

Threads of Reasoning: A Case Study

J.H. Sandee
Technische Universiteit Eindhoven
Dept. of Elec. Eng., Control Systems
PO Box 513, 5600 MB Eindhoven
The Netherlands
j.h.sandee@tue.nl

W.P.M.H. Heemels, G.J. Muller
Embedded Systems Institute
PO Box 513, 5600 MB Eindhoven
The Netherlands

P.F.A. van den Bosch
Océ Technologies BV
PO Box 101, 5900 MA Venlo
The Netherlands

M.H.G. Verhoef
Chess Information Technology BV
PO Box 5021, 2000 CA Haarlem
The Netherlands

Sixteenth Annual International Symposium of the
International Council On Systems Engineering (INCOSE)
8 - 14 July 2006

Copyright © 2006 by J.H. Sandee. Published and used by INCOSE with permission.

Abstract. In the design of technology intensive products like copiers, wafer steppers and televisions, one searches for a product that satisfies the product requirements as well as the business drivers. The main need in an early design phase is to bring structure in the typical chaos of uncertainty and the huge amount of realization options present. Potential realization choices all have advantages and disadvantages, which cause tensions and conflicts. The earlier the (essential) conflicts and tensions are identified, the better it is. Turning them from implicit to explicit helps the system architect in making the trade-off consciously or at least in selecting the most important tensions and conflicts that require further in-depth investigation. In this respect we demonstrate the effectiveness of a technique called “threads of reasoning”. The illustrative case study is the design of the paper flow control (sensors, actuators, control architecture, etc.) in a high-volume copier/printer.

This work has been carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute.
This project is partially supported by the Netherlands Ministry of Economic Affairs under the Senter TS program.

1 Introduction

The complexity of products being designed by industry today is increasing at an astonishing rate. The search is for a product that will satisfy the requirements within certain margins, e.g. development costs, production costs, response time, time to market, physical dimensions, power consumption, noise production and so on. Often, these requirements are conflicting, so that a right balance must be found.

The main need in the design of a product is to bring structure in the typical chaos of uncertainty and the huge amount of options present. This is most profound in the early design phase. Even typical product requirements might be uncertain in the sense that they are only known up to a certain degree or are still open for discussion. Potential solutions or applied technologies all have advantages and disadvantages, which cause tensions and conflicts. For instance, in the design of a printer one might consider using stepper motors, DC servo-motors or

a combination of both for driving the sheets of paper through the paper path. While stepper motors have the advantage of being cheaper (particularly as they do not require expensive encoders and because of their long lifetime), they are in general less accurate in positioning the sheets of paper. This causes a conflict between the important requirements printing accuracy on one hand and cost price on the other. Of course, more requirements might play a role in such a decision (e.g. size, power consumption, etc).

This article describes the technique of *threads of reasoning* (Muller 2004) to find such tensions. The technique is illustrated by its application in the design of the paper flow control of a high-volume copier/printer. The details of the technique are given together with a 5-step iterative scheme on how to create these threads. Once identified, the main tensions and conflicts are further investigated by modeling and measurements. The specific model-based investigations are only indicated briefly.

In several communities there are alternative and / or related techniques available to identify the main relations, conflicts and tensions in the design of a product. For instance, in requirement engineering and more particular in (Wieringa 2004) one uses the term “problem bundle” that has similar properties as a thread of reasoning. In (Wieringa 2004) these bundles are adopted for structuring a design problem at hand and relating this to the solution space. In product line engineering one has methods like Pulse (see e.g. (Bayer et al. 1999)) and in the system engineering community one uses risk management approaches (INCOSE technical board 2004, chapter 6). These techniques create similar overviews, but more retrospective. Also in VAP (visual architecting process), see (Malan et al. 2005, chapter 2), and in ARES (Architectural Reasoning for Embedded Software) (Jazayeri et al. 2000) related techniques can be found. In TRIZ (Altshuller 2000) two important concepts are introduced that are also crucial in our reasoning method: formulating the “ideal” solution and identifying the conflicts in realizing the ideal product. Quality function deployment (QFD) (ReVelle 1998) relates product requirements of the customer to design choices, that has from an abstract point of view resemblance with the reasoning used in this article. However, a distinguishing feature of threads of reasoning is that it is graph- instead of matrix-oriented. Matrix-oriented techniques have the tendency that the number of relationships easily explodes and one easily loses overview of the essential threads. Threads of reasoning are particularly focused on keeping only the essential tensions and conflicts, which we consider an advantage. As a consequence, it is possible to graphically represent the overview of the most important design issues. Moreover, most of the mentioned methods have a tendency to move more towards the customer context and less to the realization aspects. The case study here shows how threads of reasoning can also be used for conceptual and realization choices of the technical design.

The disadvantage of the explosion of the number of relationships is also encountered in a complementary approach in which one archives the design process including the conceptual and realization choices (Alexander 2002). Often the argumentation why a certain choice has been made is included as well. The documentation typically consists of a chronologically ordered sequence of choices with the aim of traceability: how was a certain choice made at some point in time? If some design changes are made in a later stage, one can still apply the reasoning as kept in the archive. In practice this is often not doable due to the enormous complexity, which often is the cause that the “tracing” is not kept up-to-date with the consequence that its value diminishes. Threads of reasoning aim at keeping the essence of the design choices and help to keep overview. In addition, although tracing techniques have their own added value, their maintenance requires much more effort than using threads of reasoning.

