Module Modeling and Analysis: System model

Gerrit Muller
HSN-NISE
Frogs vei 41 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

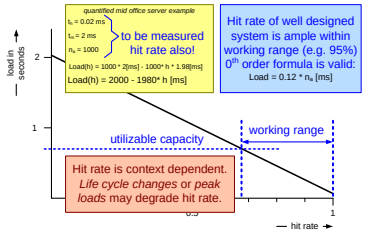
This module addresses Modeling and Analysis Performance. What are the customer performance needs, what are the operational performance considerations? What are the performance related design choices? How to analyze feasibility, explore design options, and how to validate performance?

version: 0.4 status: preliminary draft July 4, 2016

Contents

Chapter 1

Modeling and Analysis: System Model



1.1 Introduction

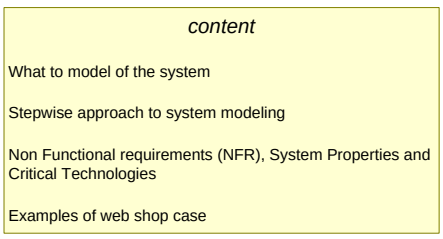


Figure 1.1: Overview of the content of this paper

Figure 1.1 shows an overview of this paper. We will discuss what to model of the system of interest, see also Figure 1.2. We will provide a stepwise approach to system modeling, based on the relations between Non Functional Requirements (NFR), system design properties and critical technologies. Several examples will be shown using the web shop case.

In our modeling we will focus on the NFR's, such as performance, reliability,

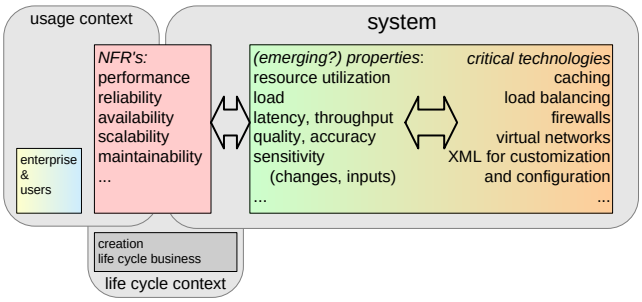


Figure 1.2: What to Model in System Context?

availability, scalability, or maintainability. We assume that the functional requirements and system decomposition are being created in the system architecting and design activity. In practice many NFR's emerge, due to lack of time and attention. We recommend to reduce the risks by modeling and analysis of relevant NFR's where some higher risk is perceived. Figure 1.2 shows that the external visible system characteristics depend on the design of system properties, such as resource utilization, load, latency, throughput, quality, accuracy, or sensitivity for changes or varying inputs. Note that these properties also often emerge, due to lack of time or attention. Not only the design of these properties determine the external visible system characteristics, but also the chosen technologies has a big impact. Therefor we also have to look at critical technologies, for example caching, load balancing, firewalls, virtual networks, or XML for customization and configuration.

1.2 Stepwise approach to system modeling

We recommend an approach where first the system is explored: what is relevant, what is critical? Then the most critical issues are modeled. Figure 1.3 shows a stepwise approach to model a system.

1. **Determine relevant Non Functional Requirements (NFR's)** where the relevance is often determined by the context: the usage context or the life cycle context.

2. **Determine relevant system design properties** by looking either at the NFR's or at the design itself: what are the biggest design concerns?

3. **Determine critical technologies** criticality can have many reasons, such as working close to the working range limit, new components, complex functions with unknown characteristics, sensitivity for changes or environmental conditions, et cetera.

4. **relate NFR's to properties to critical technologies** by making a graph of relations. Such a graph often has many-to-many relations.

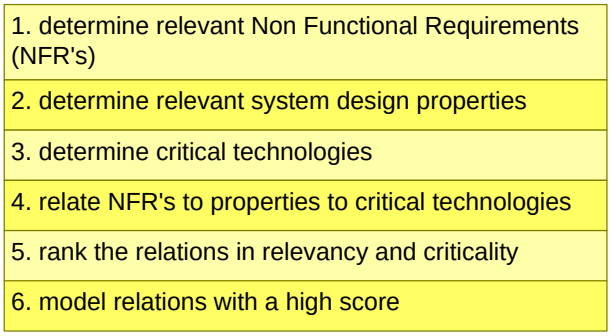


Figure 1.3: Approach to System Modeling

5. Rank the relations in relevancy and criticality to find potential modeling candidates.

6. Model relations with a high ranking score a time-boxed activity to build up system understanding.

Note that this system modeling approach fits in the broader approach of modeling and analysis. The broader approach is discussed in Modeling and Analysis: Reasoning.

1.3 Example system modeling of web shop

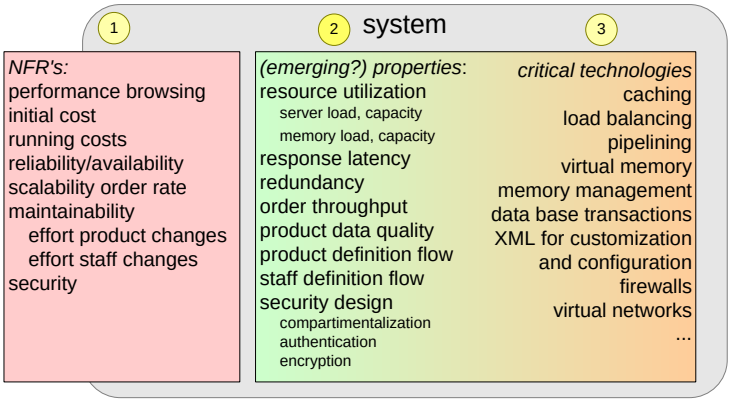


Figure 1.4: Web Shop: NFR's, Properties and Critical Technologies

Figure 1.4 shows the results of step 1, 2, and 3 of the approach.

