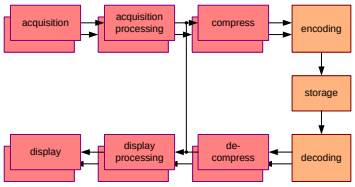
The conceptual view

-



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

Abstract

The purpose of the conceptual view is described. A number of methods or models is given to use in this view: construction decomposition, functional decomposition, class or object decomposition, other decompositions (power, resources, recycling, maintenance, project management, cost, ...), and related models (performance, behavior, cost, ...); allocation, dependency structure; identify the infrastructure (factoring out shareable implementations), classify the technology in *core*, *key* and *base* technology; integrating concepts (start up, shutdown, safety, exception handling, persistency, resource management,...).

version: 0.7 status: preliminary draft June 5, 2018

1 Introduction

The conceptual view is used to understand how the product is achieving the specification. The methods and models used in the conceptual view should discuss the *how* of the product in conceptual terms. The lifetime of the concepts is longer than the specific implementation described in the *Realization* view. The conceptual view is more stable and reusable than the *realization* view.

The dominant principle in design is decomposition, often immediately coupled to interface management of the interfaces of the resulting components. It is important to realize that any system can be decomposed in many relevant ways. The most common ones are discussed here briefly: construction decomposition, section 2, functional decomposition, section 3, class or object decomposition, other decompositions (power, resources, recycling, maintenance, project management, cost, execution architecture...), and related models (performance, behavior, cost, ...).

If multiple decompositions are used then the relationships between decompositions are important. One of the methods to work with these relationships is via allocation. Within a decomposition and between decompositions the dependency structure is important.

From development management point of view it is useful to identify the infrastructure (factoring out shareable implementations), and to classify the technology in *core*, *key* and *base* technology.

The complement of decomposition is integration. Articulating the integrating concepts (start up, shutdown, safety, exception handling, persistency, resource management,...) provides guidance to the developers and helps to get a consistently behaving system.

2 Construction decomposition

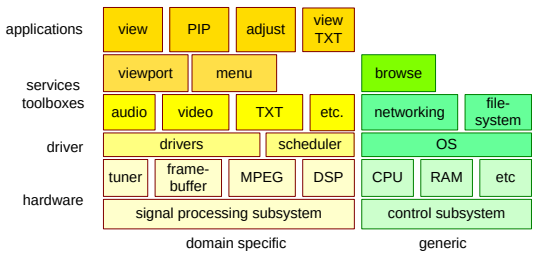


Figure 1: Example of a construction decomposition of a simple TV

The construction decomposition views the system from the construction point of view, see figure 1 for an example and figure 2 for the characterization of the construction decomposition.

The construction decomposition is mostly used for the design management. It defines units of design, as these are created and stored in repositories and later updated. The atomic units are aggregated in compound design units, which are used as unit for testing and release and this often coincides with organizational ownership and responsibility.

management of design	SW example	HW example
unit of creation storage update	file	PCB IP cells IP core
unit of aggregation for organisation test release	package module	box IP core IC

Figure 2: Characterization of the construction decomposition

In hardware this is quite often a very natural decomposition, for instance in cabinets, racks, boards and finally IC's, IP cores and cells. The components in the hardware components are very tangible. The relationship with a number of other decompositions is reasonably one to one, for instance with the work breakdown for project management purposes.

The construction decomposition in software is more ambiguous. The structure of the code repository and the supporting build environment comes close to the hardware equivalent. Here files and packages are the aggregating construction levels. This decomposition is less tangible than the hardware decomposition and the relationship with other decompositions is sometimes more complex.

3 Functional decomposition

The functions as described in the functional view have to be performed by the design. These functions often are an aggregation of more elementary functions in the design. The functional decomposition decomposes end user functions in more elementary functions.

Be aware of the fact that the word *function* in system design is heavily overloaded. It does not help to define sharp boundaries with respect to the functional decomposition. Main criterium for a good functional decomposition is its useability for design. A functional decomposition provides insight how the system will accomplish its job.

