A Decision Model for Managing and Communicating Resource Restrictions in Embedded Systems Design

Bernhard Turban[1], Christian Wolff[2], Athanassios Tsakpinis[3], Markus Kucera[3]

[1]*Electronic Systems Engineering,*
MBtech Group, Neutraubling, Germany
(former activity of Micron Electronic Devices AG)
info@mbtech-group.com

[2] *Institute for Media, Information and Cultural Studies*
Regensburg University Germany
christian.wolff@sprachlit.uni-regensburg.de

[3] *Competence Center for SE,*
University of Applied Sciences, Regensburg, Germany
{markus.kucera,athanassios.tsakpinis}@informatik.fh-regensburg.de

Abstract

Requirements and requirement traceability play a key role in ensuring that embedded systems meet their goals. This article deals with improving requirements and requirement traceability to design artifacts in the context of embedded design. We identify a way for capturing decisions concerning limited resources in embedded systems. This approach directly integrates decision-related information with other traceability information gathered during requirements engineering. It relies on the concept of quantifiable budgets for structuring the decision information (decision model). Upon this model the prototype of a requirements traceability management tool has been developed reduces redundancies and inconsistencies in requirements management and lays the ground for improved collaboration and sharing of project knowledge between project members.

1. Introduction

Due to the rising complexity (cf. Chapter 2), developing reliable Embedded Control Units (ECUs) increasingly becomes a matter of good processes ([1]) and efficient communication between all partners involved. Requirements and requirement specifications are one of the most important means of communication between project partners. Tracing Requirements to design tries to ensure that design decisions (and thus the code) do not deviate from decisions agreed upon in former design stages. Traceability means making explicit the influence of requirements on all artefacts and processes in a project as well as tracing (for details cf. [2], [29]).

Tracing requirements from requirement specifications to their final fulfilling designs is especially difficult to establish, because this transition is a complex multi-step transformation

process from problem (requirements) to solution (design), where different modes of expression (e.g. text, diagrammatic specification languages like the unified modeling language (UML), program code) with different levels of formality are involved ([3]). Design decisions involved in this transition add to the complexity of this transition process (cf. [4], [5] for details). However, capturing decisions and their underlying rationale is difficult ([3], [6]), as capturing methods interfere with the overall design process or interrupt designers in their way of thinking ([3], [6], [4]).

Keeping this in mind, a key goal of our decision model approach is to lower the barriers of making design decision explicit as much as possible: Therefore, our decision model mechanisms offer designers a simple and semiformal model as a skeletal structure to easily add basic information[1] . In the context of embedded software development for automotive applications, we have developed a prototype tool environment which allows designers and developers to add more detailed information on decisions. In [4] we presented support model for decision documentation. The process helps to relieve the know-how/rationale bearers from documentation work by shifting the effort to the rationale seekers[2], thus also helping inexperienced rationale seekers tightening their acquired knowledge.

In Chapter 2 we give a short analysis of current challenges concerning development of software-based ECUs in the automotive domain. The Automotive domain is just a typical example, but similar problems are also known for other domains. Chapter 3 deals with general problems of establishing traceability information at the transition from requirements to design phases and we briefly describe results of preliminary research ([4], [5]) important for the here introduced decision model. In Chapter 4, an additional type of decision is discussed: Making decisions about limited resources and their allocation in a design project which is a key issue for embedded applications that are extremely resource-aware (e.g. memory, bandwidth or computation resource usage). We use the concept of (resource) budgets to capture the basic rationale behind a resource-oriented design decision. An accompanying case study shows the practical relevance of this model. We assume that in comparison to other ways of modeling resource allocations, this decision model improves communication and collaboration between project members and allows for a simpler crossing of organizational boundaries in multi-party projects (ch. 5).

2. Challenges in Automotive System and Software Engineering

The proportion of software (SW) and SW-based embedded systems in everyday products increases exponentially ([7]) and at this increase is accompanied by a growth of development complexity. The following characteristics illustrate this for the automotive domain (cf. [7], [8], [9]) – similar problems can easily be found in other embedded domains:

1. Increasing cross-linking of vehicle functional features leads to increasing cross-linking of ECUs[3]. Such features are typically realized by a

[1] In [4] we have elaborated on the idea of providing a light weight formal structure for documenting design decisions. This basic information can be enriched with further information (e.g. detailed description) on demand. Basically, we follow the principle of putting "Put as little extra burden as possible on the bearer of rationale"([6]), avoiding typical usage barriers (cf. [3]) encountered when *Rationale Management* (i.e., the documenting of decisions) is used in practice.

