduced DICOM as the next generation of information model standard. The first releases were based on the ACR/NEMA standard, DICOM succeeded this standard. Note that the installed base required prolongation of ACR/NEMA-based image exchange. Release 3.1 added spine reconstruction and analysis. The spine reconstruction is analogous to the bolus chase reconstruction, however spine specific analysis was also added.

On the basis of the URF-oriented R1.1 workstation a CT/MR workstation was developed, which was released in 1994. CT/MR images are slice-based (instead of projection-based as in URF), which prompted the development of a stack view application (fast scrolling through a stack of images). Reconstruction of oblique and curved slices is supported by means of MPR (Multi Planar Reformatting). A highly specialized application was built on top of these applications. This was a dental package, allowing viewing of the jaws, with the molars, and with the required cross sections.

Release 2.1 of the CT/MR workstation added a much more powerful volume viewing application and a more specialized angio package, with viewing and analysis capability.

Also derived from the RF workstation a radiography workstation was built. R1.1 of this system was mostly a print server, while R2.1 supported the full view-print-store-communicate functionality.

The *commercial*, *service* and *goods flow* decompositions were present as part of the formalized documentation (TPD).

## 2.6 Interoperability via Information Model

The health care industry is striving for interoperability by working on standard exchange formats and protocols. The driving force behind this standardization is the ACR/NEMA, in which equipment manufacturers participate in the standardization process.

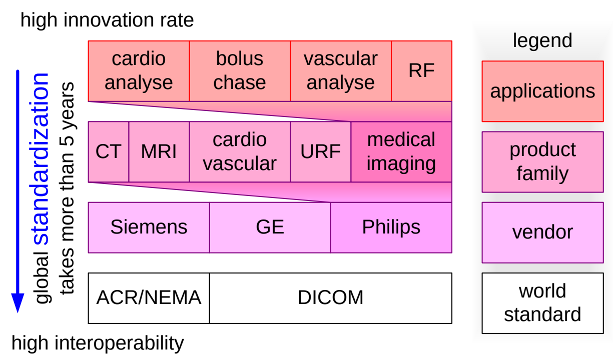


Figure 2.11: Information model, standardization for interoperability

Standardization and innovation are often opposing forces. The solution is often found in defining an extendable format. and in standardization of the mature functionality. Figure 2.11 shows the approach as followed by the medical imaging product group. The communication infrastructure and the mature application information is standardized in DICOM. The new autoprint functionality was standardized at vendor level. Further standardization of autoprint is pushed via participation in DICOM work groups.

A good strategy is to use the standard data formats as much as possible, and to build vendor specific extensions as long as the required functionality is not yet standardized. The tension between standardization and innovation is also present at many levels: between vendors, but also between product groups in the same company and also between applications within the same product. At all levels the same strategy is deployed. Product family specific extensions are made as long as no standard vendor solution is available.

This strategy serves both needs: interoperability for mature, well defined functionality and room for innovative exploration.

The information model used for import, export and storage on removable media is one of the most important interfaces of these systems. The functionality and the behavior of the system depend completely on the availability and correctness of this information. The specification of the information model and the level of adherence and the deviations is a significant part of the specification and the specification effort. A full time architect created and maintained this part of the specification.

## 2.7 Conclusion

The context of the system in the radiology department has been shown by means of multiple models and diagrams: clinical context with stakeholders, financial context, application layers in IT systems, a reference model for health care automation, clinical information flow, and URF market segmentation. Figure 2.12 shows the coverage in actual documentation of the submethods discussed in part II. The actual documentation of the *Customer Objectives* and *Application* views was quite poor, as indicated in Figure 2.12. Most of the models and diagrams shown here were not present in the documentation of 1992. The application of the system has been shown as typical case. The typical case was documented explicitly in 1992. The key driver graph, discussed in Section 2.4, is also a reconstruction in retrospect. The limited attention for the *Customer Objectives* and *Application* views is one of the main causes of the late introduction of printing functionality.

The functional view was well documented in 1992. The functions and features have been discussed briefly in Section 2.5. The functions and features were well documented in so-called *Functional Requirement Specifications*. Interoperability, discussed briefly in Section 2.6, was also documented extensively. Figure 2.12 shows that the coverage of the *Functional* view is high.

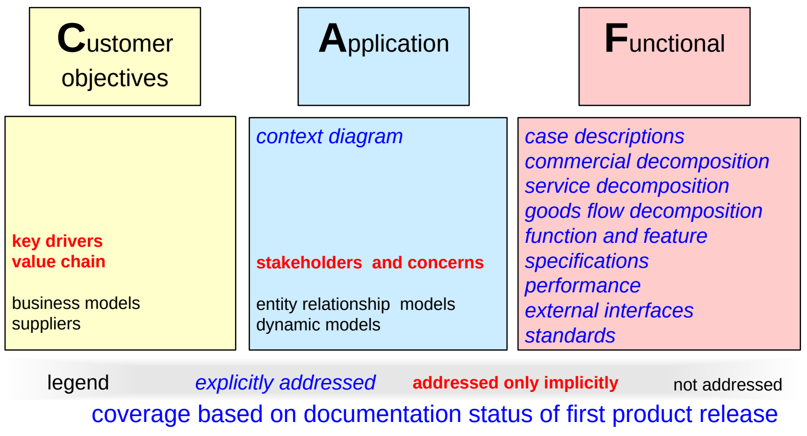
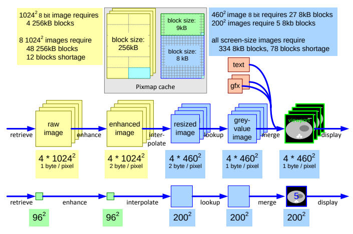


Figure 2.12: Coverage of submethods of the CAF views

# Chapter 3

# Medical Imaging Workstation: CR Views



## 3.1 Introduction

The conceptual and realization views are described together in this chapter. The realization view, with its specific values, brings the concepts more alive.

Section 3.2 describes the processing pipeline for presentation and rendering, and maps the user interface on these concepts. Section 3.4 describes the concepts needed for memory management, and zooms in on how the memory management is used to implement the processing pipeline. Section 3.3 describes the software architecture. Section 3.5 describes how the limited amount of CPU power is managed.

The case material is based on actual data, from a complex context with large commercial interests. The material is simplified to increase the accessibility, while at the same time small changes have been made to remove commercial sensitivity. Commercial sensitivity is further reduced by using relatively old data (between 8 and 13 years in the past). Care has been taken that the value of the case description is maintained.

## 3.2 Image Quality and Presentation Pipeline

The user views the image during the examination at the console of the X-ray system, mostly to verify the image quality and to guide the further examination. Later the same image is viewed again from film to determine the diagnosis and to prepare the report. Sometimes the image is viewed before making a hardcopy to optimize the image settings (contrast, brightness, zoom). The user expects to see the same image at all work-spots, independent of the actual system involved.

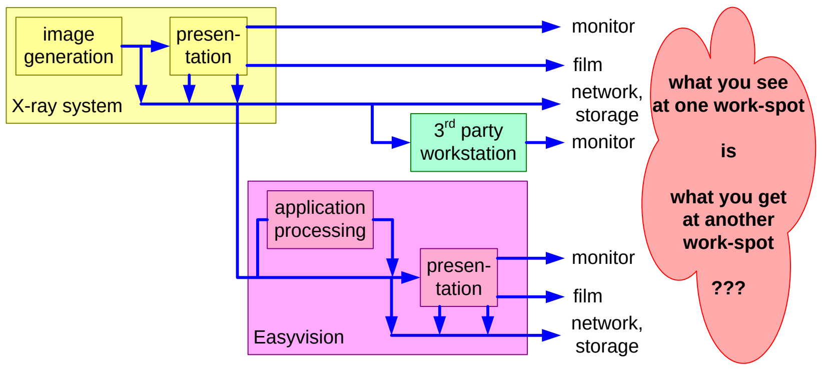


Figure 3.1: The user expectation is that an image at one work-spot looks the same as at other work-spots. This is far from trivial, due to all data paths and the many parties that can be involved

Figure 3.1 shows many different possible work-spots, with different media. The user expects *What You See Is What You Get* (WYSIWYG) everywhere. From an implementation point of view this is far from trivial. To allow optimal handling of images at other locations most systems export images halfway their internal processing pipeline: acquisition specific processing is applied, rendering specific processing is not applied, but the rendering settings are transferred instead. All systems using these intermediate images need to implement the same rendering in order to get the same image perception. The design of these systems is strongly coupled, due to the shared rendering know-how.

Figure 3.2 shows the rendering pipeline as used in the medical imaging workstation. Enhancement is a filter operation. The coefficients of the enhancement kernel are predefined in the acquisition system. The interpolation is used to resize the image from acquisition resolution to the desired view-port (or film-port) size. The grey-levels for display are determined by means of a lookup table. A lookup table (LUT) is a fast and flexible implementation of a mapping function. Normally the mapping is linear: the slope determines the contrast and the vertical offset the brightness of the image. Finally graphics and text are superimposed on the image, for instance for image identification and for annotations by the user.

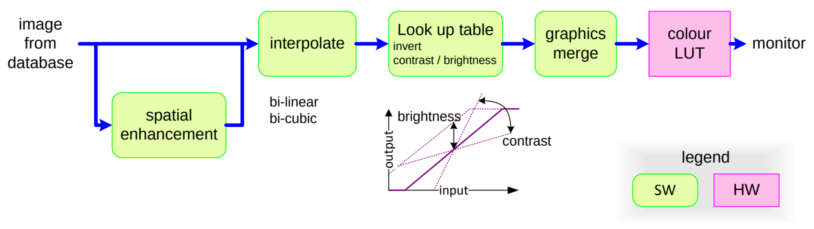


Figure 3.2: The standard presentation pipeline for X-ray images

The image interpolation algorithm used depends on desired image quality and on available processing time. Bi-linear interpolation is an interpolation with a low-pass filter side effect, by which the image becomes less sharp. An ideal interpolation is based on a convolution with a sinc-function ($sin(x)/x$). A bi-cubic interpolation is an approximation of the ideal interpolation. The bi-cubic interpolation is parameterized. The parameter settings determine how much the interpolation causes low pass or high pass filtering (blurring or sharpening). These bi-cubic parameter choices are normally not exported to the user interface, the selection of values requires too much expertise. Instead, the system uses empirical values dependent on the interpolation objective.