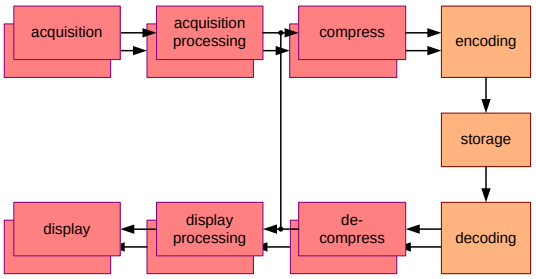


Figure 3: Example functional decomposition camera type device

Figure 3 shows an example of (part of) a functional decomposition for a camera type device. It shows communication, processing and storage functions and their relations. This functional decomposition is **not** addressing the control aspects, which might be designed by means of a second functional decomposition, but from control point of view.

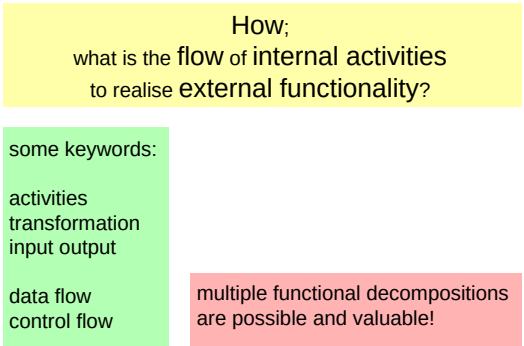


Figure 4: Characterization of the functional decomposition

4 Designing with multiple decompositions

The design of complex systems always requires multiple decompositions, for instance a construction and a functional decomposition. Many designers in the design team need support to cope with this multiplicity.

Most designers don't anticipate cross system design issues, for instance when asked in preparation of design team meetings. This limited anticipation is caused by the locality of the viewpoint, implicitly chosen by the designers.

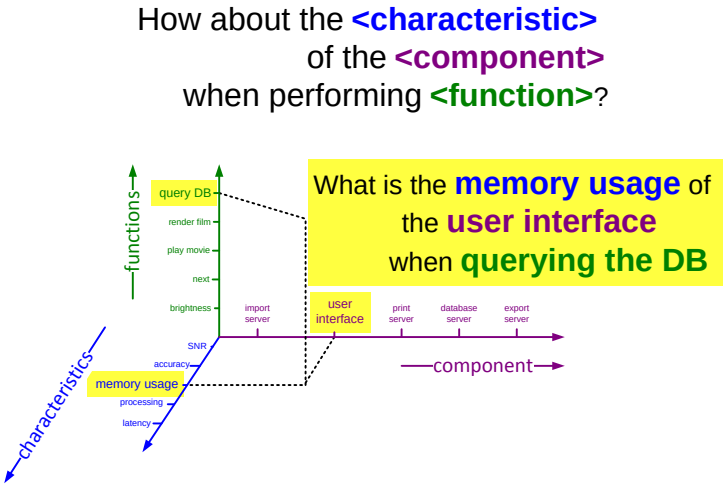


Figure 5: Question generator for multiple decompositions

Figure 5 shows a method to help designers to find system design issues. A three dimensional space is shown. Two dimensions are the decomposition dimension (component and functional), the last dimension is the design characteristic dimension.

For every point in this 3D space a question can be generated in the following way:
How about the <*characteristic*> of the <*component*> when performing <*function*>?
Which will result in questions like:
How about the *memory usage* of the *user interface* when *querying the database*?

The designers will not be able to answer most of these questions. Simply asking these questions helps the designer to change the viewpoint and discover many potential issues. Luckily most of the not answered questions will not be relevant. The answer to the memory usage question above might be *insignificant* or *small*.

The architect has to apply a priori know how to select the most relevant questions in the 3D space. Figure 6 shows a set of selection factors that can be used to determine the most relevant questions.

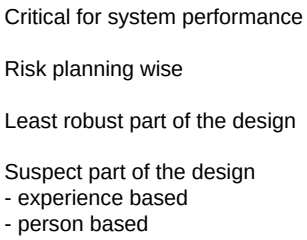


Figure 6: Selection factors to improve the question generator

Critical for system performance Every question that is directly related to critical aspects of the system performance is relevant. For example *What is the CPU load of the motion compensation function in the streaming subsystem?* will be relevant for resource constrained systems.