The outline of the article is as follows. In the next section we describe the industrial context being document printing systems. We also indicate the problem statement of the article and put it in the perspective of multi-disciplinary design. The exact case study on the design of the control architecture for the flow of sheets through the printer is also given in section 2. In section 3 “threads of reasoning” is described. In section 4 threads of reasoning are applied to identify the tensions and conflicts in the case study. This leads to tensions that require a further study via modeling, measurements or other techniques to obtain a well-founded trade-off. In the same section we indicate briefly which models have been applied to do the in-depth analysis. In section 5 the conclusions are stated.

2 Printer design context and problem formulation

2.1 Research topic

The *research topic* of the current article is coupled to the Boderc project (see Embedded Systems Institute 2003), which aims at developing a design methodology based on multi-disciplinary modeling to predict the performance of a system in an early design phase (see figure 2.1). The *methodology* (Heemels et al. 2006) consists of several modeling *formalisms*, analysis *techniques* to make implicit knowledge in models explicit, *tools* to support modeling and analysis, and a *method* providing guidelines how and in which order to apply formalisms, techniques and tools.

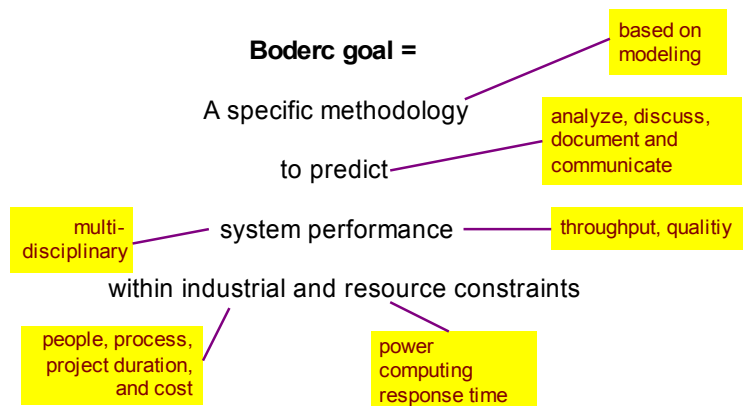
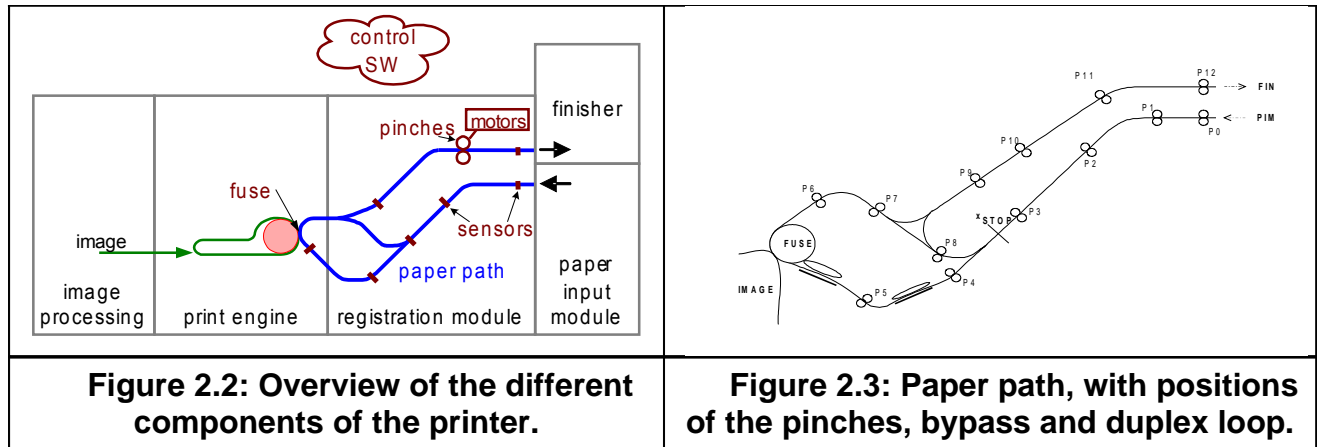


Figure 2.1: Annotated research goal of Boderc

Typically, the industrial constraints of project duration and available man power imply that one should focus the analysis (via modeling) on the most critical issues (instead of “wasting” effort on less relevant problems). Therefore, one of the steps in the Boderc method will be the identification of the most *essential* conflicts and tensions. This is where the current article is positioned. Threads of reasoning, as explained in section 3, is one of the techniques to find these essential tensions, to create overview and insight in the overall design and indicate on which issues further (model-based) analysis is required.

2.2 Industrial case description

The case study zooms in on the embedded control of the paper flow through a high-volume document copying and printing system. As we will not consider the scanning part of the copier, we will use the term *printer* in this article.



In figure 2.2 a schematic overview of the printer is presented. A sheet is separated from the trays in the Paper Input Module (PIM), after which it is sent to the paper path that transports the sheets accurately in the direction of the print engine, where the image is fused on a sheet of paper. After that, the sheet is transported by the paper path to the finisher (FIN). In figure 2.3 a more detailed drawing of a paper path is given.

The design of a printer should be such that *performance indicators* like

- throughput (pages per minute),
- printing accuracy (positioning of the image on the sheet),
- time-to-first-print (the time it takes before the first sheet comes out of the printer, after pressing “start”)

are satisfactory within certain *resource constraints* like

- power usage,
- cost price,
- size,

and *business constraints* like

- time-to-market and
- available man-power, amongst others.

The term *main design drivers* will be used for the above items.

For the case study, the mechanical lay-out is already given meaning that positions of (paper transport) pinches, the length and shape of the paper path, etc. are known. The design process is in the phase of selecting the control architecture, including:

- Selection of actuators (type and number of motors)
- Selection of sensors
- Selection of the processing architecture (e.g. centralized versus distributed control)
- Selection of operating system (event-driven or time sliced architectures?)
- How will the sheets be scheduled given a print job?