[2] A person interested in the backgrounds of a decision at later phases of the project.

[3] A typical scenario might look like this: A car crash triggers crash sensors which activated several airbag ECUs and a crash management ECU (CM-ECU). The CM-ECU sends an „Unlock_Doors" signal to all door ECUs, requests the position from the Global Positioning System-ECU and sends an automatic emergency call via a Universal Mobile Telecommunications System-ECU to local rescue organizations.

collaboration of several ECUs, leading to higher interdependencies between ECUs.

2. The development of a strongly cross-linked car system can only take place in collaboration with the car manufacturers (Original Equipment Manufacturers (OEM)) and heterogeneous chains of suppliers.
3. Many functions are safety-critical and require significant additional modeling effort for safeguards.
4. Many ECUs have strict or at least weak timing restrictions[4].
5. Considerably higher unit quantities in comparison with the aviation domain raise the pressure on unit cost minimization. This leads to extremely cost-optimized hardware with strict restrictions concerning resources and timing.

Items one and two above imply that frictionless information exchange between all project members is a critical success factor and requirement documents are the cornerstones of this collaboration, since they are the central interfaces between organizational units of a project. In addition, points one and two may even urge partners to employ compatible development processes. A good step towards this goal are process standards and maturity models like SPICE (Software Process Improvement and Capability dEtermination, [10]), its new domain specific adoption Automotive SPICE (cf. [10], [11]), or CMMi (Capability Maturity Model Integration, cf. [12]). Point three means that additional safety mechanisms in ECUs (e.g. Fail Safe Modes, hardware (HW) and SW redundancies) and increasing complexity put additional stress on the quality of development processes ([1]). The criticality of handling timing-related issues (4.) increases with a rising number of cross-linked ECUs (1.) and their additional collaboration needs and timing information needs to be communicated throughout organizational boundaries of all involved ECUs. Finally, point five is in potential conflict with all preceding points and is particularly challenging for subcontractors.

3. Traceability problems in the transition from requirements to design

As previously stated in more detail ([4], [5]), we assume that for establishing valuable traceability between requirements and design models, the following two major problems exist:

- Requirements evolve at all levels of different requirements engineering and design processes (E.g. Hatley et al. speak of „requirements derived from design decisions“ [13; S.37]).
- Tracing requirements from the original requirements specification to design by simple bidirectional links is inaccurate as this would assume the transition from requirements to design to be a fairly linear and one dimensional process. We rather believe that this transition is more of a creative and complex mental transfer process performed by designers when gradually transforming the problem space into a solution space.

Current approaches try to cope with this problem by using a layered process model (cf. [14]), where several requirement specification artefacts and design artefacts

[4] Mostly, not all timing restrictions of hard real time systems are strict. Some functions may also have weaker or even no timing restrictions.

mutually alternate during each design cycle. For example, Systems Engineering processes demand a system requirement specification and result in a systems design. Both artefacts form the basis for the SW requirement specification leading to the SW design. We believe that strict obedience to this artefact model results in a high degree of redundant information causing consistency problems(see [4] for further details). A second drawback is the observation that in the course of the project requirements stemming from the customer and those derived from design activities tend to be mixed up leading to a gradual unintentional overlap between requirements specifications and design artefacts. To avoid these problems we propose a strict distinction between requirements issued by the customer and those derived from design activities, leading to the following taxonomy (see Fig.1):

- A *Requirement* (*REQ*) defines requirements directly stated by the customer.
- A *DesignConstraint* (*DC*) is a requirement resulting from design decisions.
- The hyperonym *RequirementalItem* (*RI*) defines conjoint characteristics and behaviour of both types of requirements.

