Using object scenarios for requirements analysis - an experience report

Albert Zündorf[1], Jürgen Leohold[2], Dieter Müller[3],
Ralf Gemmerich[1], Carsten Reckord[1], Christian Schneider[1], Sven Semmelrodt[4]

[1] University Kassel
[2] Volkswagen AG
[3] VW Bordnetze GmbH
[4] Siemens VDO Automotive AG

Abstract: This paper is an experience report applying object diagrams for requirements analysis in an industrial project in the automotive industry. The considered project has created a tool for the design of car electronic systems. This project involved an enormous amount of domain knowledge. The challenge was to involve the domain experts in the analysis, design, and implementation activities such that the transfer of domain knowledge is fostered. This paper reports on our approach to achieve this involvement and what we achieved.

1 Introduction

Modern software development processes, as e.g. the Rational Unified Process [JBR99], use textual use case descriptions for requirements elicitation. During further requirements analysis, analysis classes are identified and one may e.g. use collaboration diagrams or sequence diagrams in order to outline use case behavior. This is then further refined into design and implementation classes and into actual behavior implementation. This process may be executed iteratively, addressing small chunks of functionality one after the other.

Due to our experiences, such a process requires decent skills in abstraction, object orientation, analysis, design, design patterns, etc. Frequently, the result of an object oriented analysis or design phase will be documented and discussed using architecture diagrams or more specifically class diagrams. Unfortunately, class diagrams are quite abstract and their interpretation and the judgement of the represented design decisions requires special skills and training. While many computer scientists and professional software developers have these skills as part of their every day professional work, a project frequently involves a large number of stakeholders with a different background. The project may involve business managers, administrative experts, domain experts and potential system users. These people may have quite different backgrounds and skills. These different skills and the corresponding domain knowledge are of crucial importance for the success of the overall project. Unfortunately, many of these important people will have difficulties to interpret the information and design decisions contained in architecture and class diagrams. Thus,

the domain experts are frequently not able to communicate their knowledge with the software experts, appropriately.

In order to overcome these communication problems, the Fujaba Process proposes to use object diagrams or object scenarios as an intermediate representation or analysis means during analysis, design and implementation, cf. [DGZ05, KNNZ00]. Usual collaboration or sequence diagrams stay on a level of abstraction where certain components exchange certain messages in order to exemplify the internal process that realizes the considered use case, cf. [JBR99, BRJ99]. Our object scenarios usually describe the internal process in much more detail. We employ concrete objects with concrete attribute values and concrete links or pointers between each other. Thereby, we pin-point the representation of certain domain information in our implementation model quite early in the development process.

In the reported project, it turned out, that the design and implementation decisions outlined in object scenarios were still easily recognized by our domain experts. Pinning down certain design and implementation aspects quite early then fertilized the further refinement and elicitation of related functional requirements, considerably. In multiple iterations, our domain experts used the refined designs to enhance their textual requirements descriptions until it became easy to implement them. Since our implementation language, so-called Fujaba story diagrams, uses a graphical notation which is very close to our object scenario notation, our domain experts were even able to review our implementation. This enabled very valuable feedback from our domain experts to the software developers. For the goals of the reported project cf. [GSZR et al. 05].

The following section introduces the application domain and the software development project that serves as example in this paper. Section 3 shows some object diagrams that have been developed by the electrical engineering experts participating in this project and how we exploited these object diagrams to develop the desired cost estimation and optimization tool. We will conclude with lessons learned.

2 The OBA application

The considered example project is called OBA for "Optimization of car electric system architectures" (in German: BordnetzArchitekturen). Our task was to develop a tool that enables a car manufacturer to optimize the costs of the electronic system within a car. The considered electronic system of a car includes actors (e.g. lights, motors, ...), sensors (e.g. on/off switches, temperature sensors, rotation sensors, ...), electronic control units (ECUs), fuses, and all the wires connecting these components. The costs of the electronic system of a car consist of the costs for the parts plus the costs for assembling them within the car plus the costs of extra material as e.g. cable ties or screws, etc.