All these choices for the control architecture should be satisfactory in view of the main design drivers as mentioned above.

3 Threads of reasoning

Threads of reasoning is an iterative and informal technique to identify the most important points of tension in the problem and potential solutions. The system architect uses threads of reasoning implicitly to integrate various views in a consistent and balanced way, to get a valuable, usable and feasible product. Architects perform this job by continuously iterating over many different points of view and sampling the problem and solution space to build up an understanding of the case. These threads are made explicit by the technique of threads of reasoning.

This technique, as presented in the next section, is based on the work (Muller 2004, chapter 12). A difference between the technique used here and the one by Muller lies in the categories or views used. In particular, threads of reasoning in (Muller 2004) uses the CAFCR framework that adopts the “Customer objectives” (addressing the “what” question from the customer perspective), “Application” (addressing the “how” question of the customer), “Functional” (addressing the “what” question of the product), “Conceptual” and “Realization” views (addressing the “how” of the product). Instead of the CAFCR views, it was in our case more suitable to use the following categories:

- *main design drivers*: important requirements of the system design (typically applying to system level), see section 2.2.
- *sub drivers*: drivers, derived from the main design drivers (typically applying to subsystem level)
- *design choices*: possible solutions or realizations
- *consequences*: indicating consequences of a design choice

The threads themselves are formed by multiple connections between the categories above.

3.1 Overview of threads of reasoning

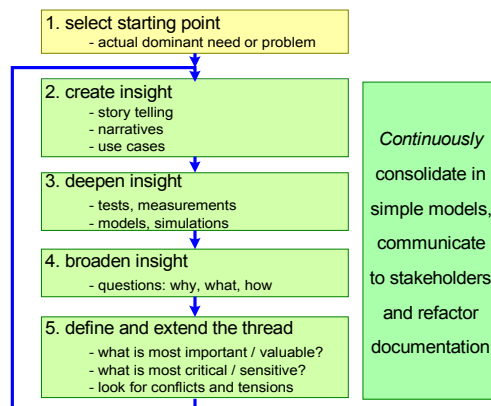


Figure 3.1: Overview of reasoning approach

Figure 3.1 gives an overview of the threads of reasoning technique. Step 1 is to *select a starting point*. After step 1 the iteration starts with step 2 *create insight*. Step 3 is *deepening the insight* and step 4 is *broadening the insight* via suitable questions. Step 5 *defines and extends the thread*. Moreover, the next iteration is prepared by step 5. In step 5, first the most important and critical threads are selected and one aims at finding conflicts and tensions. This insight and refinement might lead to selecting the next need or problem for the new iteration. During this iteration continuous effort is required to *communicate with the stakeholders* to keep them up to

date, to *consolidate in simple models* the essence of the problem and to *refactor the documentation* to keep it up to date with the insights obtained.

As mentioned before, the focus of threads of reasoning in the Boderc design methodology is to select the critical design issues (step 5) that require in-depth studies (via modeling) to make a sound design trade-off. The in-depth studies are essentially step 3 in figure 3.1. The limited models for *consolidation, communication and reasoning* are derived from these possibly more complex and detailed models for *analysis*. Especially, since these deep studies require a major part of the design time, one has to be selective in the ones that are actually carried out. Of course, this does not mean that once the answers of these analyses have been obtained, the thread of reasoning is finished. On the contrary, it might actually be altered based on the findings or continued given these new pieces of information.

Below we will describe each of the individual steps in more detail. Moreover, we will present already one thread of reasoning as an example from the case study to illustrate the steps.

Step 1: Select a starting point. A good starting point is to take a need or problem that is very hot at the moment. If this issue turns out to be important and critical then it needs to be addressed anyway. If it turns out to be not that important, then the outcome of the first iteration serves to diminish the worries in the organization, enabling it to focus on the really important issues. In practice there are many hot issues that after some iterations turn out to be non-issues. This is often caused by non-rational fears, uncertainty, doubt, rumors, lack of facts, etc. Going through the iteration, which includes fact finding, quickly positions the issues. The actual dominant needs or problems can be found by listening to what is mentioned with the greatest loudness, or which items dominate in all discussions and meetings.

Example. An important issue in the paper flow control is the question how many processing nodes should be used. Because of the size and the complexity of the software, which is both soft real-time and hard real-time for the various implemented functions, it is almost impossible to process all the code on one node, i.e. one processor. Nevertheless, there are various ways to distribute the software functionality over different (numbers of) nodes. There can be several 'local nodes' that handle separately the control of single motors. Another option is to have only two big processing nodes that handle the entire paper flow control. This design choice is selected as the starting point of the thread.

Step 2: Create insight. In this phase one wants to obtain a rough overview and insight of the chosen issue. The selected issue can be considered by means of one of the many (sub)methods to create more understanding. Typically, this can be done by the submethods story telling (Muller 2004, chapter 11), narratives (Cockburn, 2000) or scenario-based reasoning using e.g. use-cases (Cockburn, 2000). Using these submethods, it will quickly become clear what is known (and can be consolidated and communicated) and what is unknown, and what needs more study and hence forms input for the next step.

Example. To create some first insight into the problem of selecting the number and sizes of the processors in the control architecture, we linked this issue to the main design drivers (section 2.1). For the time-to-market to be short, it is important to have a predictable development process. Therefore, a concurrent design process is preferred, which is in favor of having multiple processing nodes. On the other hand, we also want the cost price to be low. Here, the question pops up how the cost price relates to the number of nodes. Looking at the driver power consumption, there is some relation (more nodes require more power?), but more specific information is needed to reveal the true relation and its importance.