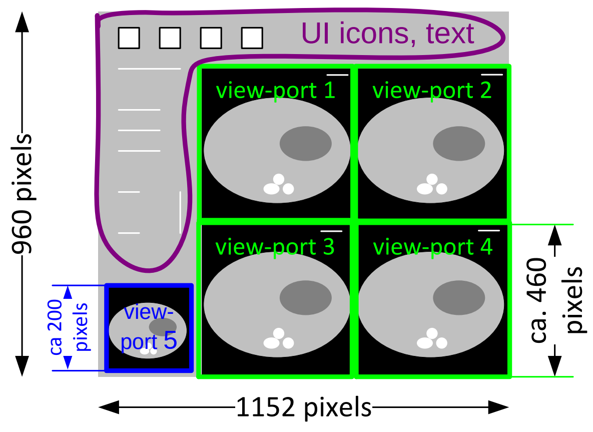


Figure 3.3: Quadruple view-port screen layout

The monitor screen is a scarce resource of the system, used for user interface control and for the display of images. The screen is divided in smaller rectangular windows. Windows displaying images are called view-ports. Every view-port uses its own instantiation of a viewing pipeline. Figure 3.3 shows an example of a screen layout, viewing four images simultaneously. At the bottom left a fifth view-port is used for navigational support, for instance in case of zooming this view-port functions as a roadmap, enabling direct manipulation of the zoom-area. The fifth view-port also has its own viewing pipeline instance.

The concepts visible in this screen layout are view-ports, icons, text, an image area (with the 4 main view-ports), and a user interface area with navigation support. The figure adds a number of realization facts, such as the total screen-size, and the size of the view-ports. The next generation of this system used the same concepts, but the screen size was 1280*1024, resulting in slightly larger view-ports and a slightly larger ratio between image area and user interface area.

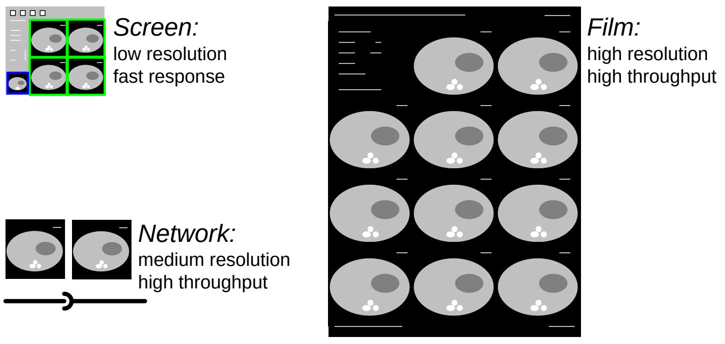


Figure 3.4: Rendered images at different destinations

At all places where source images have to be rendered into viewable images an instance of the presentation pipeline is required. Note that the characteristics of the usage of the presentation pipeline in these different processes vary widely. Figure 3.4 shows three different destinations for rendered images, with the different usage characteristics.

## 3.3 Software Specific Views

The execution architecture of Easyvision is based on UNIX-type processes and shared libraries. Figure 3.5 shows the process structure of Easyvision. Most processes can be associated with a specific hardware resource, as shown in this figure. Core of the Easyvision software architecture is the database. The database provides *fast*, *reliable*, *persistent* storage and it provides *synchronization* by means of active data. The concept of active data is based on the *publish-subscribe pattern* [3] that allows all users of a some information to be notified when changes in the information occur. Synchronization and communication between processes always takes place via this database.

Figure 3.5 shows four types of processes: *client processes*, *server processes database process*, and *operational processes*. A client interacts with a user (remote

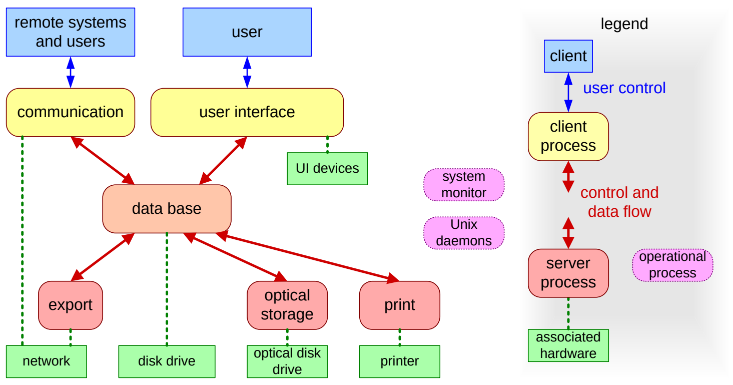


Figure 3.5: Software processes or tasks running concurrently in Easyvision

or direct), while the servers perform their work in the background. The database connects these two types of processes. Operational processes belong to the computing infrastructure. Most operational processes are created by the operating system, the so called daemons. The system monitoring processes is added for exception handling purposes. The system monitor detects hanging processes and takes appropriate action to restore system operation.

A process as unit of design is used for multiple reasons. The criteria used to determine the process decomposition are:

**management of concurrency** Activities that are concurrent run in separate processes.

**management of shared devices** A shared device is managed by a server process.

**unit of memory budget** Measurement of memory use at process level is supported by multiple tools.

**unit of distribution over multiple processors** A process can be allocated to a processor, without the need to change the code within the process.

**unit of exception handling** Faults are contained within the process boundaries. The system monitor observes at process level, because the operating system provides the means at process level.

Manageability, visibility and understandability benefit from a limited number of processes. One general rule is to minimize the amount of processes, in the order of magnitude of ten processes.

The presentation pipeline, as depicted in Figure 3.2, is used in the *user interface* process, the *print* server and the *export server*.

Figure 3.6 shows the software from the dependency point of view. Software in higher layers depends on, has explicit knowledge of, lower layers of the software.

Software in the lower layers should not depend on, or have explicit knowledge of software in higher layers.

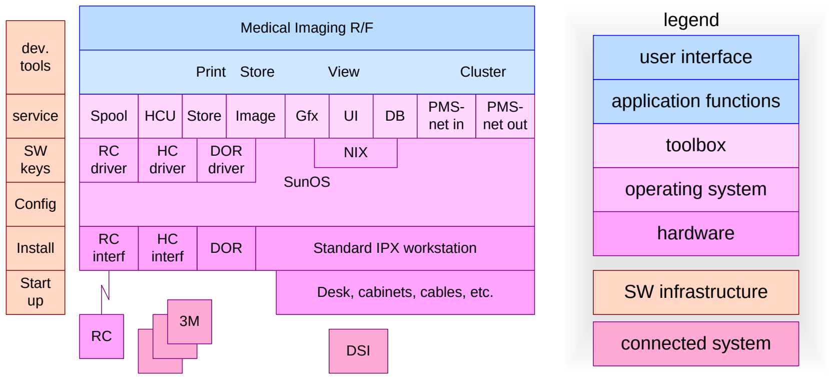


Figure 3.6: Simplified layering of the software

The caption of Figure 3.6 explicitly states this diagram to be simplified. The original design of this software did not use the layering concept. The software has been restructured in later years to make the dependency as layering explicit. The actual number of layers based on larger packages did exceed 15. Reality is much more complex than this simplified diagram suggests.

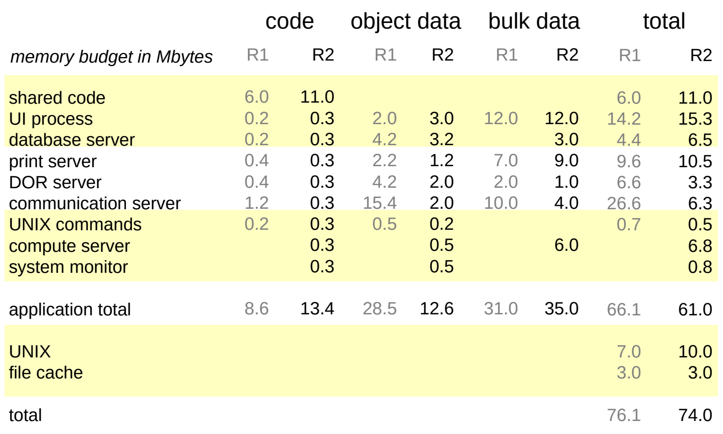
## 3.4 Memory Management

The amount of memory in the medical imaging workstation is limited for cost reasons, but also for simple physical reasons: the workstation used at that moment did not support more than 64 MByte of physical memory. The workstation and operating system did support virtual memory, but for performance reasons this should be used sparingly.

A memory budget is used to manage the amount of memory in use. Figure 3.7 shows the memory budgets of release 1 and release 2 of Easyvision RF side by side. Three types of memory are distinguished: *program or code*, read-only from operating system point of view, *object data*, dynamically allocated and deallocated in a heap-based fashion, and *bulk data* for large consecutive memory areas, mostly used for images.

Per process, see Section 3.3, the typical amount of memory per category is specified. The memory usage of the operating system is also specified. The dynamic libraries, that contain the code shared between processes, is explicitly visible in the budget.

The figure shows the realization for two successive releases, for which we can observe that the concepts are stable, but that the realization changes significantly.



| *memory budget in Mbytes* | code | | object data | | bulk data | | total | |
|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R1 | R2 | R1 | R2 | R1 | R2 |
| shared code | 6.0 | 11.0 | | | | | 6.0 | 11.0 |
| UI process | 0.2 | 0.3 | 2.0 | 3.0 | 12.0 | 12.0 | 14.2 | 15.3 |
| database server | 0.2 | 0.3 | 4.2 | 3.2 | | 3.0 | 4.4 | 6.5 |
| print server | 0.4 | 0.3 | 2.2 | 1.2 | 7.0 | 9.0 | 9.6 | 10.5 |
| DOR server | 0.4 | 0.3 | 4.2 | 2.0 | 2.0 | 1.0 | 6.6 | 3.3 |
| communication server | 1.2 | 0.3 | 15.4 | 2.0 | 10.0 | 4.0 | 26.6 | 6.3 |
| UNIX commands | 0.2 | 0.3 | 0.5 | 0.2 | | | 0.7 | 0.5 |
| compute server | | 0.3 | | 0.5 | | 6.0 | | 6.8 |
| system monitor | | 0.3 | | 0.5 | | | | 0.8 |
| application total | 8.6 | 13.4 | 28.5 | 12.6 | 31.0 | 35.0 | 66.1 | 61.0 |
| UNIX | | | | | | | 7.0 | 10.0 |
| file cache | | | | | | | 3.0 | 3.0 |
| total | | | | | | | 76.1 | 74.0 |

Figure 3.7: Memory budget of Easyvision release 1 and release 2

Release 1 used a rather straightforward communication server, operating on all import streams in parallel, keeping everything in memory. This is very costly with respect to memory. R2 serializes the memory use of different import streams and uses the memory in a more pipelined way. These changes result in a significant reduction of the memory being used. In the same time frame the supplier dictated a new operating system, SunOS was end-of-life and was replaced by Solaris 2. This had a negative impact on the memory consumption; the budget shows an increase of 7 MByte to 10 MByte for the UNIX operating system.