Risk planning wise Questions regarding critical planning issues are also relevant. For example *Will all concurrent streaming operations fit within the designed resources?* will greatly influence the planning if resources have to be added.

Least robust part of the design Some parts of the design are known to be rather sensitive, for instance the priority settings of threads. Satisfactory answers should be available, where a satisfactory answer might also be *we scheduled a priority tuning phase, with the following approach.*

Suspect part of the design Other parts of the design might be suspect for several reasons. For instance experience learns that response times and throughput do not get the required attention of software designers (experience based suspicion). Or for instance we allocated an engineer to the job with insufficient competence (person based suspicion).

Figure 7 shows another potential optimization, to address a line or a plane in the multi dimensional space. The figure shows an example of a memory budget for the system, which is addressing all memory aspects for both functions and components in one budget. The other example is the design specification of a database query, where the design addresses the allocation to components as well as all relevant design characteristics.

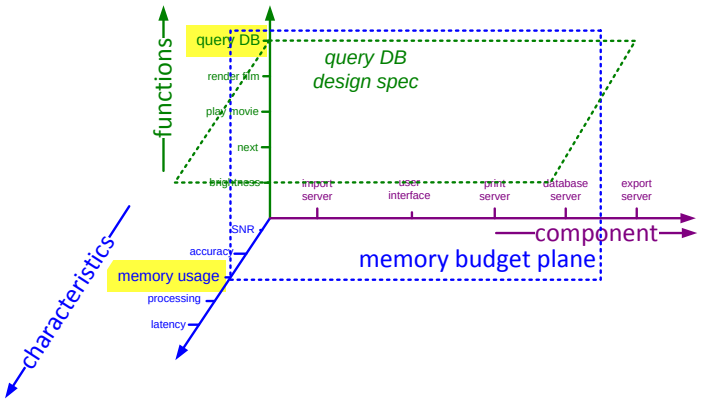


Figure 7: Addressing lines or planes at once in the multiple dimensions

5 Internal Information Model

The information model as seen from the outside from the system, part of the functional view, is extended into an internal information model. The internal information model is extended with design choices, for instance derived data information is cached to achieve the desired performance. The internal data model might also be chosen to be more generic (for reasons of future extendibility), or less generic (where program code is used to translate the specific internal models in the desired external models.

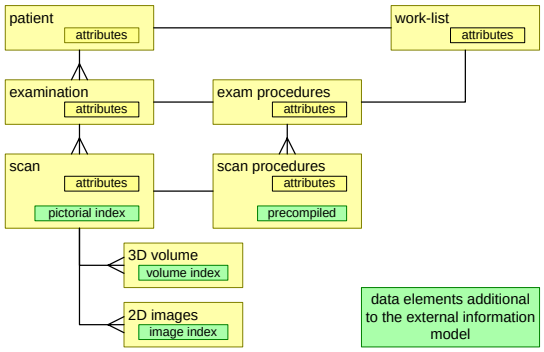


Figure 8: Example of a partial internal information model

The internal information model is an important means to decouple parts of the design. The functional behavior of the system is predictable as long as components in the system adhere to the internal information model.

Figure 8 shows an example of a part of an information model. In this example several information elements which are derived from the primary data are stored explicitly to improve the response time. The pictorial index, existing of reduced size images, is an example of derived information, which takes some time to calculate. This index is build in the background during import, so that the navigation can use it, which makes the navigation very responsive.

All considerations described in section **??**, such as the layering hold also for the internal information model.

6 Execution architecture

The execution architecture is the run time architecture of a system. The process decomposition plays an important role in the execution architecture. Figure 9 shows an example of a process decomposition.

One of the main concerns for process decomposition is concurrency: which concurrent activities are needed or running, how to synchronize these activities. A

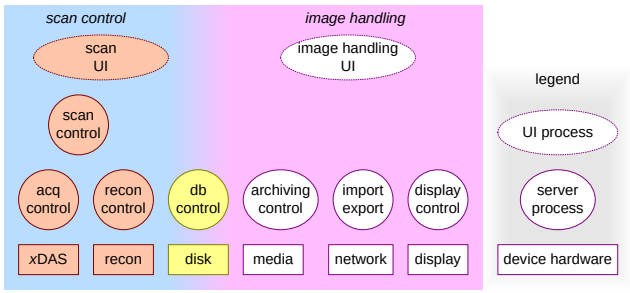


Figure 9: Example process decomposition

process or a task of an operating system is a concept which supports asynchronous functionality as well as separation of concerns by providing process specific resources, such as memory. A thread is a lighter construction providing support for asynchronous activities, without the separation of concerns.