Step 3: Deepening the insight. The insight is deepened by gathering specific facts. This can be done by modeling (simulation), or by tests and measurements on existing systems. Since the presented technique is iterative, in a first iteration one aims at using simple models, measurements or facts that are obtained in a reasonably short time. Typically back-of-the-envelope calculations or rules of thumb that are known from previous projects are useful. In a second or subsequent iteration one selects the essential issues (most uncertain, most important) that require more modeling and analysis effort. This aspect is coupled directly to the Boderc design methodology (see Section 2.1) based on multi-disciplinary modeling: to discover and select the in-depth modeling activities that have to be performed to support the system architect in taking (well-founded) design choices. Typically, the models are aimed at shedding light on the tensions and conflicts, which were identified earlier (step 5, first iteration).

Example. To get deeper insight in the issues of cost price and power usage of processors, more specific information is needed. For the cost price it turned out that the use of more nodes produces higher costs, mainly because of production costs of the supporting hardware for these processors. A rough quantitative estimate showed that the price per node is typically about 40 euros, of which 10 euros is calculated for the controller and 30 euros for the printed circuit board (PCB). Because for every node a separate PCB is used, doubling the number of processors roughly means doubling the cost price, although the cost price of the processor can be somewhat less for simple variants. Looking at power demands, it turned out that both the smaller and the bigger processors use about 3 Watt. It would therefore be beneficial to have as few processors as possible. On the other hand, if we look at the power demands from other modules in the printer, that use up to 2 kW, we assume that the power demand from the processors is of minor importance (Freriks et al. 2005a). Therefore, the power issue will not be included in this thread of reasoning as we aim at describing only the most important aspects.

Step 4: Broadening the insight. Needs and problems are never nicely isolated from the context. In many cases the reason why something is called a problem is because of the interaction between the function and the context. The insight is broadened by relating the need or problem to the other views or categories. This can be achieved by answering *why*, *what* and *how* questions. Examples: How can a main design driver be realized by sub drivers? How is a certain issue tackled? Why is a certain design choice good for a specific main design driver? What are the consequences of a design choice? How is the consequence related to a specific driver? The insight in the main design driver dimension can also be broadened by looking at the interaction with related system qualities: what happens with safety or reliability when we increase the performance?

Example. If we separate the software over multiple nodes, how efficiently can the software still be implemented? What happens if all software would run on two processors (e.g. would there be problems with synchronization)? How would multiple processors be connected? Of course, these questions reveal the need for more facts, for which more iterations of the process are needed.

Step 5: Define and extend the thread. In the previous steps and corresponding discussion of the needs, design choices and problems, many new issues pop up. A single problem can trigger an avalanche of new problems. Key in the approach is not to drown in this infinite ocean full of issues, by addressing the relevant aspects of the problem. This is done by evaluating:

- 1) Which specification and design decisions seem to be the most conflicting, i.e. where is the most tension;

- 2) What is the value or the importance of the problem for the customer;
- 3) How challenging it is to solve, at least in the sense that problems that can be solved in a trivial way should immediately be solved;
- 4) How critical the implementation is. The implementation can be critical because it is difficult to realize, or because the design is rather sensitive or rather vulnerable (for example, hard real-time systems with processor loads close to 70% or higher).

To evaluate the above aspects, the system architect often uses 'gut-feeling' based on many years of experience. To do the evaluation in a more structured way, several methods are available. Analysis techniques, such as Failure Mode Effects and Criticality Analysis (FMECA) can be used to analyze the impact of potential problems in the system. Typically, these techniques are used when the design is finished but they can be equally productive during other life-cycle phases. To compare various solutions, trade studies (INCOSE technical board 2004, section 11.16) can effectively be applied.

The next crucial element is to define the thread: identification of the tension between needs and implementation options. The problem can be formulated in terms of this tension. We believe that a clearly articulated problem is half of the solution.

The insights obtained so far in terms of most crucial and critical threads and tensions, should help to select the new need or problem to go into the next iteration (back to step 2).

Example. At this moment in our reasoning on the number and size of processing nodes, the first thread becomes visible, as visualized in figure 3.2.

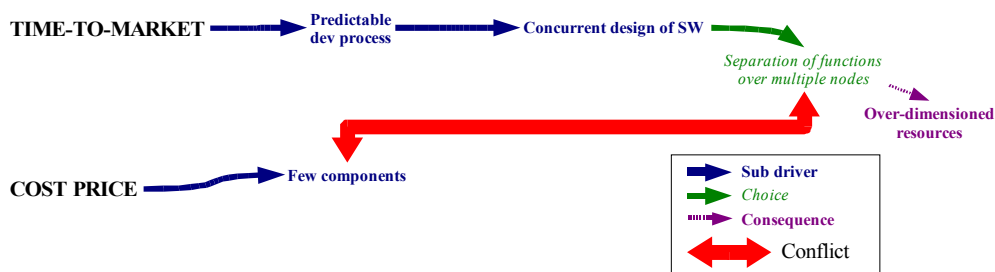


Figure 3.2: Example thread in the design of the paper flow control.

The thread is structured by means of the framework of the categories as introduced before. The interpretation of this visualization is as follows:

- On the left of the picture, the relevant *main design drivers* are given in capitals.
- From the main design drivers, *sub drivers* are derived, indicated in bold face (blue).
- Specific *design choices* result from these sub drivers, indicated in italic (green).
- The *consequences* that come with specific choices, are depicted with small dashed arrows (purple).
- The main conflicts that are identified between any of the above mentioned aspects of the system, are depicted with thick double arrows (red).