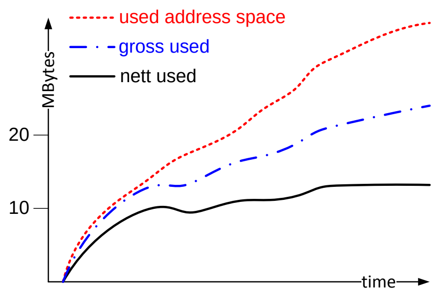


Figure 3.8: Memory fragmentation increase. The difference between gross used and nett used is the amount of unusable memory due to fragmentation

The decomposition in object data and bulk data is needed to prevent memory fragmentation. Fragmentation of memory occurs when the allocation is dynamic with different sizes of allocated memory over time. The fragmentation increases over time. Due to the paging of the virtual memory system not all fragmentation is disastrous. Figure 3.8 shows the increase of the amount of memory over time.

The net amount of memory stabilizes after some time, but the gross amount of memory increases due to ongoing fragmentation. The amount of virtual memory in use (and the address space) is increasing even more, however a large part of this virtual memory is paged out and is not really a problem.

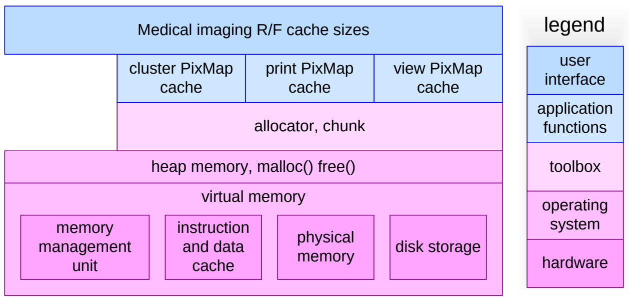


Figure 3.9: Cache layers at the corresponding levels of Figure 3.6

The hardware and operating system support fast and efficient memory-based on hardware caching and virtual memory, the lowest layer in Figure 3.9. The application allocates memory via the heap memory management functions malloc() and free(). From an application point of view a sheer infinite memory is present, however the speed of use depends strongly on the access patterns. Data access with a high locality are served by the data cache, which is the fastest (and smallest) memory layer. The next step in speed and size (slower, but significantly larger) is the physical memory. The virtual memory, mostly residing on disk, is the slowest but largest memory layer.

The application software does not see or control the hardware cache or virtual memory system. The only explicit knowledge in the higher software layers of these memory layers is in the dimensioning of the memory budgets as described later.

The toolbox layer provides anti-fragmentation memory management. This memory is used in a cache like way by the application functions, based on a *Least Recently Used* algorithm. The size of the caches is parameterized and set in the highest application layer of the software.

The medical imaging workstation deploys pools with fixed size blocks to minimize fragmentation. A two level approach is taken: pools are allocated in large *chunks*, every chunk is managed with fixed size *blocks*. For every chunk is defined which bulk data sizes may be stored in it.

Figure 3.10 shows the three chunk sizes that are used in the memory management concepts chunks, block sizes and bulk data sizes as used in Easyvision RF. One chunk of 1 MByte is dedicated for so-called *stamp* images, 96*96 down scaled images, used primarily for visual navigation (for instance pictorial index). The block size of 9 kbytes is exactly the size of a stamp image. A second chunk of 3 MBytes is used for large images, for instance images with the original acquisition

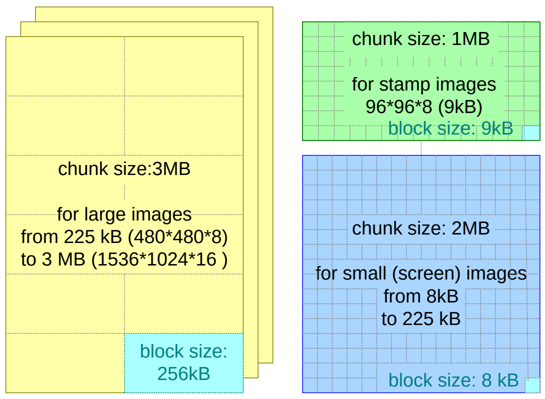


Figure 3.10: Memory allocators as used for bulk data memory management in Easyvision RF

resolution. Small images, such as images at display resolution, will be allocated in the third chunk of 2 MBytes. The dimensioning of the block and chunk sizes is based on a priori know-how of the application of the system, as described in Section 2.3. The block sizes in the latter two chunks are 256 kbytes for large images and 8 kbytes for small images. These block sizes result in balanced and predictable memory utilization and fragmentation within a chunk.

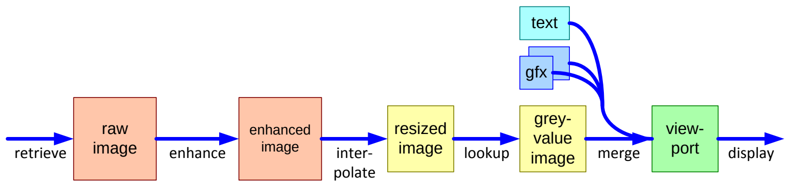
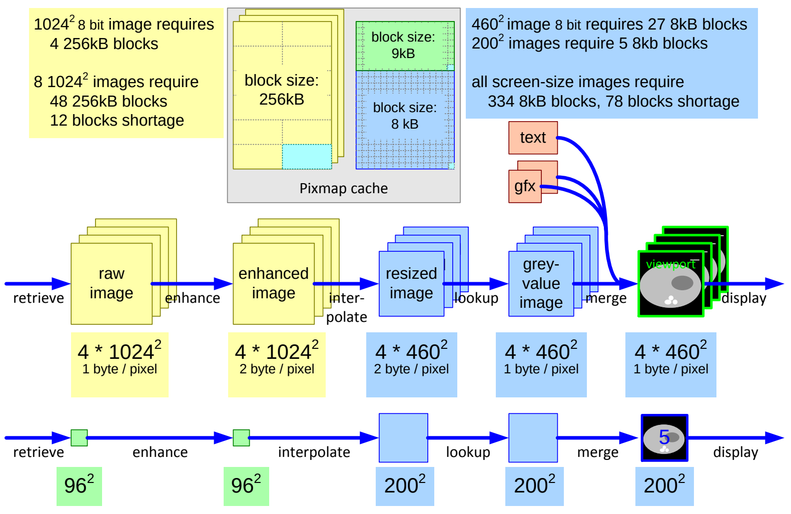


Figure 3.11: Intermediate processing results are cached in an application level cache

The chunks are used with cache like behavior: images are kept until the memory is needed for other images. Figure 3.11 shows the cached intermediate results. This figure is a direct transformation of the viewing pipeline in Figure 3.2, with the processing steps replaced by arrows and the data-arrows replaced by stores. In Section 3.5 the gain in response time is shown, which is obtained by caching the intermediate images.

Figure 3.12 shows how the *chunks* are being used in quadruple viewing (Figure 3.3). The $1024^2$ images with a depth of 1 or 2 bytes will be stored in the 3 MB chunks. The smaller interpolated images of $460^2$ will go into the 2 MB chunks, requiring 27 blocks of 8kB for an 1 byte pixel depth or 54 blocks for 2 2 bytes per pixel.



$1024^2$ 8 bit image requires 4 256kB blocks
8 $1024^2$ images require 48 256kB blocks 12 blocks shortage
block size: 256kB
block size: 9kB
block size: 8 kB
Pixmap cache
$460^2$ image 8 bit requires 27 8kB blocks
$200^2$ images require 5 8kb blocks
all screen-size images require 334 8kB blocks, 78 blocks shortage
text
gfx
retrieve
raw image
enhance
enhanced image
inter-polate
resized image
lookup
grey-value image
merge
viewport
display
4 * $1024^2$ 1 byte / pixel
4 * $1024^2$ 2 byte / pixel
4 * $460^2$ 2 byte / pixel
4 * $460^2$ 1 byte / pixel
4 * $460^2$ 1 byte / pixel
retrieve
enhance
interpolate
lookup
merge
display
5
$96^2$
$96^2$
$200^2$
$200^2$
$200^2$

Figure 3.12: Example of allocator and cache use. In this use case not all intermediate images fit in the cache, due to a small shortage of blocks. The performance of some image manipulations will be decreased, because the intermediate images will be regenerated when needed.

Also the screen size images of the navigation view-port fall in the range that maps on the 2 MB chunk, requiring 5 blocks per $200^2$ image.

Everything added together requires more blocks than available in the 2 and 3 MB chunks. The cache mechanism will sacrifice the least recently used intermediate results.

For memory and performance reasons the navigation view-port is using the stamp image as source image. This image, which is shown in a small view-port at the left hand side of the screen, is only used for navigational support of the user interface. Response time is here more important than image quality.

The print server uses a different memory strategy than the user interface process, see Figure 3.13. The print server creates the film-image by rendering the individual images. The film size of 4k*5k images is too large to render the entire film at once in memory: 20 Mpixels, while the memory budget allows 9 Mbyte of bulk data usage. The film image itself is already more than the provided memory budget!

The film image is built up in horizontal bands, which are sent to the laser printer. The size of the stroke is chosen such that input image + intermediate results + 2 bands (for double buffering) fit in the available bulk data budget. At the same time the band should not be very small because the banding increases the overhead and some duplicate processing is sometimes needed because of edge effects.