1. Determine relevant Non Functional Requirements (NFR's) For the web shop the following requirements are crucial: performance browsing, initial cost,

running costs, reliability/availability, scalability order rate, maintainability (effort to enter product changes and effort to enter staff changes) and security.

2. **Determine relevant system design properties** based on experience and NFR's the following properties were identified as relevant: resource utilization (server load, server capacity, memory load, and memory capacity), response latency, redundancy, order throughput, product data quality, product definition flow, staff definition flow, security design (which can be refined further in compartimentalization, authentication, and encryption). Note that we mention here design issues such as *product definition flow* and *staff definition flow*, which have an direct equivalent in the usage context captured in some of the customer's processes.

3. **Determine critical technologies** Based on experience and risk assessment the following technologies pop up as potentially being critical: caching, load balancing, pipelining, virtual memory, memory management, data base transactions, XML for customization and configuration, firewalls, and virtual networks.

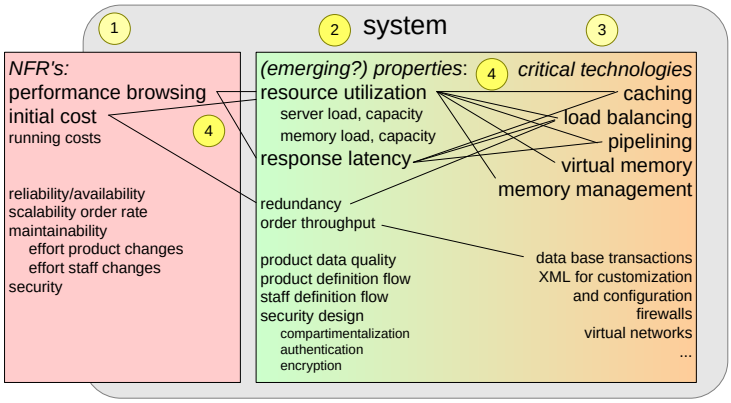


Figure 1.5: 4. Determine Relations

Figure 1.5 shows for a small subset of the identified requirements, properties and technologies the relations. The performance of browsing is related to the resource management design and the concurrency design to meet the response latency. The resource management design relates to several resource specific technologies from caching to memory management. The cost requirements also relate to the resource utilization and to the cost of redundancy measures. The dimensioning of the system depends also on the design of the order throughput. Crucial technology for the order throughput is the data base transaction mechanism and the related performance.

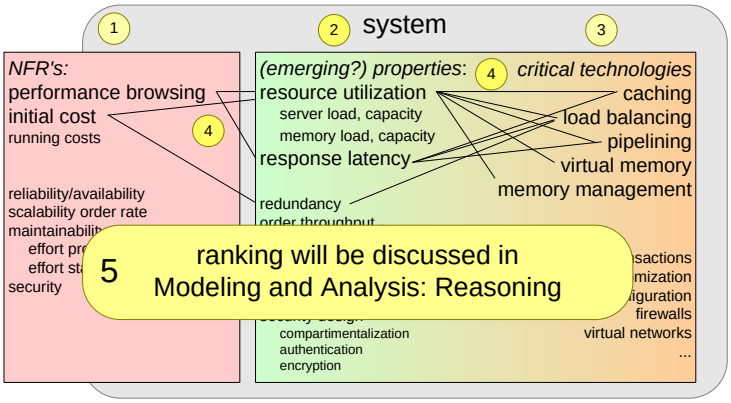


Figure 1.6: 5. Rank Relations

Ranking, Figure 1.6 will be discussed in the *Modeling and Analysis: Reasoning* paper. For this example we will mostly focus on the relations shown in Figure 1.5.

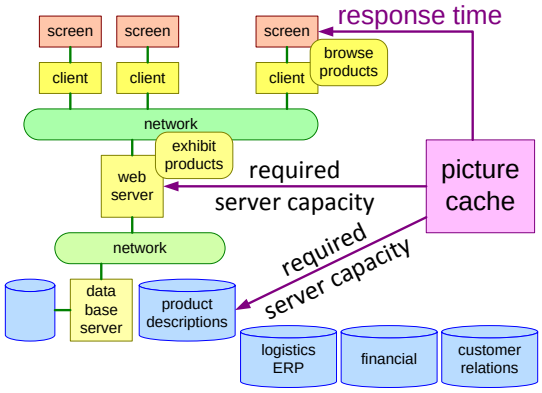


Figure 1.7: Purpose of Picture Cache Model in Web Shop Context

Figure 1.7 shows the picture cache as a specific example of the use of caching technology. The purpose of picture cache is to realize the required performance of product browsing at the client layer. At the web server layer and at the data base layer the picture cache should realize a limited server load of the product exhibition function.

The most simple model we can make for the server load as function of the number of requests is shown in Figure 1.8. This is a so called zero order model, where only the direct parameters are included in the model. It is based on the very simple assumption that the load is proportional with the number of requests.