Note that in step 3 we already concluded that the main design driver power should not be included in this thread. Hence, a step 5 action of discarding less relevant aspects of a thread was already applied. We see that from the question of how many processing nodes to use, a conflict arises between the drivers 'time-to-market' and 'cost price'. As the most profound conflict is

identified now, this can be input for step 2 and subsequently step 3. More detailed models (in comparison with the simple estimates of cost price done earlier) would be very useful to deepen the insight, which would support in making of this trade-off in the early design phase. From our first simple models we concluded that for reasons of cost price we want as few processing nodes as possible. However, a proper software design should still be feasible within a limited time span (influencing time-to-market). Therefore, we created a Parallel Object Oriented Specification Language (POOSL) model (Putten et al. 1997). With this modeling language and analysis techniques, several possible architectures are evaluated and compared on their feasibility with respect to timing requirements. Note that a part of the argumentation of a particular choice is captured now in the specific models made. In another setting (or a different architecture) this can be used to reevaluate the design choice. So some kind of “tracing” – as discussed in the introduction – is kept.

The thread of reasoning of figure 3.2 was obtained by iterating one-and-a-half times through the 5-step scheme of figure 3.1. As we will see, this is typical for the case at hand as the aim of threads of reasoning in this setting is to select the in-depth models to be made. Normally more iterations – for instance, continuing after the modeling step – are used to find the essential tensions and conflicts.

4 Threads of reasoning for the case study

The structure that covers the most important threads and their relationships can be complicated for the design of complex systems, like a high-volume copier/printer. In addition to the thread presented previously, we will describe two other essential threads in the control of the paper flow. In the figures below we will use the same interpretation of the visualization as in figure 3.2.

4.1 Example thread: stepper motors versus DC servo-motors

In this second example thread, the starting point is the use of stepper motors instead of DC servo-motors for driving the pinches. The use of DC servo-motors is common for the printer manufacturer and less experience is present with stepper motors.

To create insight (step 2), the use of stepper motors was related to the identified main design drivers. It was easy to see that stepper motors relate to the cost price of the system, as the reason to select them in the first place was the fact that they are cheap. DC servo-motors are more expensive because of their need for (expensive) encoders and shorter lifetime. The use of stepper motors also relates to the printing accuracy. The accuracy of a stepper motor is limited because of various reasons, like its mechanical construction, cogging and overshoot (Freriks et al. 2005b). Because the stepper motors have to control the movement of the sheet, the sheet can only be controlled with limited accuracy.

To see whether the aspects discussed above are really important, we need to deepen our insight (step 3), in this case by quantifying the reasoning. The first aspect was the cost price. The average price of a (low power) stepper motor does not differ that much from the average cost price of a DC-motor. Both can be obtained (for large quantities) for typically less than 10 euros. For both types of motors an electrical driver is required, which also costs about the same for a stepper motor as for a DC-motor, i.e. circa 3 euros for low power applications. An encoder, which is solely needed to control the DC-motor, cannot be obtained below 20 euros for high resolution rotary encoders. This is the main reason why the use of stepper motors is preferred.

Another aspect that needs some quantification is the accuracy of the stepper motor. First measurements reveal that this indeed is an important issue. Figure 4.1 shows a plot of position

against time of a stepper motor running at 1 rotation/sec. Four steps are visualized of a 200 steps/revolution motor. The dashed line (blue) corresponds to the reference position, the solid line (red) to the actual measured position. The horizontal lines indicate the size of the four steps that are visualized. Each step of the motor can be translated to a step-size in the order of 0.2 mm of the paper. From the figure it can be seen that the inaccuracy in the motors position is about 1 step size, i.e. 0.2 mm. As the printing accuracy is defined at 1 mm, the paper needs to be positioned with an accuracy well below 1 mm. The obtained value of 0.2 mm is therefore critical and needs to be evaluated further. It is nevertheless hard to quantify the impact on the real position of the sheet, because of load differences, the occurrence of slip and interactions between two motors that are controlling the same sheet of paper for some period of time. Therefore, more extensive models are needed.

Note that the above reasoning illustrates the typical back-of-the-envelope calculations that quantify the reasoning.

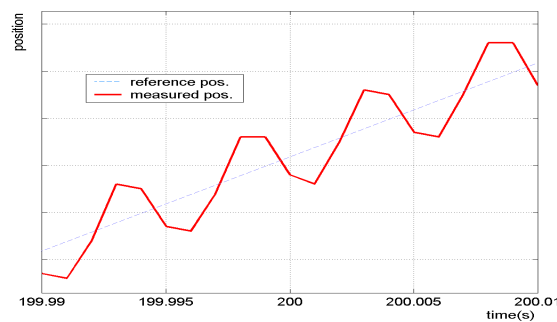


Figure 4.1: Measurement result of stepper motor

Like in the first example thread, we broaden our insight by means of the *how*, *what* and *why* questions (step 4). The first question could be how the motor should be controlled. The answer to this question is that a frequency generator needs to be implemented as for every step of the rotor, a drive pulse is needed. The follow-up question to this answer is how this frequency generator could be implemented. This pinpoints the question whether to do this with dedicated hardware or in software. Note that this question is a very common struggle nowadays in industry. It comes down to the question whether cost price or accuracy and predictability is more important. Normally, hardware implementations are more reliable and faster or more accurate, but increase the cost price of the system.