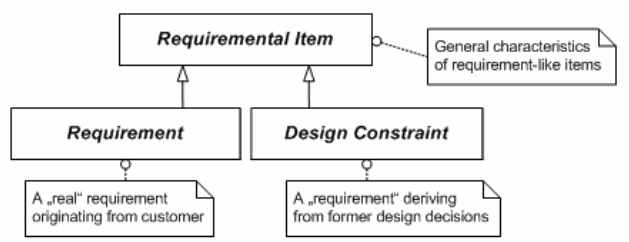

Figure 1: A taxonomy of *requiremental items* ([4]).

4. Resource Allocation as a Special Decision Making Case

In design activities for embedded systems an additional decision type can be identified dealing with non-functional aspects of limited resources such as memory resources (e.g. Read Only Memory (ROM), Random Access Memory (RAM), Electrically Erasable Programmable Read Only Memory (EEPROM)) or timing restrictions. A core goal of embedded design is the effective administration and distribution of such resources and different strategies for handling this problem exist:

0. The allocation is a more or less unconscious or uncontrolled process (i. e., no explicit strategy is established).
1. A resource estimation is performed as part of the design and estimations are checked and adapted at each development cycle.
2. Resource allocation is explicitly modeled in the design model (e.g. by using UML profiles such as the UML Profile for Schedulability, Performance, and Timing profiles [15, ch. 4] or MARTE ([16])).

With respect to collaboration in complex development teams or organizations is considered, approaches 1 and 2 have limitations in the following aspects:

- Propagation and communication of changes to all team members involved in the change can be cumbersome.

- Minimizing redundancies as a major source of inconsistencies can result in communication errors.
- The seamless adoption and refinement of other designers' design results can be extremely difficult.
- Sharing project knowledge in general will become more difficult.

The following example illustrates these shortcomings in more detail.

4.1. Example Scenario for Resource Allocation Design Decision Making

Suppose we have the following example use case (*Fig.2*) for a lights steering device in an automotive context: The system retrieves different signals from the Controller Area Network (CAN) bus. The lights steering task determines whether some lights must be activated or deactivated. The lights are steered via Pulse-width modulation (PWM) and diagnostic information is retrieved via analog feedback.

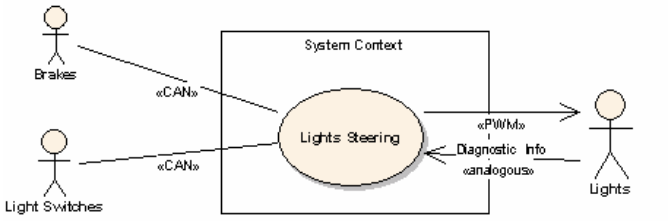

Figure 2: Example use case of the case study.

The corresponding ECU's SW design is shown in (Fig. 3).

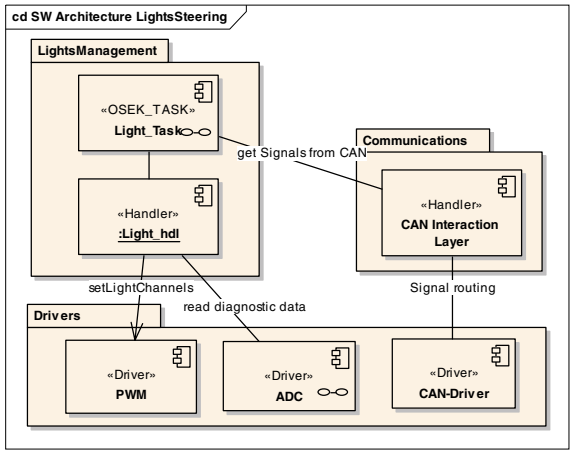

Figure 3: Example SW design for use case of Fig.2.

A high level SW architect has partitioned the SW into three subsystems (packages *LightsManagement*, *Communications*, and *Drivers*). For each subsystem a subsystem designer determines their subcomponents, illustrating design collaboration. This separation of several layers of design responsibility is common for larger projects with complicated application domains. The following project decisions have been made:

- The lights management contains an active process *Light_Task* with a complex state machine. An underlying light handler *Light_hdl* knows how to manage the underlying drivers according to the light signals to set. Both components are being developped in-house.
- The drivers (*PWM*, *ADC* and *CAN*) are supplied by different subcontractors. Code size, performance and other parameters are highly dependent on their individual configuration. Therefore, a subcontractor manager shall monitor each driver for these parameters.
- The *CAN Interaction Layer* depends on the types of signals relevant for the device. These settings are defined by the customer (OEM), because it affects communication.