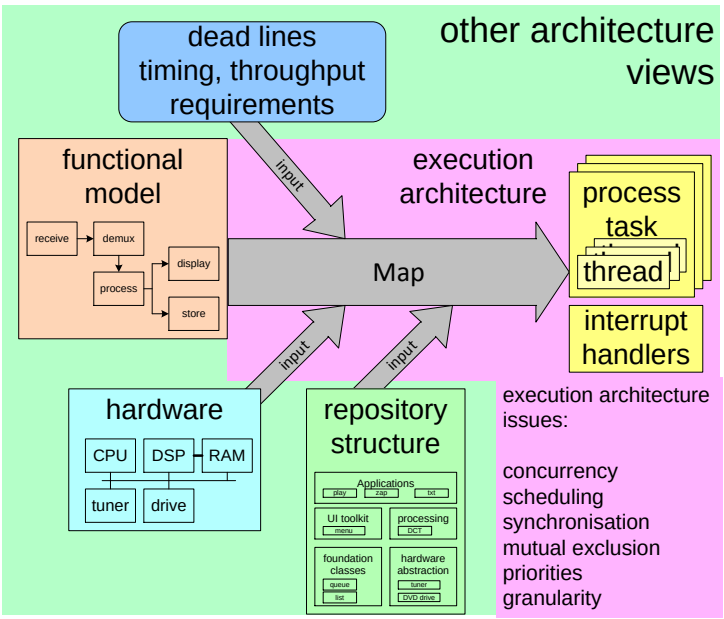


Figure 10: Execution architecture

The execution architecture must map the functional decomposition on the process decomposition, taking into account the construction decomposition. In practice many building blocks from the construction decomposition are used in multiple functions mapped on multiple processes. These shared building blocks are aggregated in shared (or dynamic link) libraries. Sharing is advantageous from memory

consumption point of view, some attention is required for the configuration management side[1].

Figure 10 shows the role of the execution architecture. The main inputs are the real time and performance requirements at the one hand and the hardware design at the other hand. The functions need to be mapped on processes, threads and interrupt handlers, synchronization method and granularity need to be defined and the scheduling behavior (for instance priority based, which requires priorities to be defined).

[1]The *dll-hell* is not an windows-only problem. Multiple pieces of software sharing the same library can easily lead to version problems, module 1 requires version 1.13, while module 2 requires version 2.11. Despite all compatibility claims it often does not work.

7 Performance

The performance of a system can be modeled by complementing models. In figure 11 the performance is modelled by a flow model at the top and an analytical model below. The analytical model is entirely parameterized, making it a generic model which describes the performance ratio over the full potential range.

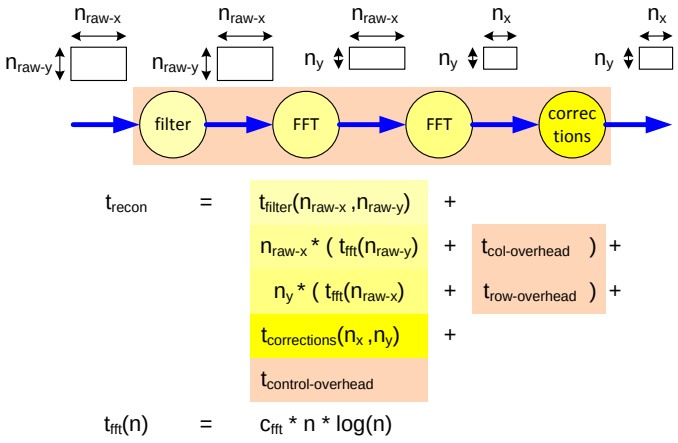


Figure 11: Performance Model

Later in the realization view it will be shown that this model is too simplistic, because it focuses too much on the processing and does not take the overheads sufficiently in account.