When we introduce a cache based design, then the server load depends on the

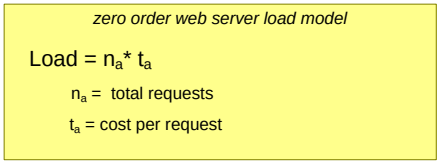


Figure 1.8: Zero Order Load Model

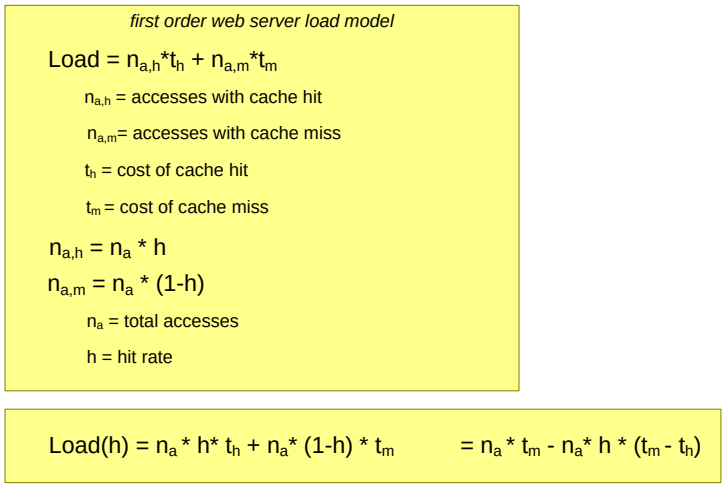


Figure 1.9: First Order Load Model

effectiveness of the cache. Requests that can be served from the cache will have a much smaller server load than requests that have to be fetched from the data base. Figure 1.9 shows a simple first order formula, where the contributions of requests from cache are separated of the requests that need data base access. We introduce an additional parameter h, the hit-rate of the cache. This helps us to create a simple formula where the server load is expressed as a function of the hit-rate.

The simple mathematical formula starts to make sense when we instantiate the formula with actual values. An example of such an instantiation is given in Figure 1.10. In this example we use values for request handling of $t_h = 20\mu s$ and $t_m = 2ms$. For the number of requests we have used $n_a = 1000$, based on the assumption that we are serving the product exhibition function with millions of customers browsing our extensive catalogue. The figure shows the server load as function of the hit-rate. If the available server capacity is known, then we can deduce the minimal required hit-rate to stay within the server capacity. The allowed range of hit-rate values is called the working range.

In Figure 1.11 we zoom in on the hit-rate model. First of all we should realize that we have used assumed values for t_h, t_m and n_a. These assumptions were

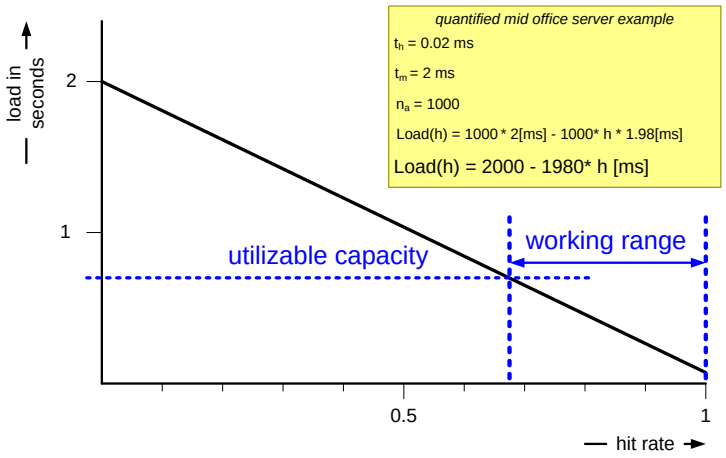


Figure 1.10: Quantification: From Formulas to Insight

based on experience, since we know as experienced designers that transactions cost approximately 1 ms, and that the cache roughly improves the request with a factor 100. However, the credibility of the model increases significantly by measuring these quantities. Common design practice is to design a system well within the working range, for example with a hit-rate of 95% or higher. If the system operates at these high hit-rates, then we can use the zero-order formula for the system load again. Using the same numbers for performance we should then use $t_a \approx 0.95 * t_h + 0.05 * t_m \approx 0.12ms$. Another assumption we have made is that the hit-rate is constant and independent of the circumstances. In practice this is not true. We will, for instance, have varying request rates, perhaps with some high peak values. If the peak values coincide with lower hit rates, then we might expect some nasty performance problems. The hit-rate might also be impacted by future life cycle changes. For example new and different browser functionality might decrease the hit-rate dramatically.

Another system property that was characterized as relevant was the response time design. Response time depends on the degree of concurrency, the synchronization design and the time required for individual operations. Figure 1.12 shows a timing model for the response time, visualizing the above mentioned aspects for the retrieval of a picture in case of a cache miss.

Yet another system design property is the use of resources, such as memory. Figure 1.13 shows the flow of pictures throughout the system, as a first step to address the question how much memory is needed for picture transfers.

In Figure 1.14 we zoom in on the web server to have a look at the memory used for picture transfers. This figure shows a number of alternative design options, ranging from a minimal set of copies in the memory to the more realistic situation

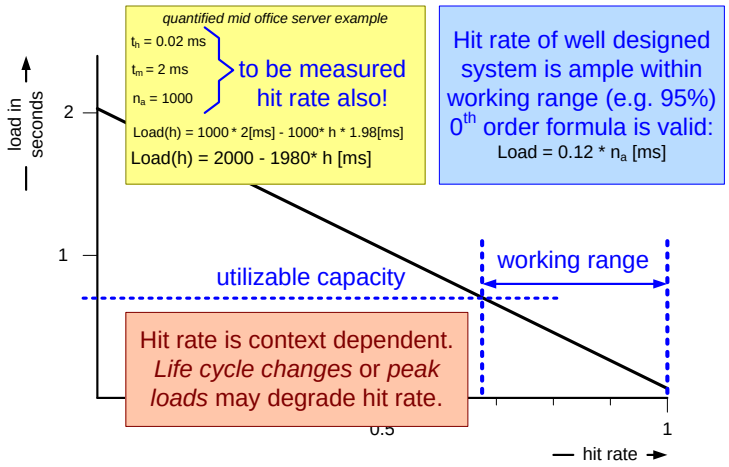


Figure 1.11: Hit Rate Considerations

where every thread contains multiple copies of every picture, while at the same time multiple threads serve concurrent customers.