This leads to the following RAM consumption estimation laid down as a separate chapter in the design document of the high level designer:

Module	**Light_Task**	**Light_hdl**	**CIL**	**CAN-Drv**	**PWM**	**ADC**	**Buffer**
RAM (1500 Byte available)	600 Byte	250 Byte	100 Byte	300 Byte	100 Byte	100 Byte	50 Byte

Table 1: RAM - resource estimation

Such tables are a common format for documenting resource assignments in design documents (cf. [17]). The tabular format has the main advantage that it easily gives an overview, but it has important weaknesses when collaborative aspects are considered:

- First of all, even though these assignments are typically called estimations, they should rather be treated as *RIs*. This implies that a mechanism must be in place to communicate these *RIs* on time to all interested stakeholders - especially if changes occur during project progress.
- Further, the allocation settings are estimated at a certain design stage and thus are an integral part of the design documents at this stage. Therefore, further processing of this information by other designers is difficult. In our case study the estimations are made at the level of modules and included into the documentation of the high-level design. If the module designer of the complex *Light_Task* wants to refine the resource estimation into a more detailed estimation, a problem arises. In this case he would have to copy the information "*Light_Task == 600 Bytes*" into some document of his responsibility. This leads to unnecessary redundancy causing consistency problems, when this setting changes later in the project.
- These problems are even more critical, if some parts of the project are delivered by a subcontractor – as it happens to be the case in our example. In this case, all relevant requirements for the item to supply must be provided (as required by SPICE process ACQ.4 Supplier Monitoring, see [10]). In our case, the RAM estimations, since they are *RIs*, must be communicated as requirements to the supplier. This also leads to a high degree of redundancy with even worse effects, if changes are not communicated.

4.4 *Budgeted Resource Constraints* (*BRC*) as requiremental items

In consideration of this problem a way to perform such resource allocation decisions in a handy fashion is needed which also allows for the communication of the results for each considered design element throughout the entire project in an efficient way. An additional aspect here is the fact that the results of a decision act as new *RIs* on the design elements they are assigned to. As the literature shows (cf. [18], [19], [20], [15; p.317], [21; p. 124], [17], [22]), most resource allocation activities consist of numerically truncating a larger resource amount into smaller subsets – more or less in analogy with the abstraction hierarchy of a system's/software's design (see ch. 5.4 and Fig. 6 / 7 below). Obviously, this can be compared to the process of preparing and distributing budgets in business administration or project management area ([23]). Therefore, we propose to enhance our taxonomy of requiremental items by an additional type of *RI* called *Budgeted Resource Constraint* (BRC, see fig. 4):

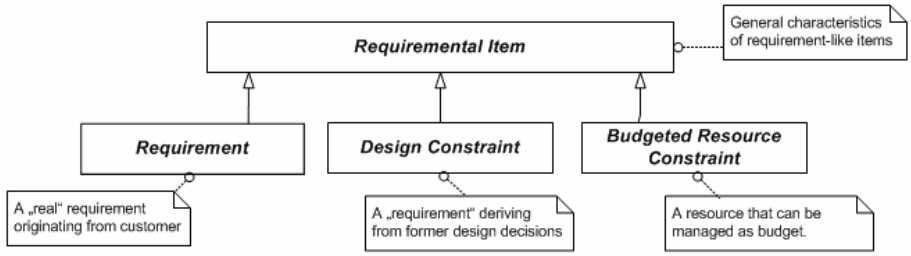

Figure 4: Types of *Requiremental Items* with *Budgeted Resource Constraints.*

BRCs are similar to *Design Constraints* (*DC*s) as they represent the results of a decision making process and can be assigned as *RIs* to any design element. However, there are the following differences when compared with other *RIs* (such as *DCs*):

- *BRCs* represent numerical values whose associated design elements may not exceed the maximum value of the assigned *BRC*.
- A *BRC* can be subdivided into sub *BRCs*. Thus *BRCs* at the same time represent a decision making process as well as its results.
- As *BRCs* represent numerical values, whose sub *BRCs* divide resource amounts into smaller budgets for more detailed parts of the design, automatic consistency checks (e.g. tests for budget overruns) can avoid wrong allocations. Budget overruns may be detected at an early project stage.
- Individual *BRCs* can be added to one design item only, whereas requirements and design constraints may be added to several items.