The print server uses the same memory management concepts as shown in the figure with cache layers, Figure 3.9. However the application level caching does

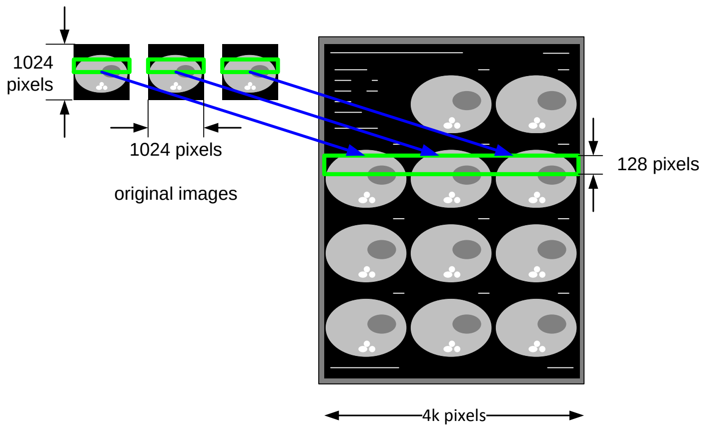


Figure 3.13: Print server is based on different memory strategy, using bands

not provide any significant value for this server usage, because the image data flow is straightforward and predictable.

## 3.5 CPU Usage

The CPU is a limited resource for the Easyvision. The performance and throughput of the system depend strongly on the available processing power and the efficiency of using the processing power. CPU time and memory can be exchanged partially, for instance by using caches to store intermediate results.

Figure 3.14 shows typical update speeds and processing times for a single image user interface layout. Contrast brightness (C/B in the figure) changes must be fast, to give immediate visual feedback when turning a contrast or brightness wheel. Working on the cached resized image about 7 updates per second are possible, which is barely sufficient. The gain of the cached design relative to the non-cached design is about a factor 8 (7 updates per second versus 0.9 updates per second). Zooming and panning is done with an update rate of 3 updates per second. The performance gain for zooming and panning is from application viewpoint less important, because these functions are used only exceptionally in the daily use.

Retrieving the next image (also a very frequent user operation), requires somewhat more than a second, which was acceptable at that moment in time. This performance is obtained by slightly compromising the image quality: a bilinear interpolation is used for resizing, instead of the better bi-cubic interpolation. For the monitor, with its limited resolution this is acceptable, for film (high resolution, high brightness) bi-cubic interpolation is required.

For background tasks a CPU budget is used, expressed in CPU seconds per

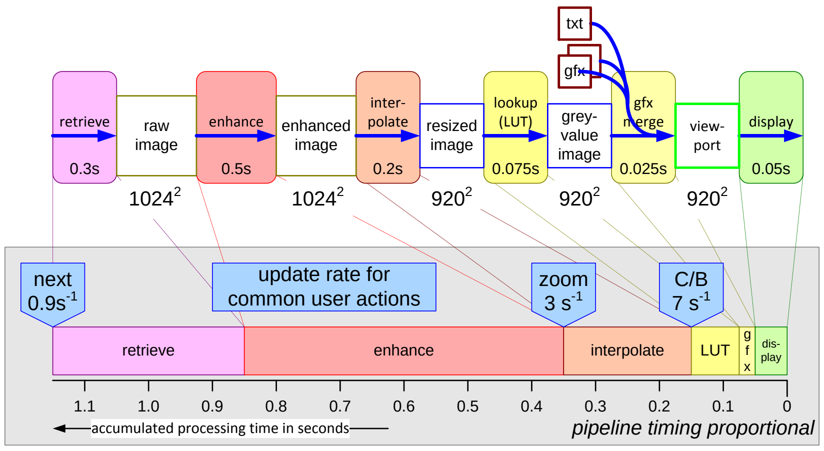


Figure 3.14: The CPU processing times are shown per step in the processing pipeline. The processing times are mapped on a proportional time line to visualize the viewing responsiveness

Mega-byte or Mega-pixel. This budget is function-based: importing and printing. Most background jobs involve a single server plus interaction with the database server.

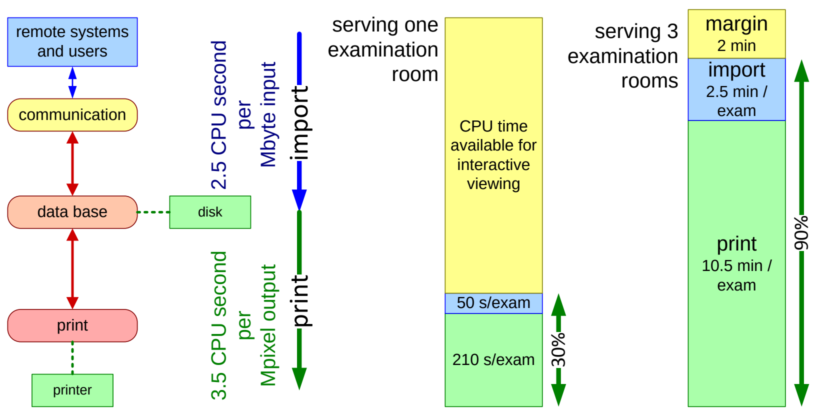
Two use cases are relevant: interactive viewing, with background jobs, and pure print serving. For interactive response circa 70% of CPU time should be available, while the load of printing for three examination rooms, which is a full throughput case, must stay below 90% of the available CPU time. Figure 3.15 shows the load for serving a single examination room and for serving three examination rooms. Serving a single examination room takes 260 seconds of CPU time per examination of 15 minutes, leaving about 70% CPU time for interactive viewing. Serving three examination rooms takes 13 minutes of CPU time per 15 minutes of examinations, this is just below the 90%.

## 3.6 Measurement Tools

The resource design as described above is supported in the implementation by means of a few simple, but highly effective measurement tools. The most important tools are: *Object Instantiation Tracing*, *standard Unix utilities* and a *heap viewer*.

The resource usage is measured at well defined moments in time, by means of events. The entire software is event-based. The event for resource measurement purposes can be fired by programming it at the desired point in the code, or by a user interface event, or by means of the Unix command line.

The resource usage is measured twice: before performing the use case under



remote systems and users
communication
data base
disk
print
printer
2.5 CPU second per Mbyte input
import
3.5 CPU second per Mpixel output
print
serving one examination room
CPU time available for interactive viewing
50 s/exam
210 s/exam
30%
serving 3 examination rooms
margin 2 min
import 2.5 min / exam
print 10.5 min / exam
90%

Figure 3.15: Server CPU load. For a single examination room sufficient CPU time is left for interactive viewing. Serving three examination rooms fits in 90% of the available CPU time.

study and afterwards. The measurement results show both the changes in resource usage as well as the absolute numbers. The initialization often takes more time in the beginning, while in a steady running system no more initialization takes place. Normally the real measurement is preceded by a set of actions to bring the system in a kind of steady state.

Note that the budget definitions and the *Unix utilities* fit well together, by design. The types of memory budgeted are the same as the types of memory measured by the Unix utilities. The typically used Unix utilities are:

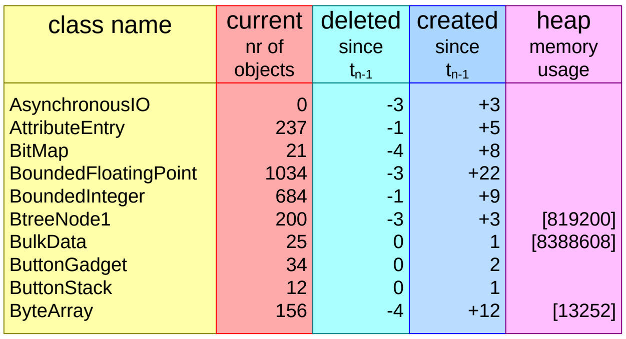
**ps** process status and resource usage per process

**vmstat** virtual memory statistics

**kernel resource stats** kernel specific resource usage

The *heap-viewer* shows the free and allocated memory blocks in different colors, comparable with the standard Windows disk defragmentation utilities.

The *Object Instantiation Tracing* (OIT) keeps track of all object instantiations and disposals. It provides an absolute count of all the objects and the change in the number of objectives relative to the previous measurement. The system is programmed with Objective-C. This language makes use of run-time environment, controlling the creation and deletion of objects and the associated housekeeping. The creation and deletion operations of this run-time environment were rerouted via a small piece of code that maintained the statistics per class of object instantiations and destructions. At the moment of a trigger this administration was saved in readable form. The few lines of code (and the little run time penalty) have paid



| class name | current nr of objects | deleted since $t_{n-1}$ | created since $t_{n-1}$ | heap memory usage |
|---|---|---|---|---|
| AsynchronousIO | 0 | -3 | +3 | |
| AttributeEntry | 237 | -1 | +5 | |
| BitMap | 21 | -4 | +8 | |
| BoundedFloatingPoint | 1034 | -3 | +22 | |
| BoundedInteger | 684 | -1 | +9 | |
| BtreeNode1 | 200 | -3 | +3 | [819200] |
| BulkData | 25 | 0 | 1 | [8388608] |
| ButtonGadget | 34 | 0 | 2 | |
| ButtonStack | 12 | 0 | 1 | |
| ByteArray | 156 | -4 | +12 | [13252] |

Figure 3.16: Example output of OIT (Object Instantiation Tracing) tool

many many times. The instantiation information gives an incredible insight in the internal working of the system.

The *Object Instantiation Tracing* also provided heap memory usage per class. This information could not be obtained automatically. At every place in the code where malloc and free was called some additional code was required to get this information. This instrumentation has not been completed entirely, instead the 80/20 rule was applied: the most intensive memory consumers were instrumented to cover circa 80% of the heap usage.

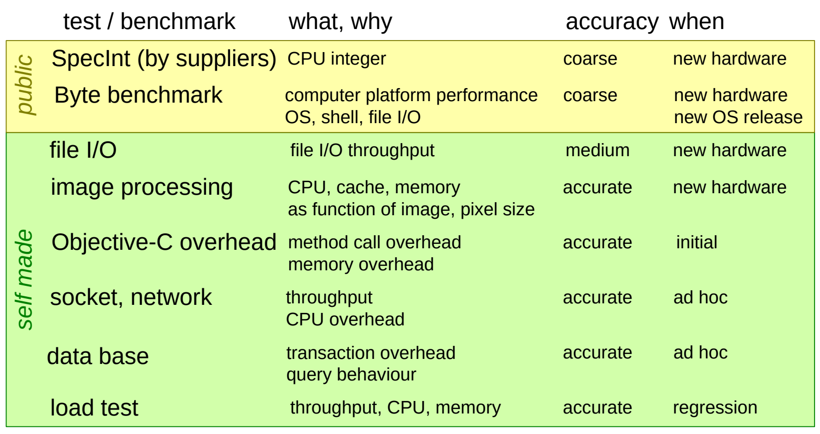
Figure 3.16 shows an example output of the OIT tool. Per class the current number of objects is shown, the number of deleted and created objects since the previous measurement and the amount of heap memory in use. The user of this tool knows the use case that is being measured. In this case, for example, the *next image* function. For this simple function 8 new BitMaps are allocated and 3 AsynchronousIO objects are created. The user of this tool compares this number with his expectation. This comparison provides more insight in design and implementation.