These alternative design options are transformed in a simple mathematical model in Figure 1.15. This formula is parametrized for the different specification and design parameters:

n = number of data base access threads a design parameter dimensioning the amount of concurrency towards the data base layer.

m = number of picture cache threads a design parameter dimensioning the amount of concurrency of the picture cache itself.

k = number of web server threads a design parameter dimensioning the amount of concurrency of client access to the web server.

s = picture size in bytes an application and design dependent parameter, the average size in bytes of the pictures.

c = in memory cache capacity in number of pictures a design parameter determining the size of a picture cache in number of pictures per picture cache thread.

This formula is instantiated in a quantified table in Figure 1.16, for different values of the design parameters. Note that depending on the chosen design parameters the picture cache maps on completely different storage technologies, with the related different performance characteristics.

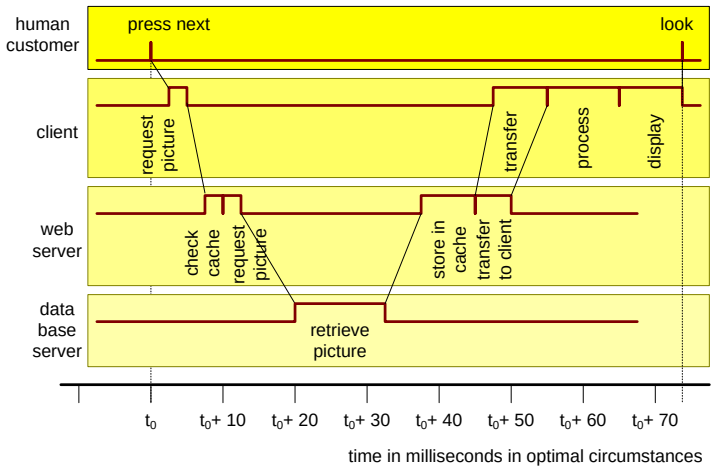


Figure 1.12: Response Time

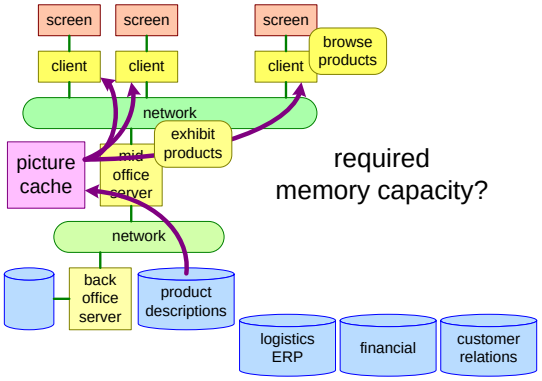


Figure 1.13: What Memory Capacity is Required for Picture Transfers?

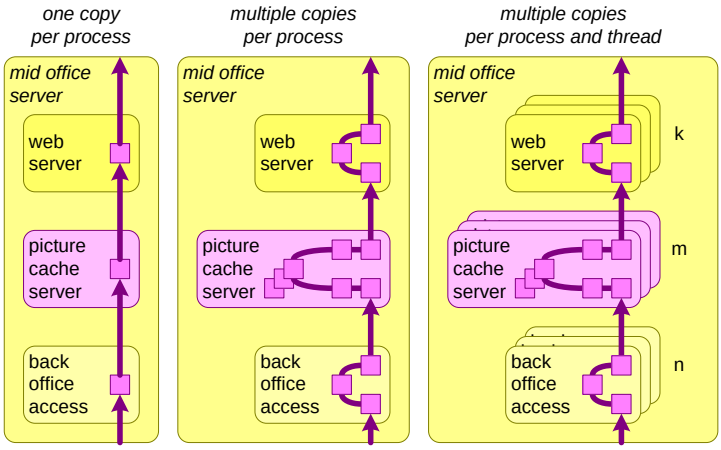


Figure 1.14: Process view of picture flow in web server

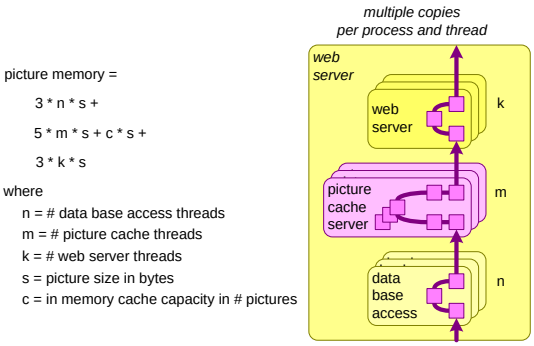


Figure 1.15: Formula memory Use Web Server

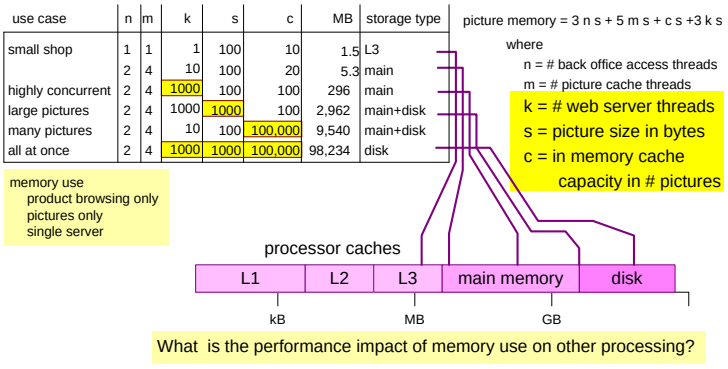


Figure 1.16: Web server memory capacity

1.4 Discussion

In the previous section we have modeled a few parts of the system. Figure 1.17 shows that we have covered so far a small part of the system space We can describe the system space by several dimensions: functions, data and aspects. Our coverage so far has been limited to the browse and exhibit products function, looking at the pictures as data, looking at the aspects of server memory use, response time and server load.