Resuming the example described above *fig. 5* illustrates the resource allocation problem presented using *BRCs* as implemented in our tool environment. The connections to the design elements illustrate so called *satisfy-links* as used in traceability models (cf. [24]) to indicate, that a design element must satisfy the given "requirement" (in our terminology: *requiremental item* (*RI*)). In this situation, the SW architecture is assigned to fit in a total budget of 1500 bytes of RAM. This *BRC* is subdivided into six sub *BRCs* assigned to the six modules in the SW architecture, thus showing a more detailed partitioning of the RAM budget.

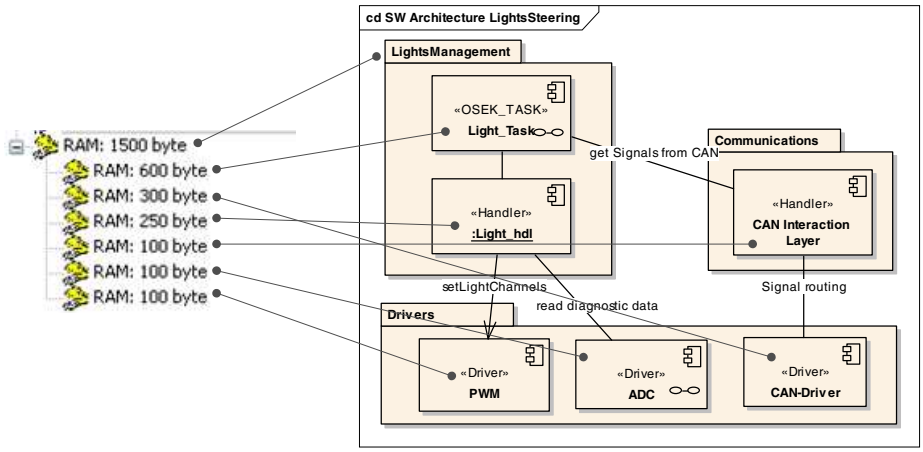

Figure 5: Resource Allocation Example with *Budgeted Resource Constraints.*

Comparing fig. 5 with table 1, we see that both representations have an equivalent meaning. In fact, the idea of budgets in HW and SW engineering is not new (cf. [20], [15; p.317], [21; p. 124], [17], [22]). What we want to point out beyond the appealing (and well-known) aspect of a more or less easy mathematical model enabling consistency checks are the advantages of the budget concept itself, when it comes to *collaboration* and *sharing project knowledge* between project members. In this sense we use the budget concept as a *means of communication* during software design. The following chapters will provide more details on this.

5. BRCs - Advantages for Collaboration and Sharing Project Knowledge

The following situations of our example project show the value of BRC for the following communication situations:

- Within-project refinement,
- communicating information over organizational boundaries,
- change management, and
- different views on the same problem.

5.1 Within project refinement

During the first design cycle of the "*Light_hdl*" (*LH*), the *LH* is forecast to have a very tight RAM budget. Therefore the designer identifies several specific aspects for which he arranges budgets according to his current information and needs (see fig. 6 below):

- In normal mode, the module uses the settings in EEPROM mirrored to RAM for steering the lights. RAM consumption depends on the number of steered channels and the number of bytes needed for each channel.
- The diagnostic part supervises regular checks of the electrical current between the ECU and the connected lights to detect malfunctions as short circuit or open drain. Malfunctions lead to the deactivation of a light channel.

- In the case of severe error conditions, e.g. loss of EEPROM data, the fail over mode assures that at least essential functions like brake lights and indicators work. The code and configurations are fixed in ROM, thus no particular portion of RAM is needed.