Figure 3.17 shows an overview of the benchmarking and other measurement tools used during the design. The overview shows per tool what is measured and why, and how accurate the result is. It also shows when the tool is being used.

The Objective-C overhead measurements, to measure the method call overhead and the memory overhead caused by the underlying OO technology, is used only in the beginning. This data does not change significantly and scales reasonably with the hardware improvements.

A set of coarse benchmarking tools was used to characterize new hardware options, such as new workstations. These tools are publicly available and give a coarse indication of the hardware potential.

The application critical characterization is measured by more dedicated tools, such as the image processing benchmark, which runs all the algorithms with different image and pixel sizes. This tool is home made, because it uses the actual image



| | test / benchmark | what, why | accuracy | when |
|---|---|---|---|---|
| public | SpecInt (by suppliers) | CPU integer | coarse | new hardware |
| public | Byte benchmark | computer platform performance<br>OS, shell, file I/O | coarse | new hardware<br>new OS release |
| self made | file I/O | file I/O throughput | medium | new hardware |
| self made | image processing | CPU, cache, memory<br>as function of image, pixel size | accurate | new hardware |
| self made | Objective-C overhead | method call overhead<br>memory overhead | accurate | initial |
| self made | socket, network | throughput<br>CPU overhead | accurate | ad hoc |
| self made | data base | transaction overhead<br>query behaviour | accurate | ad hoc |
| self made | load test | throughput, CPU, memory | accurate | regression |

Figure 3.17: Overview of benchmarks and other measurement tools

processing library used in the product. The outcome of these measurements were used to make design optimizations, both in the library itself as well as in the use of the library.

Critical system functionality is measured by dedicated measurement tools, which isolate the desired functionality, such as file I/O, socket, networking and database.

The complete system is put under load conditions, by continuously importing and exporting data and storing and retrieving data. This load test was used as regression test, giving a good insight in the system throughput and in the memory and CPU usage.

## 3.7 Conclusion

This chapter described several decompositions: a functional decomposition of the image processing pipeline, a construction decomposition in layers and a process decomposition of the software. The image quality, throughput and response time have been discussed and especially the design choices that have been made to achieve the desired performance level. The design considerations show that design choices are related to consequences in multiple qualities and multiple CAFCR views. Reasoning over multiple CAFCR views and multiple qualities is needed to find an acceptable design solution. All information presented here was explicitly available in product creation documentation.

A number of submethods has not been described here, such as start up and shutdown, but these aspects are covered by the documentation of 1992. Figure 3.18 shows the coverage of the submethods described in part II by the documentation of the first release. This coverage is high for most submethods. Safety, reliability and

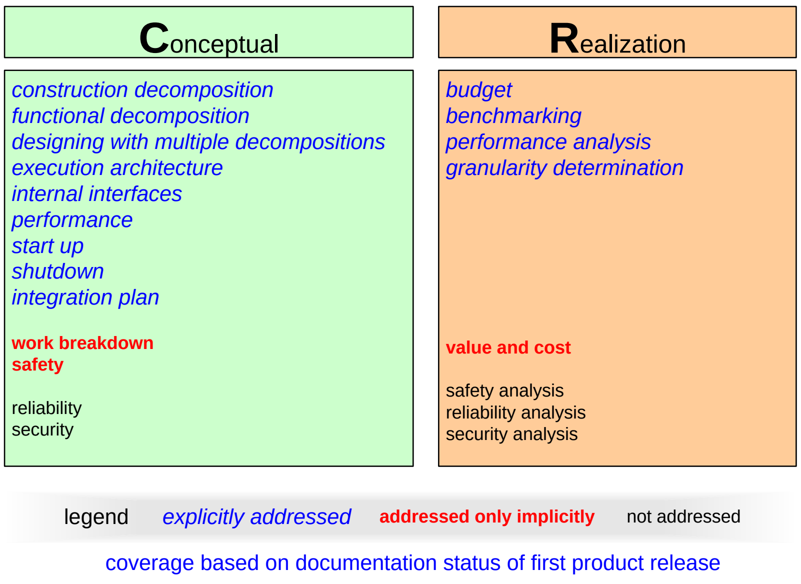
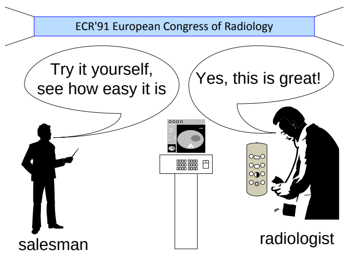


Figure 3.18: Coverage of submethods of the CR views

security were not covered by the documentation in 1992, but these aspects were added in later releases of the product.

# Chapter 4

# Story Telling in Medical Imaging


ECR'91 European Congress of Radiology
Try it yourself, see how easy it is
Yes, this is great!
salesman
radiologist

## 4.1 Introduction

Stories have not been used explicitly in the development of the medical imaging workstation. Informally however a few stories were quite dominant in creating insight and focus. These informal stories do not meet all criteria described in Chapter **??**, especially the specificity is missing. The typical case, as described in Chapter 2 is complementary to the stories. We now add the required specific quantitative details.

The main stories dominating the development were:

**The sales story** how to capture the interest of the radiologist for the product, see Section 4.2.

**The radiologist at work** describing the way a radiologist works. This story explains why the radiologist is **not** interested in viewing, but very interested in films, see Section 4.3.

**The gastro intestinal examination** how the URF system is used to examine patients with gastro intestinal problems. This story is not described here, because it is outside the scope of the discussed thread of reasoning

Section 4.4 relates the stories to the CAFCR model and discusses the criteria for stories as described in Chapter **??**.

## 4.2 The Sales Story

The main function of the medical imaging workstation is rather invisible: layout and rendering of the medical images on film. To support the sales of the product more attractive appealing functionality was needed. The medical community is a rather conservative community, as far as technology is concerned: computers and software are mostly outside their scope. The sales approach was to provide an easy to use product, showing recognizable clinical information.

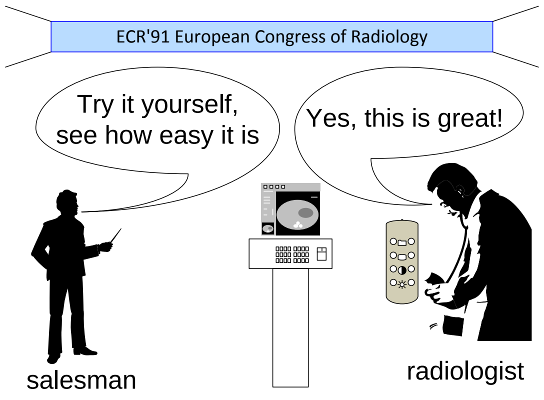

ECR'91 European Congress of Radiology
Try it yourself, see how easy it is
Yes, this is great!
salesman
radiologist

Figure 4.1: The main sales feature is easy viewing

At the European Congress of Radiology the system was shown to the radiologist. The radiologists were immediately challenged to operate the system themselves, see Figure 4.1.

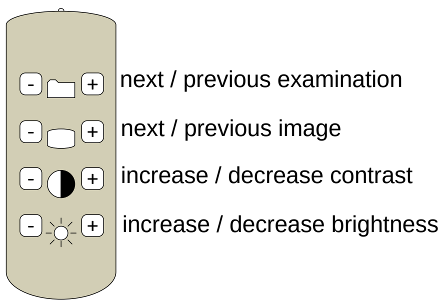

next / previous examination
next / previous image
increase / decrease contrast
increase / decrease brightness

Figure 4.2: The simple remote control makes the viewing easy

The frequently used operations were available as single button operations on the remote control, see Figure 4.2: Select the examination, by means of previous/next examination buttons; Select image by previous/next image buttons; Adapt contrast and brightness by increase/decrease buttons.

Note that this is a nice sales feature, but that in day-to-day life the radiologist does not have the time to stand behind the workstation and view the images in this way. The viewing as described in Section 4.3 is much faster and efficient.

## 4.3 The Radiologist at Work

The radiologist has the following activities that are directly related to the diagnosis of a patient: supervising the examination, viewing the images to arrive at a diagnosis, dictating a report and verifying and authorizing the textual version of the report. Figure 4.3 shows these activities.

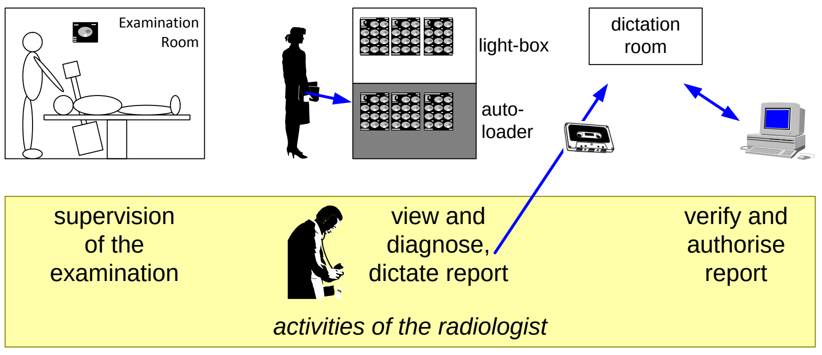

Examination Room
light-box
auto-loader
dictation room
supervision of the examination
view and diagnose, dictate report
verify and authorise report
activities of the radiologist

Figure 4.3: Radiologist work-spots and activities

The radiologist is responsible for the image acquisition in the examination room. The radiologist is not full-time present in the examination rooms, but supervises the work in multiple rooms. The radio technicians and other clinical personnel do most of the patient handling and system operation.

The films with examinations to be viewed are collected by clinical personnel and these films are attached in the right order to carriers in the auto-loader. The auto-loader is a simple mechanical device that can lift a set of films out of the store to the front of the lightbox. Pressing the button removes the current set of films and retrieves the next set of films.