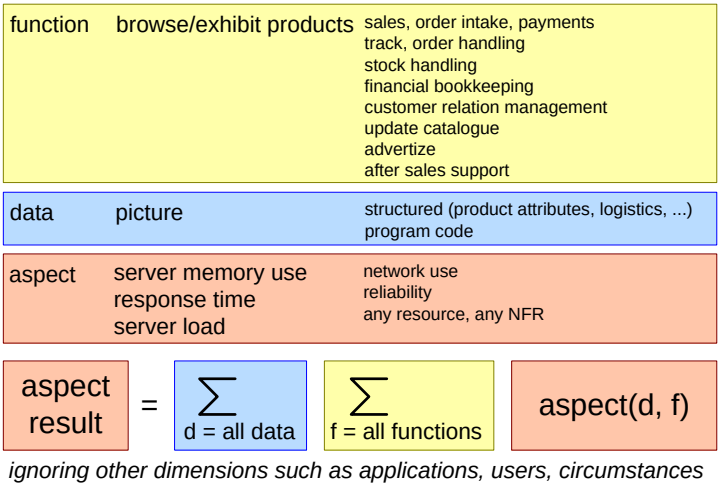


Figure 1.17: Only a small part of the system has been modeled so far

This figure shows many more functions, types of data and aspects present in systems. To answer one of the NFR like questions we have to combine the aspect results of functions and all data types. In practice the context also impacts the NFR's, we have still ignored applications, users, and circumstances.

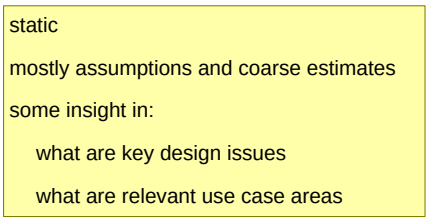


Figure 1.18: The modeling so far has resulted in understand some of the systems aspects

Figure 1.18 adds to this by reminding us that we so far have only made static

models, mostly based on assumptions and coarse estimates. Nevertheless we have obtained some insight in key design issues and relevant use cases.

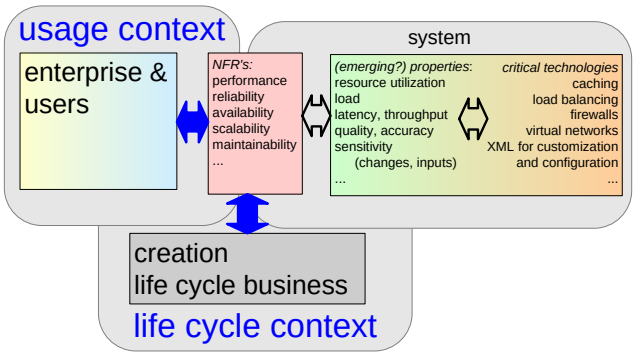


Figure 1.19: Refinement of the system models takes place after context modeling

We are far from finished with system modeling. However, with the results obtained so far it is important to take the next step of the broader iteration: modeling of the contexts, see Figure 1.19. Many of the models we have made of the system trigger questions about the system use and the life cycle. What is the expected amount of browsing, by how many customers, for what size of catalogue? What is the preferred picture quality? How relevant is the maintenance effort related to the product catalogue? et cetera.

1.5 Summary

Conclusions
Non Functional Requirements are the starting point for system modeling
Focus on highest ranking relations between NFR's and critical technologies
Make simple mathematical models
Evaluate quantified instantiations
Techniques, Models, Heuristics of this module
Non functional requirements
System properties
Critical technologies
Graph of relations

Figure 1.20: Summary of system modeling

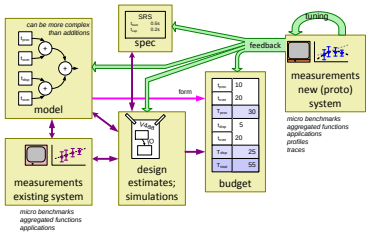
Figure 1.20 shows a summary of this paper. We have shown that Non Functional Requirements are the starting point for system modeling. Our approach focuses on the highest ranking relations between NFR's and critical technologies. For these relations we make simple mathematical models that are evaluated by quantified instantiations of these models.

1.6 Acknowledgements

Roelof Hamberg caught several errors in the detailed models

Chapter 2

Modeling and Analysis: Budgeting



2.1 Introduction

Budgets are well known from the financial world as a means to balance expenditures and income. The same mechanism can be used in the technical world to balance for instance resource use and system performance.

A **budget** *is*
a **quantified instantation** *of a* **model**

A **budget** *can*
prescribe *or* **describe** *the* **contributions**
by **parts** *of the* **solution**
to the **system quality** *under consideration*

Figure 2.1: Definition of a budget in the technical world

Budgets are more than an arbitrary collection of numbers. The relationship

between the numbers is guided by an underlying model. Figure 2.1 shows *what* a budget is. Technical budgets can be used to provide guidance by prescribing allowed contributions per function or subsystem. Another use of budgets is as a means for understanding, where the budget describes these contributions.

We will provide and illustrate a budget method with the following attributes:

- a goal
- a decomposition in smaller steps
- possible orders of taking these steps
- visualization(s) or representation(s)
- guidelines

2.2 Budget-Based Design method

In this section we illustrate a *budget-based design* method applied at waferstepper, health care, and document handling systems, where it has been applied on different resources: overlay, memory, and power.

2.2.1 Goal of the method

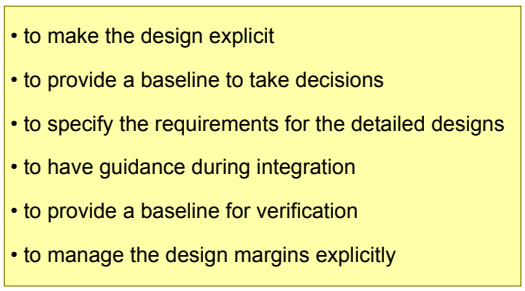


Figure 2.2: Goals of budget based design

The goal of the budget-based design method is to guide the implementation of a technical system in the use of the most important resource constraints, such as memory size, response time, or positioning accuracy. The budget serves multiple purposes, as shown in Figure 2.2.