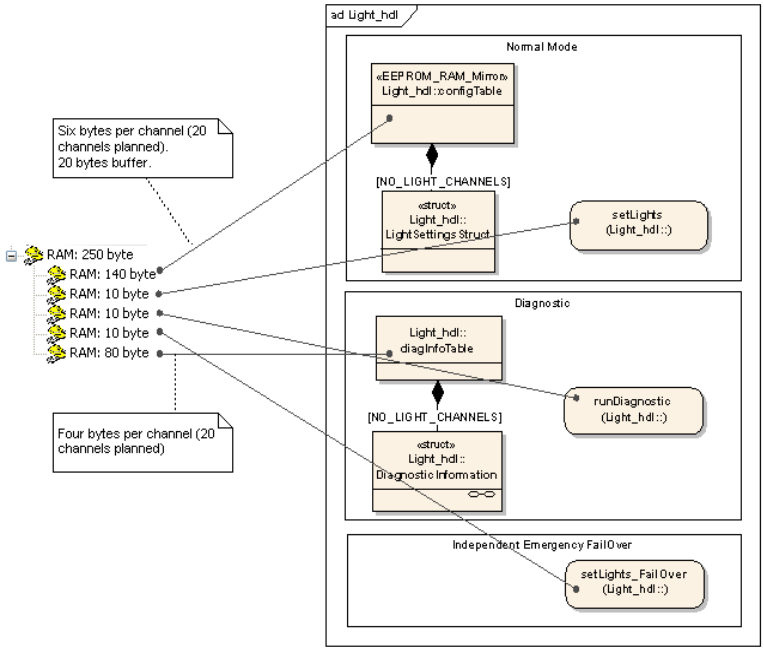

Figure 6: Sub budgeting of the "Light_hdl" module

With our proposed type of *BRCs*, designers of sub levels can directly continue to process results produced in previous design decision processes.

5.2 Communicating Information across Organizational Boundaries

Information must often be provided across organizational boundaries. Such boundaries can be subprojects within the same company or between different companies.

In our case study, drivers are provided by different subcontractors. This implies that all requirements for the drivers must be provided throughout all parties involved. In our experience, functional aspects are communicated in a quite complete fashion, but such non-functional aspects (e.g. restrictions on memory, timing, etc.) resulting from former design decisions are often forgotten.

The solution described here supports exporting all types of *RIs* associated with a design element as a new requirement specification into requirements management tools like DOORS® that can be delivered to the subcontractor. Since our *BRCs* are treated as normal *RIs*, they are directly propagated to the subcontractors via automatically generated requirement specifications. In later development phases, these requirement specifications can be continuously synchronized with the settings in the design element, thus ensuring proper propagation of requirements to subcontractors.

5.3 Change Management

During project progress changes occur that force designers to change decisions and assumptions. Managing those changes efficiently is essential to avoid project deviations. Two important heuristics should be considered:

- Changes should be kept as local as possible to avoid unnecessary complexity.
- Changes must be implemented in a consistent way.

Our model supports handling changes of *BRCs* as local as possible. Continuing our example, it might happen that the "*runDiagnostic*" function needs more than 10 bytes of RAM (see *fig. 6* above). In this case, the designer can first try to find an internal solution of the problem (e.g. find a way to cut down on some bytes in the "*diagInfoTable*"). If this is not possible, the designer can escalate the problem to a more high-level designer.

In another situation, new requirements from the customer could make the creation of a new, additional module necessary. This case has effects on the design as a whole, since most of the modules already present might suffer a budget cut in their *BRCs* as a consequence. Our tool visualizes changed *BRCs* (in red color) to alert designers of sub-layers to analyze the impacts on their assignments.

If the sub designer has made his changes and consistency checks (e.g. detecting budget overruns) pass, the designer can mark the change as implemented. After this, the *BRC* is shown in normal mode.

5.4 Different Views on the same Problem

In software design theory the idea that different aspects of SW can be modeled by different views has been proposed (cf. [25]). The same can be claimed for non-functional aspects modeled by *BRCs*.

Besides the direct allocation view (see *fig. 5* and *6* above) our tool supports creating an enhanced table representation. *Fig. 7* shows this tabular lineup between *BRCs* and their allocated design elements. Both columns additionally show their hierarchical break down.