Prerequisites for the development of a car electronic system are usually the employed sensors and actors including their placement within the car. In addition, the geometry of the car body is usually already defined including possible spaces for the placement of ECUs and including all channels where cables may be laid and mounted. The main architectural decisions for a car electric system are how many ECUs it employs, where

these ECUs are placed in the car and which control functionality is placed on which ECU. Similarly, the number and placement of fuse boxes has to be chosen. Depending on these decisions, the wiring of the car has to be routed to connect the ECUs with each other and with the sensors and actors they need to access and to provide the power supply to all components. Our task was to develop tool support for the developers of car electronic systems enabling them to study alternative architectures with minimal effort and helping them to optimize the electronic system with respect to overall costs and e.g. weight.

As a prerequisite for the optimization of a car electric system, we first needed to develop a cost model that allows us to compute the overall costs of a chosen electric system and to analyze the gain or loss caused by an alternative architecture. This cost model requires an enormous amount of domain specific knowledge. This knowledge is provided by the industrial partners in this project, namely the Volkswagen AG, a car manufacturer or OEM, Siemens VDO, a supplier of ECUs and electronic components, and VW Bordnetze, a manufacturer for wiring harnesses. The OBA project requires not only an intensive involvement of domain experts but also the exploitation of a large number of heterogenous sources of data. First of all, the car body geometry is provided by a CAD system owned by the OEM. We need the geometry information in order to calculate cable lengths and to check space restrictions. In addition, we need the list of electronic components like actors and sensors We also need a higher level description of software modules that are going to run the car, e.g. the window lift module, the locking control module, the light control module, etc. For each module, we need to know which sensors and actors it accesses, with which other modules it communicates and in which ECU it will be located. Next we need a catalogue of wire kinds together with rules for the determination of an appropriate wire kind for a given connection. This depends on the current that is to be transported as well as on temperature and environmental conditions in the crossed car areas.

3   The OBA cost model

Since we employ complex data structures for the optimization of the wiring harness and for the conception of ECUs, and due to our experiences, we decided to develop the OBA tool with an object oriented model using the Fujaba environment [Fu02] and generate the actual implementation in Java.

The modelling of the OBA application started with the car body. On top of this we have built the wiring model and the models for the ECUs and for the software functions. This is the basis for the optimization algorithms. In the OBA project one major source of complexity was the sheer amount of aspects that had to be addressed. For example the costs for the wiring harness included aspects like:

- Cable costs depending on cable length and cable kind.
- Costs of cable mounting elements.
- Costs of wire protection means.
- Costs and kinds of connectors.
- Cables may be split at a certain point.

2.1 The car body
… The car body consists of different car areas representing units of assembly and units of similar environmental conditions. The changeovers between car areas require additional protection means against different environmental conditions, e.g. humidity. In addition, car area changeovers require special mounting efforts. …The car areas contain all kinds of mutually connected wire channels, cavities, etc. …

Figure 1: First textual requirements on car bodies (translated from German)

- The costs of mounting the wiring harness.
- So called threading costs (to be revisited).

This list is far from completeness. Covering ECUs, fuse boxes, communication busses etc. involved similar lists. One of the main challenges of the OBA project was to structure all of these aspects appropriately. This required a close collaboration between the domain experts and the software engineers. At the beginning of the development, our domain experts came up with textual requirements descriptions as shown in Figure 1. These requirements where of course quite coarse grain and thus insufficient for further analysis, design and implementation. In addition, most other requirements descriptions were based on the car body description, e.g. the description of spatial constraints for ECUs or for the diameter of cable bundles. The next layer of requirements was based on these two lower requirements layers and so on. Without a clear common understanding of the basic requirement layers, it became progressively harder to read and write new requirements on top of them.