The last step in this first iteration is the visualization of the thread. This is depicted in figure 4.2. We see that two important conflicts have been identified that need more attention. The first one is the use of dedicated hardware for the frequency generator in relation to the use of few components to reduce the cost price. The second conflict is identified between the limited accuracy of stepper motors and the requirements on the control accuracy of the sheets.

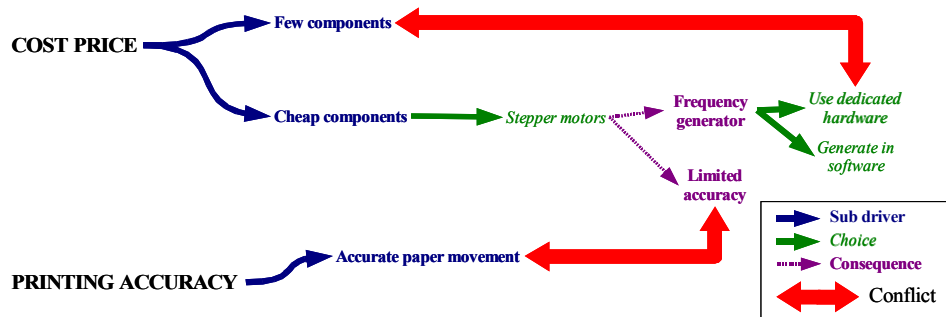


Figure 4.2: Thread of the example of stepper motors

4.2 Example thread: time sliced versus event-driven architecture

During the design a time sliced architecture was proposed for the processing nodes, on which for each node, multiple tasks are scheduled. The idea is that by assigning each task its own time slice, the execution of different functions is temporally separated and task interference is thus avoided. Therefore, software functions can be developed and tested separately while guaranteeing that it will work after combining them on one processor if each task fits in a slice and there are enough slices. The fact that this choice also has some important disadvantages, makes it a good starting point for a new thread (step 1).

To create insight (step 2), we again relate the issue to the main design drivers. The main reason for adapting the time sliced architecture is to shorten the time-to-market, as it enables predictable and composable software design. Furthermore, we can make use of existing knowledge from past experience of the printer manufacturer (since the time sliced architecture has been applied in the past).

One of the disadvantages of using time slices is the inefficient use of available processing power. Because each task gets a pre-determined part of the available processor time, tasks cannot use the slack time of each other. To quantify the inefficiency of the time sliced scheduling in our case (step 3), we created a simple spread-sheet model which shows the tasks, the expected processor usage and the size of the slices. It also includes an estimation of the interrupts that can occur. Because the interrupts can interrupt any task, a task can effectively take longer to execute than its measured execution time (without interruption). To guarantee the composability of the system, we have to take this interrupt overhead into account for every slice. It turned out that the overhead of the interrupts in a time sliced approach is 20%, while if we replace the time sliced approach by e.g. a rate monotonic scheduler, it becomes much less: 3%.

To broaden our insight (step 4), we could ask ourselves what the influence of the choice of the time sliced architecture would be on the printing accuracy. From past experience, but also from literature it is known that the time sliced architecture introduces a limited action-reaction speed. As we need very tight paper-image synchronization for accurate printing, this choice does influence the printing accuracy and therefore needs further in-depth investigation (via modeling).

Figure 4.3 visualizes this thread, together with the first two example threads. From the analysis above, two conflicts are identified between the use of the time sliced architecture (because of the main design drivers: time-to-market, cost price and printing accuracy).

4.3 Total overview

Figure 4.3 visualizes the three example threads of reasoning in one overview graph. It is interesting to see how these conflicts relate to each other. One example is found in the printing accuracy. The requirement of a high printing accuracy not only conflicts with the use of stepper motors, but also with the use of a time sliced architecture.

With the global overview we have obtained a clear list of tension spots where multi-disciplinary models will be made for deepening the insight (step 3). In figure 4.3, light grey (yellow) boxes are added to indicate the models that have been made. These models give more insight into the identified conflicts. As mentioned before, the threads of reasoning obtained here originate from one-and-a-half cycles through the 5-step scheme to end up with the in-depth models to be made. Although figure 4.3 originates from a limited set of starting issues and from only one-and-a-half iterations, it already shows a quite complicated structure. Nevertheless, the

overview already captures the most important tensions in the design of the control architecture.