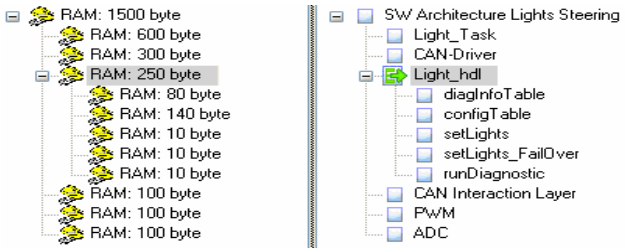

Figure 7: Tabular View with corresponding abstraction hierarchies.

Since the structure of the *BRCs* break down has a strong analogy with the break down of their associated design elements, design flaws of the assignment be can easily detected. *Fig. 8* shows this situation, where a wrongly associated item disturbs the analogy, helping the designers to detect those problems easily.

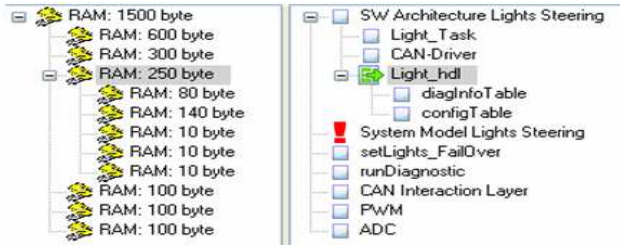

Figure 8: Tabular View with assignment inconsistency (exclamation mark).

6. Summary

We have argued that in embedded systems design decision situations exist that are not easily covered by our original decision model (cf. [4]). However, analyzing the structure of these kinds of decision problems and at the same time directing their decision model towards their structure can improve support for designers to:

- find and document design rationale as easily as possible and
- reuse this rationale in further decision processes just as easily

We suggest the budget concept for dealing with limited quantifiable resources in order to create a specialized decision model. Thus, a semiformal decision model is introduced that can be handled easily and places little additional burden on rationale bearers. Our practical experience shows that it further improves the internal and external communication within project teams significantly and supports the propagation of information to other involved parties.

Customer workshops at the former Micron Electronic Devices AG (now with the MBtech Group) have shown that the proposed models and their semantics are easily understood by designers. Designers express their need for such decision support, referring to examples where changes in resource estimations were not communicated properly in the course of the project which led to resource-related problems in later development phases.

Future research will concentrate on finding additional decision models for improving design-time requirement traceability and connecting information. A key issue, though, is to make sure that these mechanisms are compatible with each other and to allow pragmatic usage.

Acknowledgements

This research has been funded by the Bavarian Ministry of Economic Development (Grant Nr. IUK229). Furthermore, we want to thank all partners that contributed to our research.

7. References

[1] O. Benediktsson, R. Hunter and A.D. McGettrick, “Processes for Software in Safety Critical Systems”, Software Process: Improvement and Practice, volume 6 issue 1, 2001, pp. 47-62.

[2] F.A.C. Pinheiro, “Requirements traceability”, IN: S.d. Prado Leite, J. Cesar, Doorn and J. Horacio, “Perspectives on Software Requirements”, Kluwer Academic Publishers pp. 91-113, 2004.

[3] J. Horner and M.E. Atwood, “Effective Design Rationale: Understanding the Barriers”, In [26], 2006, pp.72-90.

[4] B. Turban, M. Kucera, A. Tsakpinis and Ch. Wolff, “An Integrated Decision Model For Efficient Requirement Traceability in SPICE Compliant Development”, Fifth Workshop on Intelligent Solutions in Embedded Systems (WISES), Madrid, 2007.

[5] B. Turban, A., Tsakpinis and Ch. Wolff, “Ein Entscheidungsmodell für das Tracing von Anforderungen”, In: W.-G. Bleek, J. Raasch and H. Züllighoven (Eds.), “Software Engineering 2007”, Fachtagung des GI-Fachbereichs Softwaretechnik, 2007, pp. 265-266.

[6] K. Schneider, “Rationale as a By-Product”, In [27], 2006, pp. 91-109.

[7] M. Broy, “Challenges in Automotive Software Engineering”, Proceeding of the 28th international conference on Software engineering, Shanghai, 2006, pp. 33 - 42.

[8] K. Grimm, “Software-Technologie im Automobil”, in [28], 2005, pp.407 –430.