8 Safety, Reliability and Security concepts

The qualities *safety*, *reliability* and *security* share a number of concepts, such as:

- containment (limit failure consequences to well defined scope)
- graceful degradation (system parts not affected by failure continue operation)
- dead man switch (human activity required for operation)
- interlock (operation only if hardware conditions are fulfilled)
- detection and tracing of failures
- black box (log) for post mortem analysis
- redundancy

A common guideline in applying any of these concepts is that the more critical a function is, the higher the understandability should be, or in other words the simpler the applied concepts should be. Many elementary safety functions are implemented in hardware, avoiding large stacks of complex software.

9 Start up and shutdown

In practice insufficient attention is paid to the start up and shutdown of a system, since these are relatively exceptional operations. However the design of this aspect has an impact on nearly all components and functions in the system. It is really an integrating concept. The trend is that these operations become even more entangled with the normal run-time functionality, for instance by run-time downloading, stand-by and other power saving functionality.

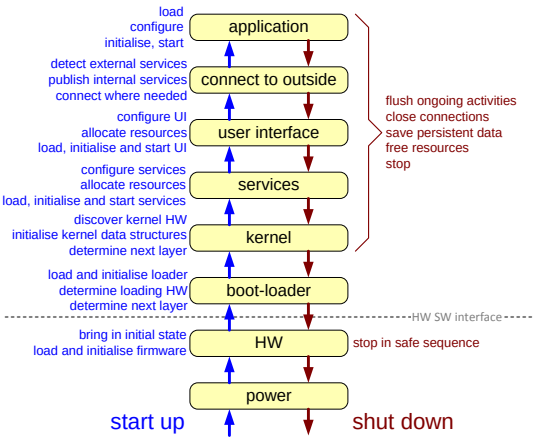


Figure 12: Simplified start up sequence

Figure 12 shows a typical start up shutdown pattern. The system is brought step by step to higher operational levels. Higher levels benefit from more available support functions, lower levels are less dependent on support functions.

One of the considerations in the design of this system aspect is the impact of failures. The right granularity of operational levels enable coping with exceptions (for example network not available). For shutdown the main question is how power failures or glitches are handled.

10 Work breakdown

Project leaders expect a work breakdown to be made by the architect. In fact a work breakdown is again another decomposition, with a more organizational point of view. The work in the different work packages should be cohesive internally, and should have low coupling with other work-packages.

Figure 13 shows an example of a work breakdown. The entire project is broken down in a hierarchical fashion: project, segment, work-package. In this example color coding is applied to show the technology involved and to show development

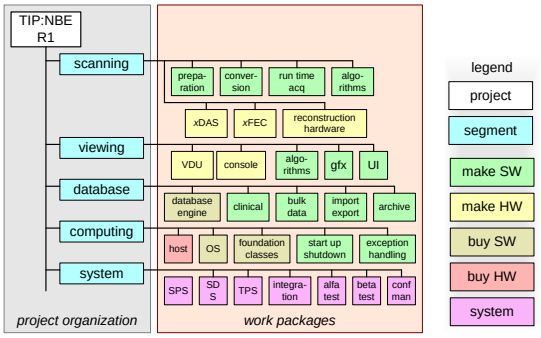


Figure 13: Example work breakdown

work or purchasing work. Both types of work require domain know how, but different skills to do the job.

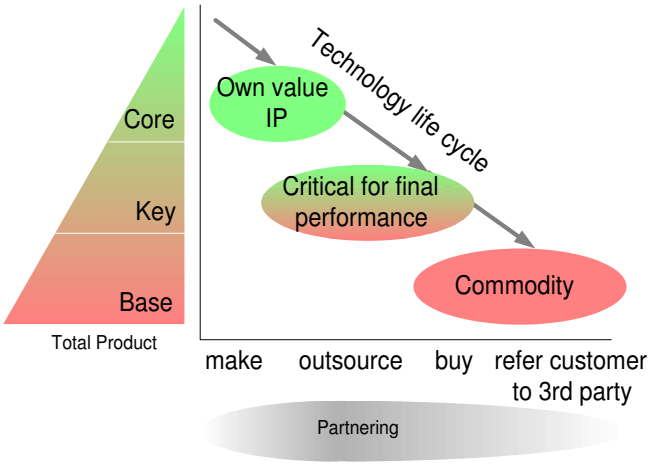


Figure 14: Core, Key or Base technology

Make versus Buy is a limited subset of an entire spectrum of approaches. The decision how to obtain the needed technology should be based on where the company intents to add value. A simple reference model to help in making these decisions is based on *core*, *key*, and *base* technology, see figure 14.