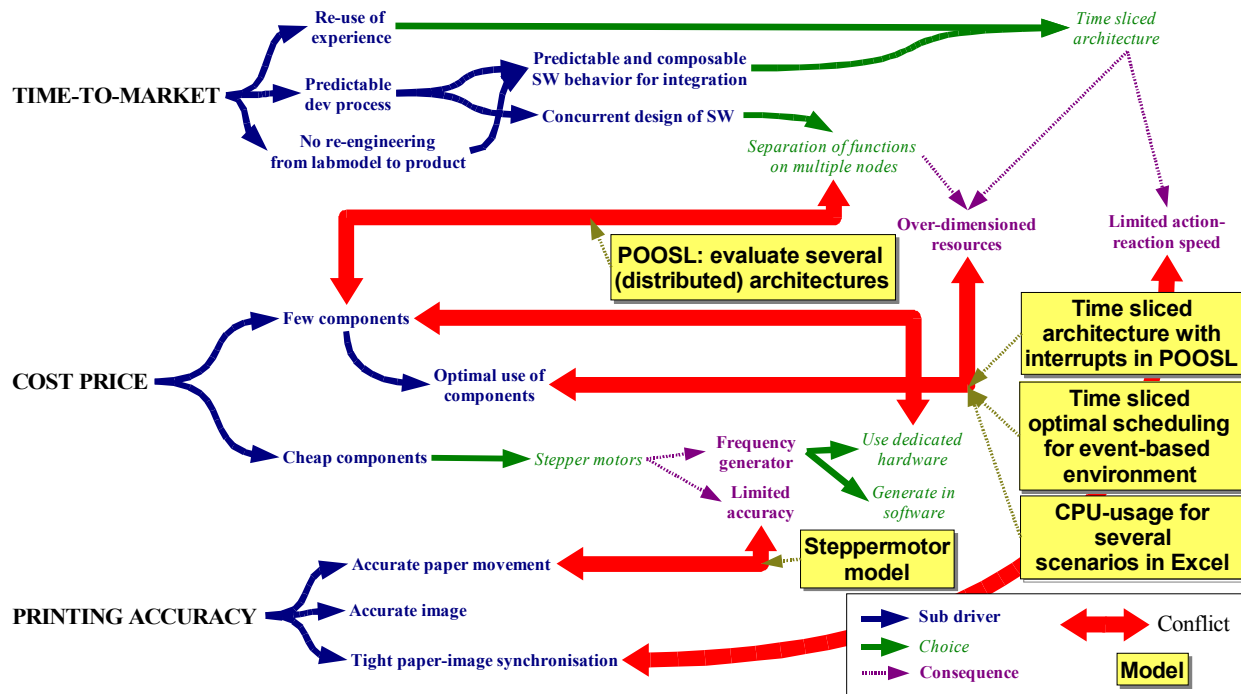


Figure 4.3: Global overview of several combined threads of reasoning.

4.4 Detailed models to obtain insight in conflicts

To deepen insight, especially at the tension spots in the design, specific models have been made. Figure 4.3 shows the objects of study of the models in the yellow/light gray boxes.

To get more insight in the conflict explained in section 3.1 (the size and number of processing nodes), a POOSL (Parallel Object Oriented Specification Language) model is created (Putten et al. 1997). With this modeling language and the analysis techniques, several possible architectures are evaluated and compared.

A second model was made in the language POOSL to analyze the processor load for the scenario in which the time sliced architecture is ‘polluted’ with interrupts, necessary to make optimal use of components. This is a more detailed model than the spread-sheet model described in section 4.2. Both models can also be used to see what the consequences are when the frequency generators for the stepper motors are implemented in software.

To make optimal use of the processors (and minimize the number of processors), a model was made to calculate optimal schedules for tasks in a time sliced architecture (Baruah et al 1997).

A stepper motor model, created in Matlab/Simulink, was used to analyze the positioning accuracy of stepper motors (Freriks et al. 2005b).

5 Conclusions

In this article the technique of threads of reasoning was applied to identify the most important tensions and conflicts in an industrial case study. The case consists of the multi-disciplinary design of the paper flow control in a high-volume copier/printer. Amongst other techniques, this technique helps to bring structure in the typical chaos of uncertainty and the huge amount of realization options present in early design phases.

Threads of reasoning is one of the techniques used in the (Boderc) design methodology that aims at using multi-disciplinary models to predict system performance in an early design phase, while respecting the business constraints of available man power and time-to-market. The restriction in available design time (related to time-to-market and available man power) implies that in-depth and often time-consuming modeling and analysis should be performed only for the essential and critical issues. Threads of reasoning turn out to be – at least in the case of designing the control architecture for a printer – an effective means to find these issues and to create overview.

Combined with the in-depth models, threads of reasoning provides the system architect with valuable insight that supports him in making the important design trade-offs and to reduce some of the uncertainty in the early design phase. It results in a very concise picture with the important tensions depicted *explicitly*. Especially, the combination with (multi-disciplinary) modeling leads to a design process that becomes more explicit. It forces the designer to quantify choices by replacing hand-waving with facts. This stimulates and focuses the discussion with the consequence of a shorter time-to-market and a more predictable design process. Moreover, a part of the argumentation of a particular design choice is captured now in the specific models made and techniques used.

Threads of reasoning form an informal technique in which some generic patterns can be observed. We captured the technique in an iterative procedure that consists of 5 steps. Of course, variations are possible to this procedure. One choice is the use of different views or categories as applied in this case study. Also the way the threads are formed can be different. One can for instance first perform a lot of viewpoint hopping to get broad (but shallow) threads or first go into depth within one view before changing viewpoint.

Based on the case study, the following suggestions for the use of threads of reasoning can be given:

- Keep the number and the size of the threads limited by selecting the most important ones to keep overview and not to drown in details. In our case study the entanglement was much larger in a first instance of figure 4.3. Additional iterations were used to regain focus and gave rise to figure 4.3 in its present form.
- Whether certain tensions are important, depends, amongst other things, also on the level of the system design you are in. The higher the level, the less detail has to be taken into account. Often though, some iterations will have to go quite deep in a short time to gather some facts that influence design choices at a much higher level. It helps to quantify things (even if the numbers might be uncertain in an early design phase) as it sharpens the discussion and replaces 'gut-feeling' by facts. In particular, back-of-the-envelope calculations, figures-of-merit and rules-of-thumb help to identify the essential tensions and to discard the unimportant ones.
- In the reasoning process, fast exploration of the problem and solution space improves the quality of the design decisions. It is important to *sample* specific facts and not to try to be

complete. The speed of iteration is much more important than the completeness of the facts. Otherwise the risk is to get stuck within one particular aspect. It is often sufficient to know the order of magnitude and the margin of error for the trade-off analysis. Be aware that the iteration will quickly zoom in on the core design problems, which will result in sufficient coverage of the issues anyway.

- It is essential to realize that such an exploration is highly concurrent; it is neither top-down, nor bottom-up. It is typically viewpoint hopping and taking different perspectives (views or categories) all the time.