[9] M. Kucera and H. Mauser “Semi-Automatic Reliability Assessment of Safety Related Embedded Systems”, Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems, November 13-15, Dallas, USA, 2006.

[10] K. Hörmann, L. Dittmann, B. Hindel and M. Müller, “SPICE in der Praxis, Interpretationshilfe für Anwender und Assessoren”, dpunkt Verlag, Heidelberg, 2006.

[11] M. Müller, K. Hörmann, L. Dittmann and J. Zimmer, “Automotive SPICE in der Praxis: Interpretationshilfe für Anwender und Assessoren.” Dpunkt 1. Auflage, Heidelberg, 2007.

[12] R. Kneuper, “CMMI. Verbesserung von Softwareprozessen mit Capability Maturity Model Integration”, dpunkt Verlag, Heidelberg, 2006.

[13] D. Hatley, P. Hruschka and I. Pirbhai, “Komplexe Software-Systeme beherrschen”, MITP, 1. Auflage, Bonn, 2003.

[14] K. Pohl and E. Sikora, “Requirements Engineering für eingebettete Systeme”, In [28], pp.101 –140, 2005.

[15] B.P. Douglass, “Real Time UML Third Edition” Pearson Education, Boston, 2004.

[16] H. Espinoza, H. Dubois, S. Gérard, J. Medina, D. Petriu and M. Woodside, “Annotating UML Models with Non-Functional Properties for quantitative analysis”, In: Lecture Notes in Computer Science; Volume 3844 Heidelberg, 2006, pp. 79-90.

[17] G. Muller, “CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity”, Doctors Thesis Technische Universiteit Delft, 2004; http://citeseer.ist.psu.edu/muller04cafcr.html (Access 12/2007).

[18] E. Bozorgzadeh, S. Ghiasi, A. Takahashi and M. Sarrafzadeh: “Incremental Timing Budget Management in Programmable Systems”, International Conference on Embedded and Reconfigurable Systems and Architecture, pp.240-246, July 2004.

[19] C. Chen, E. Bozorgzadeh, A. Srivastava and M. Sarrafzadeh, “Budget Management with Applications”, Algorithmica 34, 2002, pp. 261-275.

[20] W. Fornarciary, P. Gubian, D. Sciuto and Ch. Silvano, “Power estimation of embedded systems: A Hardware/Software codesign approach”, In: Micheli, G; Ernst, R.: Readings in Hardware/Software Co-Design. Morgan Kaufmann 2001; pp. 249-258.

[21] B.P. Douglass, “Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems”, Pearson Education, Boston, 2003.

[22] V. Gullapalli, “Best practices improve hierarchical design constraints”, Tech Online Nov 2003; http://www.eetimes.com/news/design/showArticle.jhtml;jsessionid=O4YQ3DC1YPCQKQSNDLRSKHSCJUNN2JVN?articleID=16502497&printable=true (Access 12/2007).

[23] H. Heiser, H. Holzer and W. Sommer: “Budgetierung. Grundsätze und Praxis der betriebswirtschaftlichen Planung”, De Gruyter & Co., Berlin, 1964.

[24] B. Ramesh and M. Jarke, “Toward Reference Models for Requirements Traceability”, IEEE Transactions on Software Engineering, Vol. 27, No. 1, 2001.

[25] P.Kruchten, “Architectural Blueprints – The “4+1” View Model of Software Architecture”, IEEE SW 12 (6), Nov 1995, pp.42-50.

[26] B. Paech, A.v. Knethen, J. Doerr, J. Bayer, J., D. Kerkow, R. Kolb, A. Trendowicz, T. Punter, and A. Dutoit “An Experience-Based Approach for Integrating Architecture and Requirements Engineering”. International Conference on SE, Portland, 2003.

[27] A. Dutoit, R. McCall, I. Mistrik and B. Paech (Eds.), “Rationale Management in Software Engineering” Springer, Berlin, 2006.

[28] P. Liggesmeyer and D. Rombach (Eds.), “Software Engineering eingebetteter Systeme Grundlagen - Methodik – Anwendungen”, Elsevier, 1. Auflage, München, 2005.

[29] Ch. Rupp and Sophist Group “Requirements-Engineering und –Management”, Hanser, München, 2002.