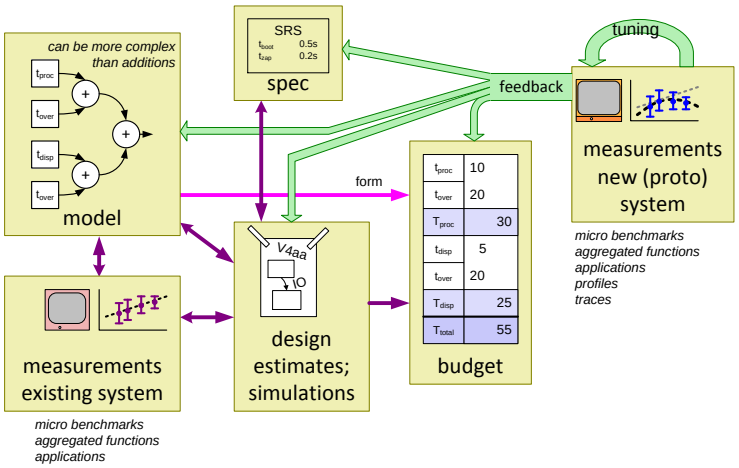


Figure 2.3: Visualization of Budget-Based Design Flow. This example shows a response time budget.

2.2.2 Decomposition into smaller steps

Figure 2.3 visualizes the budget-based design flow. This visualization makes it clear that although the budget plays a central role in this design flow, cooperation with other methods is essential. In this figure other cooperating methods are performance modeling, micro-benchmarking, measurement of aggregated functions, measurements at system level, design estimates, simulations, and requirements specification.

Measurements of all kinds are needed to provide substance to the budget. Micro-benchmarks are measurements of elementary component characteristics. The measured values of the micro-benchmarks can be used for a bottom-up budget. Measurements at functional level provide information at a higher aggregated level; many components have to cooperate actively to perform a function. The outcome of these function measurements can be used to verify a bottom-up budget or can be used as input for the system level budget. Measurements in the early phases of the system integration are required to obtain feedback once the budget has been made. This feedback will result in design changes and could even result in specification changes. The use of budgets can help to set up an integration plan. The measurement of budget contributions should be done as early as possible, because the measurements often trigger design changes.

2.2.3 Possible order of steps

Figure 2.4 shows a budget-based design flow (the *order* of the method). The starting point of a budget is a model of the system, from the conceptual view.

step	example
1A measure old systems	micro-benchmarks, aggregated functions, applications
1B model the performance starting with old systems	flow model and analytical model
1C determine requirements for new system	response time or throughput
2 make a design for the new system	explore design space, estimate and simulate
3 make a budget for the new system:	models provide the structure measurements and estimates provide initial numbers specification provides bottom line
4 measure prototypes and new system	micro-benchmarks, aggregated functions, applications profiles, traces
5 Iterate steps 1B to 4	

Figure 2.4: Budget-based design steps

An existing system is used to get a first guidance to fill the budget. In general the budget of a new system is equal to the budget of the old system, with a number of explicit improvements. The improvements must be substantiated with design estimates and simulations of the new design. Of course the new budget must fulfill the specification of the new system; sufficient improvements must be designed to achieve the required improvement.

2.2.4 Visualization

In the following three examples different actually used *visualizations* are shown. These three examples show that a multi-domain method does not have to provide a single solution, often several useful options exist. The method description should provide some guidance in choosing a visualization.

2.2.5 Guidelines

A *decomposition* is the foundation of a budget. No universal recipe exists for the decomposition direction. The construction decomposition and the functional decomposition are frequently used for this purpose. Budgets are often used as part of the design specification. From project management viewpoint a decomposition is preferred that maps easily on the organization.

The architect must ensure the *manageability* of the budgets. A good budget has tens of quantities described. The danger of having a more detailed budget is loss of overview.

The simplification of the design into budgets introduces design constraints. Simple budgets are entirely static. If such a simplification is too constraining or too

costly then a dynamic budget can be made. A dynamic budget uses situationally determined data to describe the budget in that situation. For instance, the amount of memory used in the system may vary widely depending on the function or the mode of the system. The budget in such a case can be made mode-dependent.

2.2.6 Example of overlay budget for wafersteppers

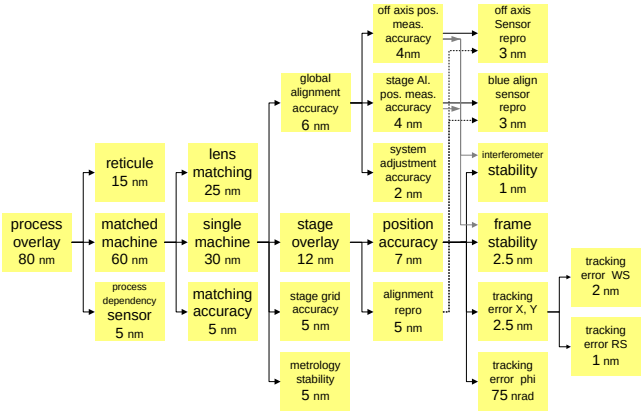


Figure 2.5: Example of a quantified understanding of overlay in a waferstepper

Figure 2.5 shows a graphical example of an "overlay" budget for a waferstepper. This figure is taken from the *System Design Specification* of the ASML TwinScan system, although for confidentiality reasons some minor modifications have been applied.