We applied thread of reasoning to a relative simple case study, compared to for instance the design of a complete aircraft. To abstract up to more complicated systems, one can apply thread of reasoning recursively along various axes of decomposition. In the example of an airplane, one could start with applying threads of reasoning to the overall design, restricting oneself in not taking too much detail into account. Separate threads can then be created of the various decomposed parts of the airplane, like the motors and the navigation instruments. In the example of the copier, we could have created a separate thread of the image processing and corresponding hardware.

An open question still is how to learn the “skill” of threads of reasoning. Being able to iterate fast through the design space and views seems to be hard, and tends to be driven by experience. Making the trade-offs in little time seems to be a skill that you can only learn by doing it. However, the guidelines given in this article and the presented examples in the case study provide a first step for learning it.

References

- Alexander, I., "Towards automatic traceability in industrial practice." *Proceedings of the First International Workshop on Traceability*, Edinburgh, Sep 2002, pp 26-31.
- Altshuller, G., "The innovation algorithm. TRIZ, systematic innovation and technical creativity." Technical innovation center, Worcester, Massachusetts, 2000, online: <http://www.triz.org>.
- Baruah, S., D. Chen, A. Mok., "Jitter concerns in periodic task systems." *Proceedings of the Eighteenth Real-Time Systems Symposium*, pages 68-77, San Francisco, CA, Dec 1997.
- Bayer, J, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J.M. DeBaud, "PuLSE: A methodology to develop software product lines." *Proceedings of the symposium on software reusability*, pp: 122-131, 1999.
- Cockburn, A., *Writing effective use cases*. Addison-Wesley, 2000.
- Embedded Systems Institute., "Summary of the Boderc project plan." 2003. Online: <http://www.esi.nl>.
- Freriks, H.J.M., W.P.M.H. Heemels, G.J. Muller., "On the systematic use of budget-based design." *Proceedings of 16th annual international symposium of the INCOSE*, 2005a.
- Freriks, H.J.M., "White paper on designing with stepper motors." 2005b. Online: <http://www.esi.nl>.
- Heemels, W.P.M.H., E. v.d. Waal, G.J. Muller., "A multi-disciplinary and model-based design methodology for high-tech systems." *Proceedings of CSER*, 2006.
- INCOSE Technical Board. "Systems engineering handbook. A “what to” guide for all se practitioners." 2004.
- Jazayeri, M., A. Ran, F. vd Linden., "Software architecture for product families." Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2000.
- Malan, R., D. Bredemeyer., "Software architecture action guide." 2005, Online:

<http://www.bredemeyer.com>.

- Muller, G.J.' "CAFRCR: A multi-view method for embedded systems architecting; balancing genericity and specificity." Ph.D. thesis, Delft university of technology, 2004.
- Putten, P.H.A. van der, J.P.M. Voeten. "Specification of reactive hardware/software systems - The method software/hardware engineering." Ph.D. thesis, Eindhoven University of Technology, Eindhoven, 1997.
- ReVelle, J.B. "The QFD handbook." Wiley, 1998.
- Wieringa, R., "Requirements engineering: problem analysis and solution specification." *ICWE*, pp: 13-16, 2004.

Acknowledgement

We thank the Boderc team, and in particular Paul van den Bosch, Maarten Steinbuch, Lou Somers, Roelof Hamberg, Hennie Freriks, Berry van der Wijst, Anget Mestrom, Oana Florescu and Eric Gaal for their contributions and stimulating discussion.

Biography

Heico Sandee is currently a Ph.D. student in the Control Systems group at the department of Electrical Engineering of the Eindhoven University of Technology (TU/e). He received his M.Sc. degree in Electrical Engineering from the TU/e, in 2002. In 2005 he visited for three months the Mechanical Systems Control Laboratory at UC Berkeley, California. His main research interest is the multi-disciplinary design of embedded dynamical systems, with real-time control applications as the main focus.

Maurice Heemels received his M.Sc. degree (with honors) in mathematics and the Ph.D. degree (cum laude) in hybrid systems theory of the TU/e, The Netherlands in 1995 and 1999, respectively. From 2000 until 2004 he has been working as an assistant professor in the control systems group (Electrical Engineering, TU/e). In June 2004 he moved to the Embedded Systems Institute. He spent three months as a visiting professor within the ETH in Zurich, Switzerland in 2001 and a same period within Océ Technologies in Venlo, The Netherlands in 2004. His research interest include modeling, analysis and control of hybrid systems and their application to industrial design problems for high-tech systems.

Peter van den Bosch received his M.Sc. degree in Electrical Engineering from the TU/e, in 2001. Since 2002, he is a researcher at the research department of Océ Technologies BV. Since 2003, he is working on the Boderc project at the Embedded Systems Institute in Eindhoven.

Gerrit Muller received his Master's degree in Physics from the University of Amsterdam in 1979. He worked from 1980 until 1997 at Philips Medical Systems as system architect. From 1997 to 1999 he was manager System Engineering at ASML. From 1999 - 2002 he worked at Philips Research. Since 2003 he is working as senior research fellow at ESI (Embedded Systems Institute). In June 2004 he received his doctorate. The main focus of his work at ESI is to work on System Architecture methods and to enable education of new System Architects.
<http://www.extra.research.philips.com/natlab/sysarch/>

Marcel Verhoef works as a consultant for Chess, Haarlem, The Netherlands. He represents Chess in the Boderc research project at the Embedded Systems Institute. He currently holds a PhD position at the Radboud Universiteit Nijmegen, Institute for Computer and Information Sciences. He holds an MSc from Technical University of Delft (1993).