The activity of viewing and determining the diagnosis takes an amazingly short time. Figure 4.4 shows this activity in some more detail. A few movements of the head and eyes are sufficient to get the overview and to zoom in on the relevant images and the relevant details. The spoken report consists of a patient identification, a few words in Latin and or some standard medical codes. The recorded

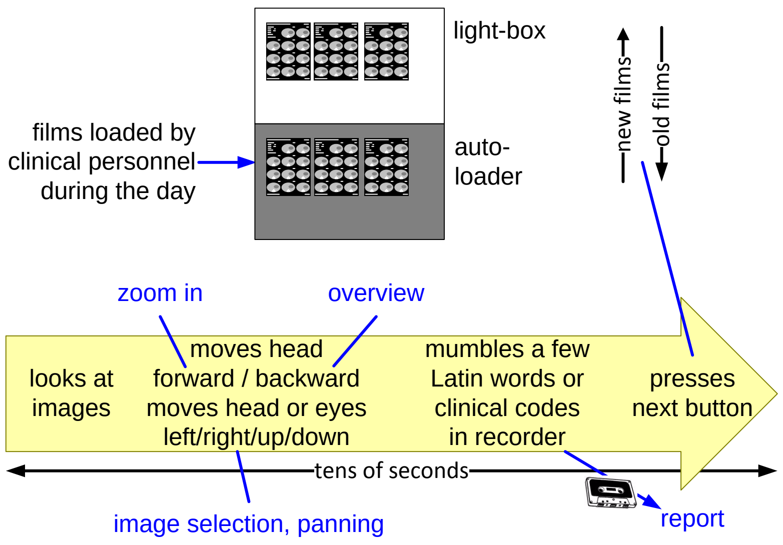

light-box
auto-loader
films loaded by clinical personnel during the day
new films
old films
zoom in
overview
looks at images
moves head forward / backward moves head or eyes left/right/up/down
mumbles a few Latin words or clinical codes in recorder
presses next button
tens of seconds
image selection, panning
report

Figure 4.4: Diagnosis in tens of seconds

spoken report is sent to the dictation department; the transcription will be verified later. The radiologist switches to the next examination with a single push on the next button of the auto-loader. This entire activity is finished within tens of seconds.

The radiologist performs this diagnosis sometimes in between patients, but often he handles a batch of patient data in one session. Later on the day the radiologist will verify and authorize the transcribed reports, again mostly in batches.

## 4.4 Towards Design

The sales story provides a lot of focus for the user interface design and especially the remote control. The functions to be available directly are defined in the story. Implicit in this story is that the performance of these functions is critical, a poor performance would kill the sales. The performance was not specified explicitly. However the implied response times were 1 second for image retrieval and 0.1 seconds for a contrast/brightness change. These requirements have a direct effect on the pipeline design and the user interface design.

Figure 4.5 shows the flow from both stories to requirements and design. It also shows the inputs that went into the stories: at the commercial side the *ease of use* as sales feature and the *film efficiency* as the main application value. The gain in film efficiency is 20% to 50% relative to the screen copy approach used originally, or in other words the typical use of 3 to 5 film sheets is reduced to 2 to 3 film sheets.

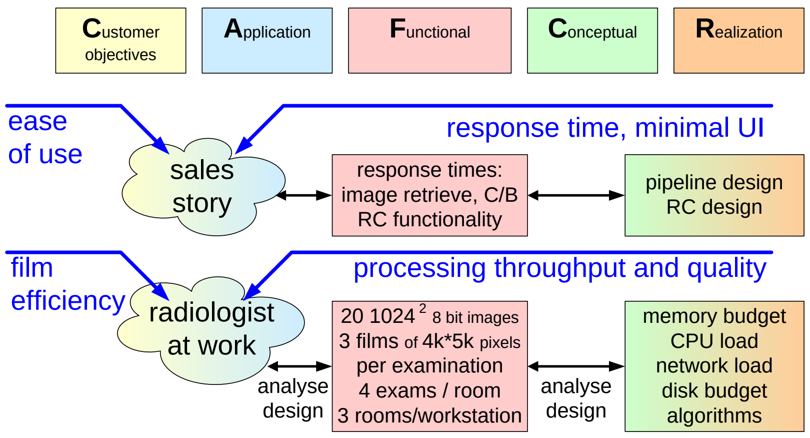


Figure 4.5: The stories in relation to the CAFCR views and the derived requirements and design choices

These numbers are based on the typical case described in Section 2.3.

The a priori know-how that the response time in a *software only* solution would be difficult, makes this a challenging story. The technical challenge in this story is to achieve the desired image quality and throughput, also in the *software only* solution.

The minimal user interface is also a design challenge. Without the sales story the user interface provided would have been much too technical, an overwhelming amount of technical possibilities would have been offered, without understanding the clinical world view.

The story of the radiologist at work, in combination with the typical case, is the direct input for the throughput specification. The throughput specification is used for the memory and disk budgets and the CPU and network loads. The image quality requirements, in combination with the loads and budgets, result in algorithmic choices.

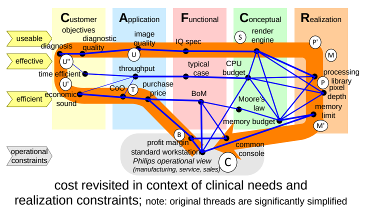
The original software completely ignored the need for printing images on film, it was not even present! The developer crew assumed that radiologists would use the workstation for “soft” diagnosis. Soft diagnosis is diagnosis from the monitor screen instead of film. A better understanding of the radiologist was needed to get the focus on the film printing functionality. The story immediately clarifies the importance of film sheets for diagnosis. The story also provides input for the functionality to create the layout of images and text on film. The auto-print functionality has been added in an extremely pragmatic way, by (mis-)using examination data fields to request printing. This pragmatic choice could only be justified by the value of this function as was made clear in this story.

## 4.5 Conclusion

Stories have not been used explicitly in the case. Somewhat less specific oral stories were provided by the marketing manager. Quantitative information was described in a typical case. The facts for quantification were provided by application managers. The presence of a quantified typical case provided the means for design, analysis and testing. The lack of explicit story, in combination with the poor coverage of the *Customer Objectives* and *Application* views as described in Chapter 2 in general, caused the late addition of the printing functionality.

# Chapter 5

# Threads of Reasoning in the Medical Imaging Case



## 5.1 Introduction

The thread of reasoning has not been applied consciously during the development of the Medical Imaging Workstation. This chapter describes a reconstruction of the reasoning as it has taken place. In Section 5.2 the outline of the thread is explained. Section 5.3 describes the 5 phases as defined in Chapter **??**:

1. Select starting point (5.3.1)
2. Create insight (5.3.2)
3. Deepen insight (5.3.3)
4. Broaden insight (5.3.4)
5. Define and extend the thread (5.3.5)

## 5.2 Example Thread

Figure 5.1 shows a set of interrelated customer objectives up to interrelated design decisions. This set of interrelated objectives, specification issues and concepts is a dominant thread of reasoning in the development of the medical imaging workstation.

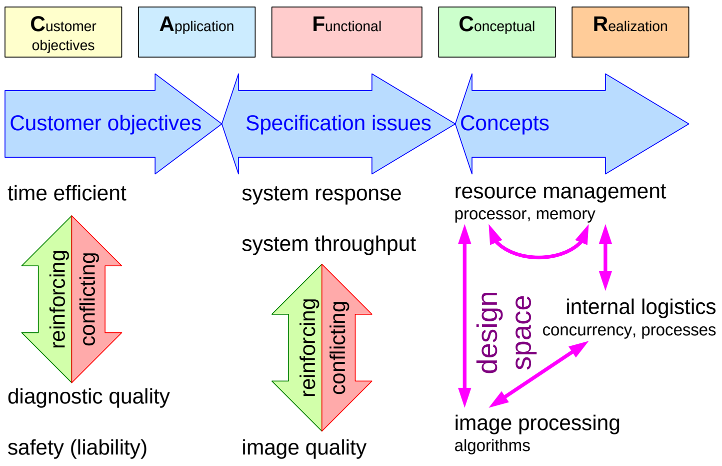


Figure 5.1: The thread of reasoning about the tension between time efficiency on the one hand and diagnostic quality, safety, and liability on the other hand. In the design space this tension is reflected by many possible design trade-offs.

The objectives of the radiologist are at the same time reenforcing and (somewhat) conflicting. To achieve a good diagnostic quality sufficient time is required or examine and study the results, which can be in conflict with time efficiency. On the other hand a good diagnostic quality will limit discussions and repeated examinations later on, by which good diagnostic quality can help to achieve time efficiency.

The customer objectives are translated into specifications. The diagnostic quality and safety/liability translate for example into image quality specifications (resolution, contrast, artefact level). A limited image quality is a primary source of a poor diagnostic quality. Artifacts can result in erroneous diagnostics, with its safety and liability consequences.

The time efficiency is achieved by system throughput. The workstation should not be the bottleneck in the total department flow or in the system response time. Waiting for results is clearly not time efficient.

Also at the specification level the reenforcing and the conflicting requirements are present. If the image quality is good, no tricky additional functions are needed to achieve the diagnostic quality. For instance if the image has a good clinical contrast to noise ratio, then no artificial contrast enhancements have to be applied. Function bloating is a primary source of decreased throughput and increased response times. The conflicting aspect is that some image quality functions are inherently time consuming and threaten the throughput and response time.

The design space is full of concepts, where design choices are needed. The concepts of *resource management*, *internal logistics* and *image processing algorithms* have a large impact on the system *response time* and *throughput*. The *image processing algorithms* determine the resulting *image quality*.

The design space is not a simple multi-dimensional space, with orthogonal, independent dimensions. The image processing algorithm has impact on the CPU usage, cache efficiency, memory usage, and image quality. The implementation of these algorithms can be optimized to one or two of these entities, often at the cost of the remaining optimization criteria. For instance: images can be stored completely in memory, which is most efficient for CPU processing time. An alternative is to store and process small parts of the image (lines) at a time, which is more flexible with respect to memory (less fragmentation), but the additional indirection of addressing the image line costs CPU time.

Adding concurrency partially helps to improve response times. Waiting times, for instance for disk reads, can then be used to do other useful processing. On the other hand additional overhead in context switching, and locking is caused by the concurrency.

The essence of the thread of reasoning is to have sufficient insight in the customer and application needs, so that the problem space becomes sharply defined and understood. This understanding is used to select the *sweet* spots of the design space, that satisfy the needs. Understanding of the design space is needed to sharpen the understanding of the problem space; in other words iteration between problem and solution space is required.