To overcome these problems, we had to pin-point the requirement layers such that a common understanding between domain experts and software engineers was achieved and such that the next layer was easy to build on top of them. Thus we asked our domain experts to exemplify some typical situations with the help of object diagrams. After some training and some more discussion, we developed object diagrams as shown in Figure 2. Actually, the object diagram shown in Figure 2 has a large number of predecessors that were refined multiple times for each requirements layer. We started with object diagrams modelling car areas in order to deal with environmental conditions as hot and wet in the engine area or cold and dry in the cabin area, cf. objects `ca1` and `ca2` at the top of Figure 2. Then we added channels and nodes modelling the parts of the car body where wires may be routed, cf. objects `ch1` through `ch4` and node `n1`. In addition, the information on required wire protection means and required mounting means is attached to channels. Next we modelled ECUs and other electronic components, cf. `ecu1` and `comp1` and `comp2`. Based on this, we introduced wires, contacts and connectors. Finally, we introduced bundles of wires since some channels may contain multiple wire bundles where each bundle has its own costs for protection means and mounting means. Altogether this process resulted in a large number of detailed object diagrams exemplifying our modelling of the different aspects to be considered for our wiring harness cost model.
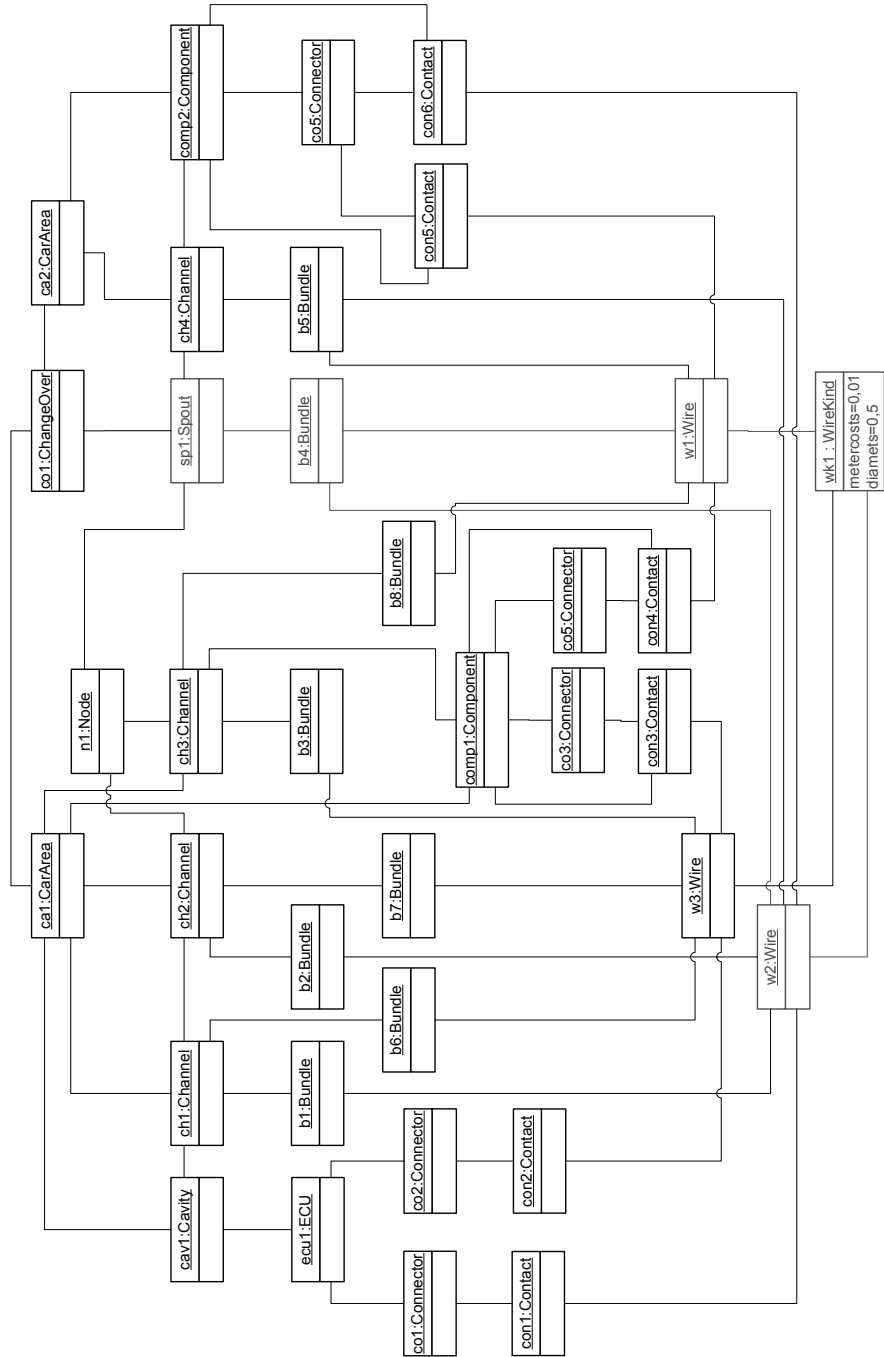
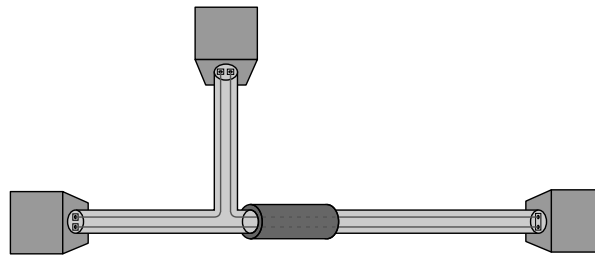Figure 2: An object diagram exemplifying the threading costs