The *goal* of the overlay budget is:

- to provide requirements for subsystems and components.
- to enable measurements of the actual contributions to the overlay during the design and integration process, on functional models or prototypes.
- to get early feedback of the overlay design by measurements.

The *steps* taken in the creation, use and validation of the budget follow the description of Figure 2.4. This budget is based on a model of the overlay functionality in the waferstepper (step 1B). The system engineers made an explicit model of the overlay. This explicit model captures the way in which the contributions accumulate: quadratic summation for purely stochastic, linear addition for systematic effects and some weighted addition for mixed effects. The waferstepper budget is created by measuring the contributions in an existing system (step 1A). At the same time a top-down budget is made, because the new generation of machines needs

a much better overlay specification than the old generation (step 1C). In discussions with the subsystem engineers, design alternatives are discussed to achieve the required improvements (step 2 and 3). The system engineers also strive for measurable contributions. The measurability of contributions influences the subsystem specifications. If needed the budget or the design is changed on the basis of this feedback (step 4).

Two *visualizations* were used for the overlay budget: tables and graphs, as shown in Figure 2.5.

The overlay budget plays a crucial role in the development of wafersteppers. The interaction between the system and the customer environment is taken into account in the budget. However, many open issues remain at this interface level, because the customer environment is outside the scope of control and a lot of customer information is highly confidential. The translation of this system level budget into mono-disciplinary design decisions is still a completely human activity with lots of interaction between system engineers and mono-disciplinary engineers.

2.2.7 Example of memory budget for Medical Imaging Workstation

The *goal* of the memory budget for the medical imaging workstation is to obtain predictable and acceptable system performance within the resource constraints dictated by the cost requirements. The *steps* taken to create the budget follow the order as described in Figure 2.4. The *visualization* was table based.

memory budget in Mbytes	code	obj data	bulk data	total
shared code	11.0			11.0
User Interface process	0.3	3.0	12.0	15.3
database server	0.3	3.2	3.0	6.5
print server	0.3	1.2	9.0	10.5
optical storage server	0.3	2.0	1.0	3.3
communication server	0.3	2.0	4.0	6.3
UNIX commands	0.3	0.2	0	0.5
compute server	0.3	0.5	6.0	6.8
system monitor	0.3	0.5	0	0.8
application SW total	13.4	12.6	35.0	61.0
UNIX Solaris 2.x				10.0
file cache				3.0
total				74.0

Figure 2.6: Example of a memory budget

The rationale behind the budget can be used to derive *guidelines* for the creation of memory budgets. Figure 2.6 shows an example of an actual memory budget for a medical imaging workstation from Philips Medical Systems. This budget decomposes the memory into three different types of memory use: code (”read only” memory with the program), object data (all small data allocations for control and

bookkeeping purposes) and bulk data (large data sets, such as images, which is explicitly managed to fit the allocated amount and to prevent memory fragmentation). The difference in behavior of these three memory types is an important reason to separate into different budget entries. The operating system and the system infrastructure, at the other hand, provide means to measure these three types at any moment, which helps for the initial definition, for the integration, and for the verification.

The second decomposition direction is the *process*. The number of processes is manageable, since processes are related to specific development teams. Also in this case the operating system and system infrastructure support measurement at process level.

The memory budget played a crucial role in the development of this workstation. The translation of this system level budget into mono-disciplinary design decisions was, as in the case of overlay in wafersteppers, a purely human activity. The software discipline likes to abstract away from physical constraints, such as memory consumption and time. A lot of room for improvement exists at this interface between system level design and mono-disciplinary design.

2.2.8 Example of power budget visualizations in document handling

Visualizations of a budget can help to share the design issues with a large multi-disciplinary team. The tables and graphs, as shown in the previous subsections, and as used in actual practice, contain all the information about the resource use. However the *hot spots* are not emphasized. The visualization does not help to see the contributions in perspective. Some mental activity by the reader of the table or figure is needed to identify the design issues.

Figure 2.7 shows a visualization where at the top the physical layout is shown and at the bottom the same layout is used, however the size of all units is scaled with the allocated power contribution. The bottom visualization shows the *power foot print* of the document handler units.

Figure 2.8 shows an alternative power visualization. In this visualization the energy transformation is shown: incoming electrical power is in different ways transformed into heat. The width of the arrows is proportional to the amount of energy. This visualization shows two aspects at the same time: required electrical power and required heat disposition capacity, two sides of the same coin.

2.2.9 Evolution of budget over time

Figure 2.9 shows a classification for budget types. It will be clear that already with four different attributes the amount of different types of budgets is large. Every type of budget might have its own peculiarities that have to be covered by the method. For instance, worst case budgets need some kind of over-kill prevention.