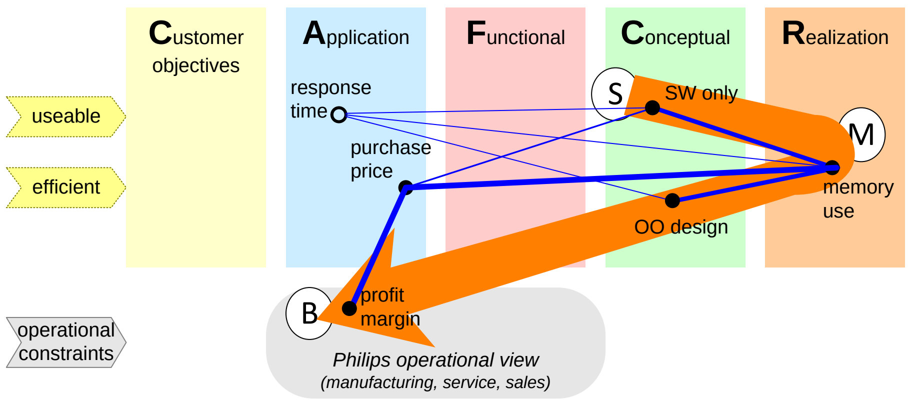
## 5.3 Exploration of Problems and Solutions

In this section the thread of reasoning is shown as it emerges over time. For every phase the CAFCR views are annotated with relevant subjects in that phase and the relations between the subjects.

Figures 5.2 to 5.6, described in Subsections 5.3.1 to 5.3.5, show the phases as described in Chapter **??**. The figures show the main issues under discussion as dots. The relations between the issues are shown as lines between the issues, where the thickness of the line indicates the relative weight of the relationship. The core of the reasoning is indicated as a thick arrow. The cluster of issues at the start point and at the finish are shown as letter in a white ellipse. Some clusters of issues at turning points in the reasoning are also indicated as white ellipse.

### 5.3.1 Phase 1: Introvert View

At the moment that the architect (me) joined the product development a lot of technology exploration had been transformed into a working prototype, the so-called *basic application*. Main ingredients were the use of Object-Oriented (OO) technology and the vision that a “software only” product was feasible en beneficial.



Introvert view: cost and impact of new technologies

Figure 5.2: Thread of reasoning; introvert phase. The starting point (S) is the a priori design choice for a SW only solution based on Object Orientation. The consequence for resource usage, especially memory (M) and the business (B), especially product margin are explored.

Experienced architects will address two major concerns immediately: will the design with these new technologies fit in the technical constraints, especially memory in this case, and will the product fit in the business constraints (do we make sufficient margin and profit)?

The response time has been touched only very lightly. The system was only capable of viewing, an activity for which response time is crucial. The prototype showed acceptable performance, so not much time was spent on this issue. Design changes to eventually solve cost or memory issues potentially lower the performance, in which case response time can suddenly become important.

Figure 5.2 shows the thread of reasoning in this early stage. Striking is the introvert nature of this reasoning: internal design choices and Philips internal needs dominate. The implicitly addressed qualities are useability and efficiency. Most attention was for the operational constraints. The direction of the reasoning during this phase has been from the Conceptual and Realization views towards the operational consequences: starting at the designers choice for OO and software only (S), via concerns over memory constraints (M) towards the business (B) constraints margin and profit. The figure indicates that more issues have been touched in the reasoning, such as response time from user point of view. In the day to day situation many more related issues have been touched, but these issues had less impact on the overall reasoning.

### 5.3.2 Phase 2: Exploring Memory Needs

The first phase indicated that the memory use was unknown and unpredictable. It was decided to extend the implementation with measurement provisions, such as memory usage. The OIT in the dynamic run time environment enabled a very elegant way of tracing object instantiations. At the same time a new concern popped up: what is the overhead cost induced by the run time environment?

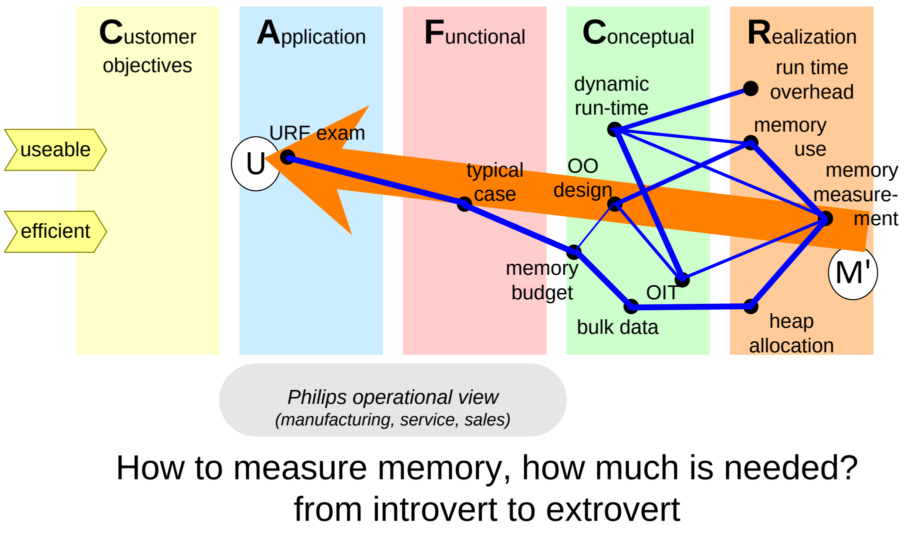


Figure 5.3: Thread of reasoning; memory needs. Create insight by zooming in on memory management (M'). Requirements for the memory management design are needed, resulting in an exploration of the typical URF examination (U).

The object instantiation tracing could easily be extended to show the amount of memory allocated for the object structures. The large data elements, such as images, are allocated on the heap and required additional instrumentation. Via the bulkdata concept this type of memory use was instrumented. Bottom up the insight in memory use was emerging.

The need arose to define relevant cases to be measured and to be used as the basis for a memory budget. An URF examination was used to define a typical case. Now the application knowledge starts to enter the reasoning process, and the reasoning starts to become more extrovert. Efficiency and usability are the main qualities addressed.

Figure 5.3 shows the thread of reasoning for Phase 2. The reasoning is still bottom-up from Realization towards Application View. The realization concerns about speed and memory consumption (M') result into concepts for resource management and measurement support. For analysis and validation a use case description in the

Functional view is needed. The use case is based on insight in a URF examination (U) from application viewpoint.

### 5.3.3 Phase 3: Extrovert View Uncovers Gaps in Conceptual and Realization Views

The discussion about the URF examination and the typical case made it very clear that radiologists perform their diagnoses by looking at the film on the lightbox. This is for them very efficient in time. Their speed of working is further increased by the autoloader, which quickly shows all films of the next examination.

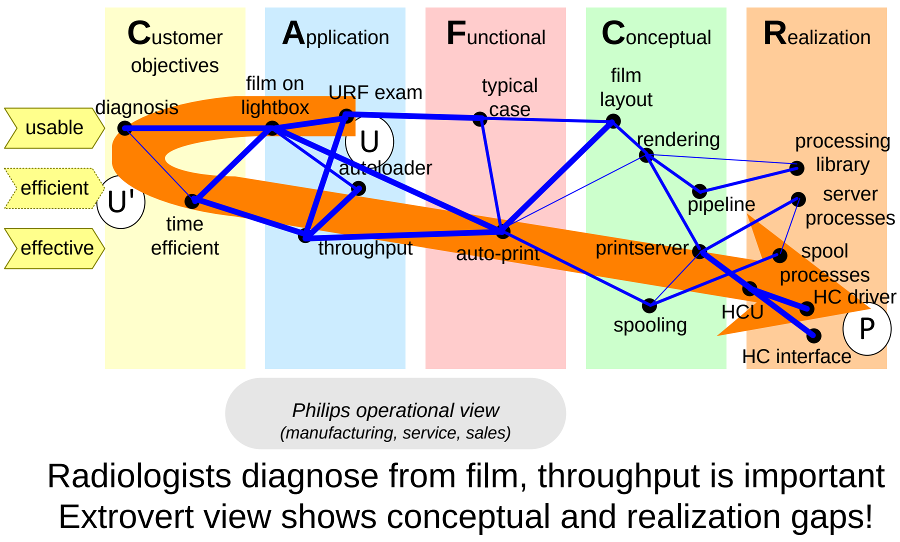


Figure 5.4: Thread of reasoning; uncovering gaps. The insight is deepened by further exploration of the URF examination (U) and the underlying objectives (U') of the radiologist. The auto-print functionality is specified as response for the radiologist needs. The technical consequences of the auto-print are explored, in this case the need for printing concepts and realization (P).

To support this typical workflow the production of filmsheets and the throughput of films and examinations is important. Interactive viewing on the other hand is from the radiologist's point of view much less efficient. Diagnosis on the basis of film takes seconds, diagnosis by interactive viewing takes minutes. The auto-print functionality enables the production of films directly from the examination room.

auto-print functionality requires lots of new functions and concepts in the system, such as background printing (spooling), defining and using film layouts, using the right rendering, et cetera. The processing library must support these functions. Also an execution architecture is required to support the concurrency: server processes and spool processes are introduced. Last but not least, hardcopy units (HCU), for example laser printers, need to be interfaced to the system. A new set of components is introduced in the system to do the printing: hardcopy interface hardware, hardcopy driver, and the hardcopy units themselves.

During this phase the focus shifted from efficiency to effectiveness. Efficiency is mostly an introvert concern about resource constraints. Effectiveness is a more extrovert concern about the quality of the result. Hitchins clearly explains in [4] efficiency and effectiveness, and points out that the focus on efficiency alone creates vulnerable and sub-optimal systems. Usability remains important during this phase, for example auto-print.

Figure 5.4 shows the thread of reasoning of Phase 3. The insights obtained during the previous phase trigger a further exploration of the Customer Objectives and Application View. The insight that an efficient diagnosis (U') is performed by means of film sheets on a lightbox (U) triggers the addition of the auto-print function to the Functional View. New concepts and software functions are needed to realize the auto-print function (P). The direction of reasoning is now top-down over all the CAFCR views.

### 5.3.4 Phase 4: from Diagnosis to Throughput

The discussion about URF examinations and the diagnostic process triggers another thread, a thread about the desired diagnostic quality. The high brightness and resolution of films on a lightbox ensures that the actual viewing is not degrading the diagnostic quality. The inherent image quality of the acquired and printed image is critical for the final diagnostic quality.