Figure 3: Example for the threading of cable bundles

With the help of the object diagrams, we had a lot of fruitful discussions with our domain experts. For example, in a first attempt our domain experts proposed that each channel object has its own attributes for the humidity and for the maximum temperature. We recognized, that many channel objects had the same values for these attributes and asked the domain experts, when these values will differ. They told us, that these values are determined by the car area and they differ only when the car area is changed. Thus, we moved the humidity and temperature attributes from the channels to the containing car areas thereby reducing the data redundancy a lot.

Similarly, at the beginning, each wire had its own attributes for meter costs, diameter and wire type. After some while, we found out that only 7 different wire diameters are used in a car and that the costs per meter were the same for all wires with the same diameter and the same wire type. To improve this, we introduced a catalogue of wire kind objects holding the price information and each wire just refers to its wire kind. Thereby, it became easy to handle changing market prices for wires. Later on, we introduced such catalogue objects also for contact kinds, connector kinds, chip kinds, etc.

Another important point was the discussion of associations. We did not ask our domain experts whether class `CarArea` should have a `contains` association to class `Channel` or whether we should introduce a composite pattern at this place. Due to our experiences, it is not fruitful to discuss with non software domain experts on this level of abstraction. Instead, we asked our domain experts, whether there is an example of deeper nesting of car areas. Indeed, some car areas contain sub car areas. In addition, there were examples where car areas contained cavities which in turn contained channels. Thus, we decided to use a composite pattern to model this hierarchy and told our domain experts that arbitrary nesting of car areas, cavities, and channels are enabled, now.

In turn, the object diagrams provided the domain experts with a clear understanding how the software models the different aspects of the application domain. This facilitated the description of the requirements, considerably. As an example for a more complex requirement description developed this way, we may consider the computation of threading costs for cable bundles. Figure 3 shows a small example for a wiring harness connecting three components where the right component is reached via a car area changeover that is protected by a rubber spout. To mount the wiring harness, some cable bundle needs to be threaded through the spout. This is a very tedious work with high mounting costs.

Computing Threading Length and Threading Direction
Consider each wire passing a rubber spout and sum up all lengths of all used cable channels before the spout. The distance of spouts and cavities is computed via the lengths of the cable channels. Consider only cavities connected by wires threaded through the spout. Compare the maximal lengths for each side of the spout. The lower one is the required threading length. This also identifies the threading side, i.e the side from which the cable bundle is threaded through the spout. Consider also the number of plugs at the threading side.

Figure 4: First iteration requirement description (translated from German)

Originally, the computation of these threading costs was quite unclear and hard to explain, cf. Figure 4. This first textual requirements description was hard to implement since it was quite unclear, how one finds the wires on the two sides of a spout and how the length is computed. With the help of example object diagrams, it became clear, that one has to consider the bundles within the spout. In addition, it became clear that there are two cases to be considered. If the spout leads directly to a channel, one may just lookup the bundles in that channel. If the spout leads to a node, one has to consider all channels attached to the node. In the project, the object diagrams were just discussed with the software developers. In this exceptional case, we lazily did not write down the improved requirements description that resulted from the discussion. However, for this paper, we created an improved textual requirements description referring to the object diagram of Figure 2 in order to exemplify how the object diagrams helped to improve the requirements description, cf. Figure 5.

To summarize, with the help of the object diagrams the requirements became very detailed and concrete. Such detailed requirements accompanied with elaborated object diagram example were an excellent basis for the implementation of the desired functionality. Thus, the object diagrams enabled our domain experts to contribute very valuable aids to the design and implementation of the system.