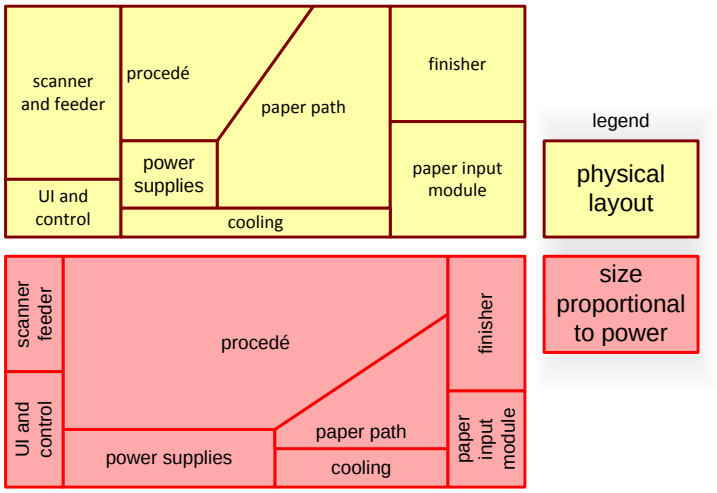


Figure 2.7: Power Budget Visualization for Document Handler

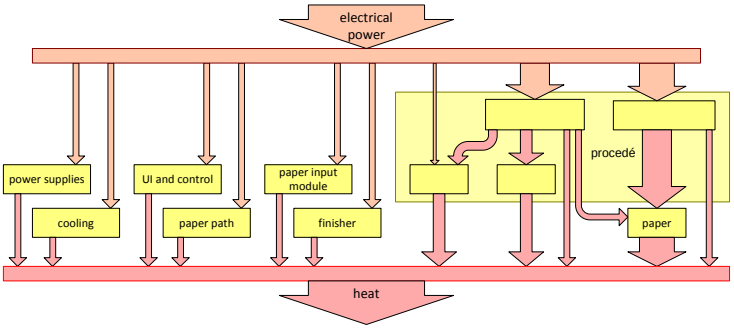


Figure 2.8: Alternative Power Visualization

Add to these different types the potential different purposes of the budget (design space exploration, design guidance, design verification, or quality assurance) and the amount of method variations explodes even more.

We recommend to start with a budget as simple as possible:

- coarse guesstimate values
- typical case
- static, steady state system conditions
- derived from existing systems

This is also shown in Figure 2.10. This figure adds the later evolutionary increments, such as increased accuracy, more attention for boundary conditions and

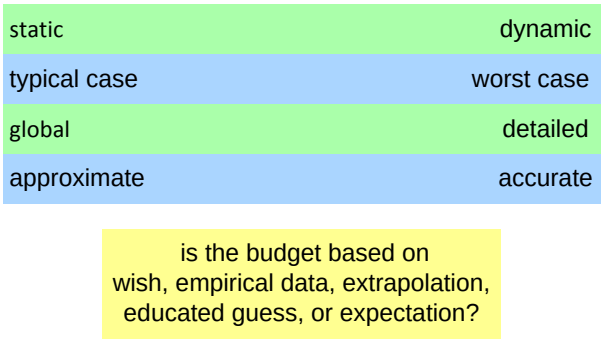


Figure 2.9: What kind of budget is required?

dynamic behavior.

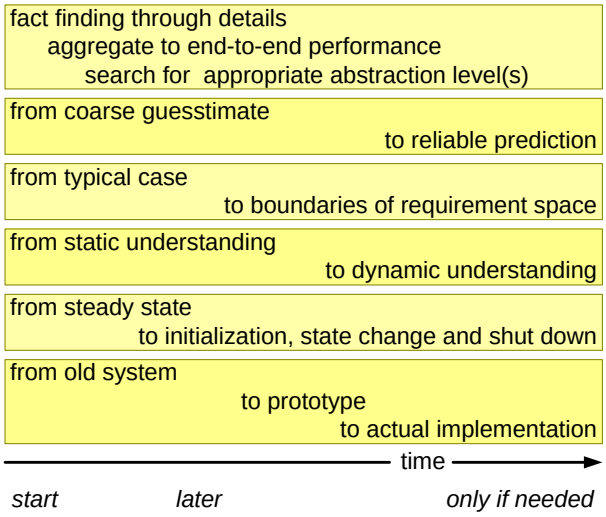


Figure 2.10: Evolution of Budget over Time

However, some fact finding has to take place before making the budget, where lots of details can not be avoided. Facts can be detailed technical data (memory access speed, context switch time) or at customer requirement level (response time for specific functions). The challenge is to mold these facts into information at the *appropriate* abstraction level. Too much detail causes lack of overview and understanding, too little detail may render the budget unusable.

2.2.10 Potential applications of budget method

For instance the following list shows potential applications, but this list can be extended much more. At the same time the question arises whether *budget-based design* is really the right submethod for these applications.

- resource use (CPU, memory, disk, bus, network)
- timing (response time, latency, start up, shutdown)
- productivity (throughput, reliability)
- image quality (contrast, signal to noise ratio, deformation, overlay, depth-of-focus)
- cost, space, time, effort (for instance expressed in lines of code)

2.3 Summary

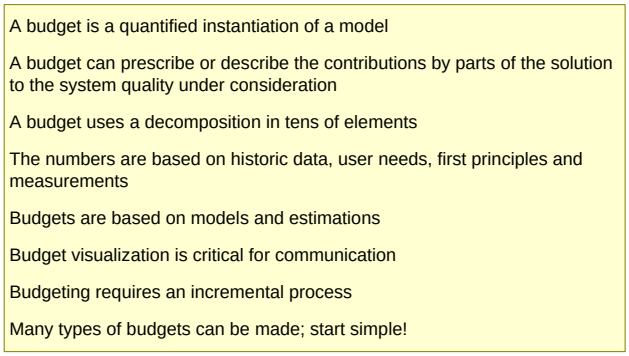


Figure 2.11: Summary of budget based design

2.4 Acknowledgements

The Boderc project contributed to the budget based design method. Figure 2.12 shows the main contributors.

The Boderc project contributed to Budget Based Design. Especially the work of

Hennie Freriks, Peter van den Bosch (Océ),

Heico Sandee and *Maurice Heemels* (TU/e, ESI)

has been valuable.

Figure 2.12: Colophon

Bibliography

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

History

Version: 0.4, date: 6 March, 2007 changed by: Gerrit Muller

- updated content slide
- added position slide

Version: 0.3, date: 23 February, 2007 changed by: Gerrit Muller

- added System Model
- changed status to preliminary draft

Version: 0.2, date: 5 January, 2007 changed by: Gerrit Muller

- added budgetting

Version: 0.1, date: 2 January, 2007 changed by: Gerrit Muller

- changed Module title in Modeling and Analysis: System Model
- added presentation System Model
- added light weight simulation presentation

Version: 0, date: 8 December, 2006 changed by: Gerrit Muller

- created module