Core technology is technology where the company is adding value. In order to be able to add value, this technology should be developed by the company itself.

Key technology is technology which is critical for the final system performance. If the system performance can not be reached by means of third party technology

than the company must develop it themselves. Otherwise outsourcing or buying is attractive, in order to focus as much as possible on *core* technology added value. However when outsourcing or buying an intimate partnership is recommended to ensure the proper performance level.

Base technology is technology which is available on the market and where the development is driven by other systems or applications. Care should be taken that these external developments can be followed. Own developments here are de-focusing the attention from the company's core technology.

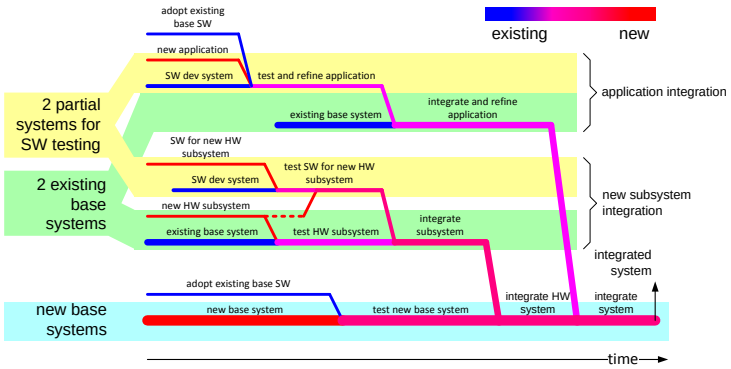


Figure 15: Example integration plan, with 3 tiers of development models

Schedules, work breakdown and many technical decompositions are heavily influenced by the integration plan. Integration is time, effort and risk determining part of the entire product creation process. The integration viewpoint must be used regular because of its time, effort and risk impact.

Figure15 shows an example integration plan. This plan is centered around 3 tiers of development vehicles:

- SW development systems
- existing HW system
- new HW system

The SW development system, existing from standard clients and servers, is very flexible and accessible from software point of view, but far from realistic from hardware point of view. The existing and new HW systems are much less accessible and more rigid, but close to the final product reality. The new HW system will be available late and hides many risks and uncertainties. The overall strategy is to move for software development from an accessible system to a stable HW system to the more real final system. In general integration plans try to avoid stacking

too many uncertainties by looking for ways to test new modules in a stable known environment, before confronting new modules with each other.

11 Acknowledgements

Constructive remarks from Peter Bingley, Peter van den Hamer, Ton Kostelijk, William van der Sterren and Berry van der Wijst have been integrated in this document.

References

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

History

Version: 0.7, date: September 18, 2003 changed by: Gerrit Muller

- corrected copy/paste artefact in ”Introduction”

Version: 0.6, date: September 9, 2003 changed by: Gerrit Muller

- updated text of Section ”Start up and shutdown” about the frequency of occurance

Version: 0.5, date: May 6, 2003 changed by: Gerrit Muller

- added text to section ”internal information model”
- added text to section ”Start up and shutdown”
- changed status to preliminary draft

Version: 0.4, date: October 3, 2002 changed by: Gerrit Muller

- added integration plan including descriptive text

Version: 0.3, date: October 1, 2002 changed by: Gerrit Muller

- added text to construction decomposition
- added figure construction decomposition characterization
- added figure functional decomposition characterization
- added information model
- section acknowledgements added

Version: 0.2, date: September 3, 2002 changed by: Gerrit Muller

- added section Safety, reliability and security concepts
- added simplified start up diagram
- added work breakdown and text
- added core key base diagram and text
- added performance model
- added execution architecture diagram and text
- added example process decomposition

Version: 0.1, date: July 9 2002 changed by: Gerrit Muller

- updated figure Functional decomposition

Version: 0, date: April 2. 2002 changed by: Gerrit Muller

- Created, no changelog yet