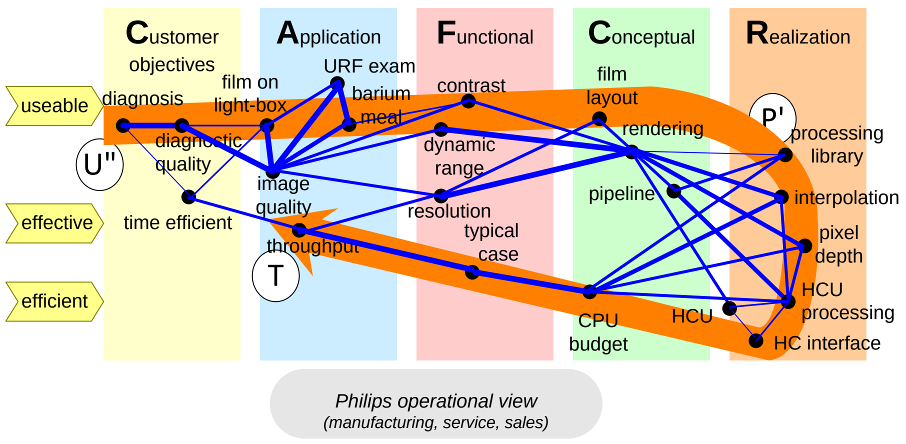
At specification level the image quality is specified in terms of resolution, contrast and dynamic range. At application level the contrast is increased by the use of barium meal, which takes the contrast to the required level in these soft (for X-ray low contrast) tissues. At the same time the combination of X-ray settings and barium meals increases the dynamic range of the produced images.

The size of the images depends on the required resolution, which also determines the film layout. The rendering algorithms must fulfil the image quality specifications. The rendering is implemented as a pipeline of processing steps from an optimized processing library.

One of the costly operations is the interpolation. One of the design options was to use the processing in the hardcopy unit. This would greatly relieve the resource (processor and memory) needs, but it would at the same time be much less flexible with respect to rendering. It was decided not to use the hardcopy unit processing.

A CPU budget was created, based on the typical case and taking into account all previous design know-how. This CPU budget did fit in the required throughput needs.

Usability, effectiveness and efficiency are more or less balanced at this moment.



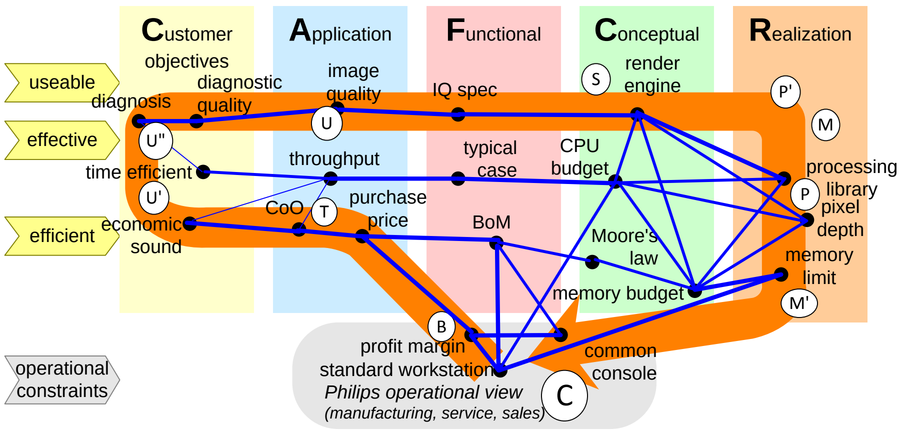
from extrovert diagnostic quality, via image quality, algorithms and load, to extrovert throughput

Figure 5.5: Thread of reasoning; phase 4. The insight is broadened. Starting at the objective to perform *diagnosis* efficient in time (U”), the application is further explored: type of examination and type of images. The specification of the imaging needs (contrast, dynamic range, resolution) is improved. The consequences for rendering and film layout on a large set of realization aspects (P’) is elaborated. The rendering implementation has impact on CPU usage and the throughput (T) of the typical case.

Figure 5.5 shows the thread of reasoning for Phase 4. During this phase the reasoning iterates over all the CAFCR views. The diagnostic quality (U”) in the Customer Objectives View results via the clinical acquisition methods in the Application view in image quality requirements in the Functional View. The layout and rendering in the Conceptual view result in a large set of processing functions (P’) in the Realization view. The specific know how of the processing in the Realization is used for the CPU and memory budgets in the conceptual view, to validate the feasibility of supporting the typical case in the Functional view. The typical case is a translation of the throughput (T) needs in the Application View.

### 5.3.5 Phase 5: Cost Revisited

At this moment much more information was available about the relation between resource needs and system performance. The business policy was to use standard of-the-shelf workstations. The purchase price by the customer could only be met by using the lowest cost version of the workstation. Another policy was to use a Philips medical console, which was to be common among all products. This console was about half of the material cost of the Medical Imaging workstation.



cost revisited in context of clinical needs and realization constraints; note: original threads are significantly simplified

Figure 5.6: Thread of reasoning; cost revisited. The entire scope of the thread of reasoning is now visible. Sufficient insight is obtained to return to the original business concern of margin and cost (C). In the mean time additional assumptions have surfaced: a common console and standard workstation to reduce costs. From this starting point all other viewpoints are revisited: via time efficient diagnosis to image quality to rendering and processing and back to the memory design.

The real customer interest is to have a system that is economically sound, and where throughput and cost of ownership (CoO) are balanced. Of course the main clinical function, diagnosis, must not suffer from cost optimizations. A detailed and deep understanding of the image quality needs of the customer is needed to make an optimized design.

Note that at this moment in time many of the considerations discussed in the previous steps are still valid and present. However Figure 5.6 is simplified by leaving out many of these considerations.

Besides efficiency, effectiveness, and usability, the operational constraint is back in the main reasoning thread. At this moment in time that makes a lot of sense, because problem and solution space are sufficiently understood. These constraints never disappeared completely, but the other qualities were more dominant in the intermediate phases.

Figure 5.6 shows the thread of reasoning in Phase 5. The original business viewpoint is revisited: do we have a commercial feasible product? A full iteration over all CAFCR views relates product costs (C) to the key drivers in the Customer Objectives. The main tensions in the product specification are balanced: image quality, throughput of the typical case and product cost. To do this balancing the main design choices in the Conceptual and Realization views have to be reviewed.

## 5.4 Conclusion

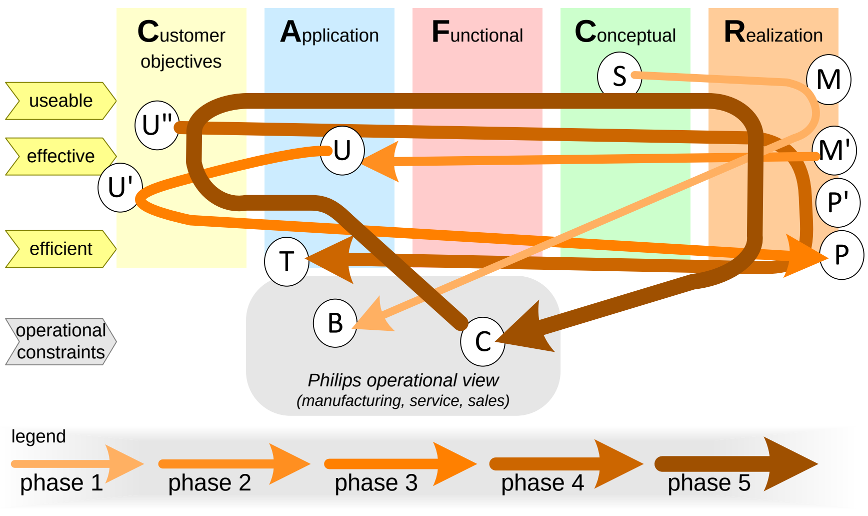


Figure 5.7: All steps superimposed in one diagram. The iterative nature of the reasoning is visible: the same aspects are explored multiple times, coming from different directions. It also shows that jumps are made during the reasoning.

The know-how at the start of the product creation was limited to a number of nice technology options and a number of business and application opportunities. The designers had the technology know-how, the marketing and application managers had the customer know-how. The product creation team went through several learning phases. Figure 5.7 shows the many iterations in the five phases. During those phases some of the know-how was shared and a lot of new know-how emerged. The sharing of know-how made it possible to relate customer needs to design and implementation options. The interaction between the team members and the search for relations between needs and designs triggered many new questions. The answers to these questions created new know-how.

The architecting process has been analyzed in retrospect, resulting in this description of *threads of reasoning*. This Chapter *Threads of Reasoning* shows that:

- The specification and design issues that are discussed fit in all CAFCR views or the operational view.

- The positioning of the issues and their relationships in the CAFCR views enable a compact description of the reasoning during the product creation.
- Submethods are used to address one issue or a small cluster of issues.
- Qualities are useful as integrating elements over the CAFCR views.
- The *threads of reasoning* are an explicit way to facilitate the interaction and the search for relations.
- The *threads of reasoning* create an integral overview.
- The *threads of reasoning* facilitate a converging specification and design.

# Bibliography

[1] Gerardo Daalderop, Ann Ouvry, Luc Koch, Peter Jaspers, Jürgen Müller, and Gerrit Muller. PACS assessment final report, version 1.0. confidential internal report XLB050-96037, September 1996.

[2] Remco M. Dijkman, Luï¿½s Ferreira Pires, and Stef M.M. Joosten. Calculating with concepts: a technique for the development of business process support. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, *Lecture Notes in Informatics*, volume 7, pages 87–98. GI-edition, 2001. http://www.google.com/url?sa=U&start=3&q=http://www.ub.utwente.nl/webdocs/ctit/1/00000068.pdf&e=7764 Proceedings of the UML 2001 Workshop on Practical UML-Based Rigorous Development Methods.

[3] EventHelix.com. Publish-subscribe design patterns. http://www.eventhelix.com/RealtimeMantra/Patterns/publish_subscribe_patterns.htm, 2000.

[4] Derek K. Hitchins. Putting systems to work. http://www.hitchins.co.uk/, 1992. Originally published by John Wiley and Sons, Chichester, UK, in 1992.

[5] Gerrit Muller. The system architecture homepage. http://www.gaudisite.nl/index.html, 1999.

**History**
**Version: 0, date: July 9, 2004 changed by: Gerrit Muller**
- created module