Usually, architecture and class diagrams play an important role in software projects. Commonly, these diagrams are used for design discussions. In the OBA project, most of the design decisions have been done based on the discussion of object diagrams. At the first glance, class diagrams have been derived from these object diagrams by just collecting the employed classes, attributes and associations. Actually, during the editing of object diagrams, we used class diagrams mainly as a glossary. This means, if one creates a new object, he decides whether the type of this object is already known and may be reused or whether a new object kind, i.e. a new class needs to be introduced. Similarly, we handled the reuse of attribute declarations and associations. Thereby, the class diagrams leveraged the consistent use of classes, attributes and links through the different object diagrams. However, inheritance structures and design patterns were introduced at class diagram level. This was the task of the software experts. Discussing these class diagrams with the domain experts was not fruitful.

Due to the lack of space, we are not going to explain Figure 6. The point we want to

Threading Length

- If a bundle of wires, cf. bundle `b4` in Figure X[(Remark: Figure 4 in this paper)], needs to be threaded through a spout `sp1`, this requires mounting time according to the number of connectors that need to be threaded and to the length of the bundle that needs to be threaded.
- To compute the *threading length*, both sides of the spout have to be considered, in our example channel `ch4` and node `n1` are *direct neighbors* of spout `sp1`.
- If the direct neighbor is a channel, we consider the wires belonging to bundles that belong to the spout and to the neighbor channel. In our example, channel `ch4` contains bundle `b5` containing wires `w1` and `w2`.
- If the direct neighbor is a node, we consider the neighbor channels connected to that node. Only, wires belonging to the spout and to one of these neighbor channels are considered. In our example, this are again wires `w1` and `w2`. Note, wire `w3` must not be considered since it does not cross the spout.
- The length of a wire is the sum of the lengths on channels containing bundles that contain the wire.
- The maximal length of the two sides of the spout is the threading length.

Figure 5: Improved requirement description based on object diagrams (Created to exemplify object diagram discussions)

make is, that our method implementation with story diagrams use a graphical notation (cf. [KNNZ00, DGZ05, Zün02]) that again employs a variant of object diagrams. Since object diagrams have already become familiar to our domain experts during the requirements analysis, with some training, our domain experts were able to review our implementation. We had a lot of fruitful discussions, where the software developers explained their story diagrams and then the domain experts started to point to special cases. In these discussions, the domain experts again used object diagrams to illustrate the special cases they were concerned about. Then the software developers used simple walk throughs to check whether these special cases were already addressed or whether the story diagram had to be adapted to handle such a special case, appropriately. Actually, in the threading length example, such a review discussion revealed that the first implementation attempt erroneously considered wires that only bypass a spout.

To summarize the implementation phase, the use of story diagrams enabled our domain experts to review our implementation and to point us to special cases. In our experience, similar reviews by our domain experts would not have been possible for an implementation in a textual programming language like Java or C++.

4   Lessons Learned

The main complexity of the OBA project was the sheer amount of domain details that contribute to the construction of a car electric system. The OBA project was initiated by our professor for car systems at the University of Kassel. Thus, the project employed

Spout::getThreadingCostsPerCarArea (neighbor: HierarchyElement): Double

this ◄ allObaElems obasystem :OBASystem ► parameters params :Parameters

[failure]

[success]

System.out.println("ERROR: No OBA system parameters!");

0.0

this ► at channelNeighbor:=(Channel) neighbor

▼ contains ▼ contains

b1 :Bundle ► wires wire :Wire ◄ wires b2 :Bundle

1: Double threadingLength := Math.max (wire.getThreadingLength(this, this, channelNeighbor), threadingLength

[end]

this ► at nodeNeighbor:=(Node) neighbor ► at channelNeighbor:Channel

▼ contains ▼ at

b1 :Bundle ► wires wire :Wire ► wires b2 :Bundle

1: threadingLenght := Math.max (wire.getThreadingLength(this, nodeNeighbor, channelNeighbor), threadingLengt

[end]

this ► threadedConnectors connector :Connector

1: Double costs := costs + param.getThreadingTimePerConnector() * param.getCostsPerMountingTime

[end]

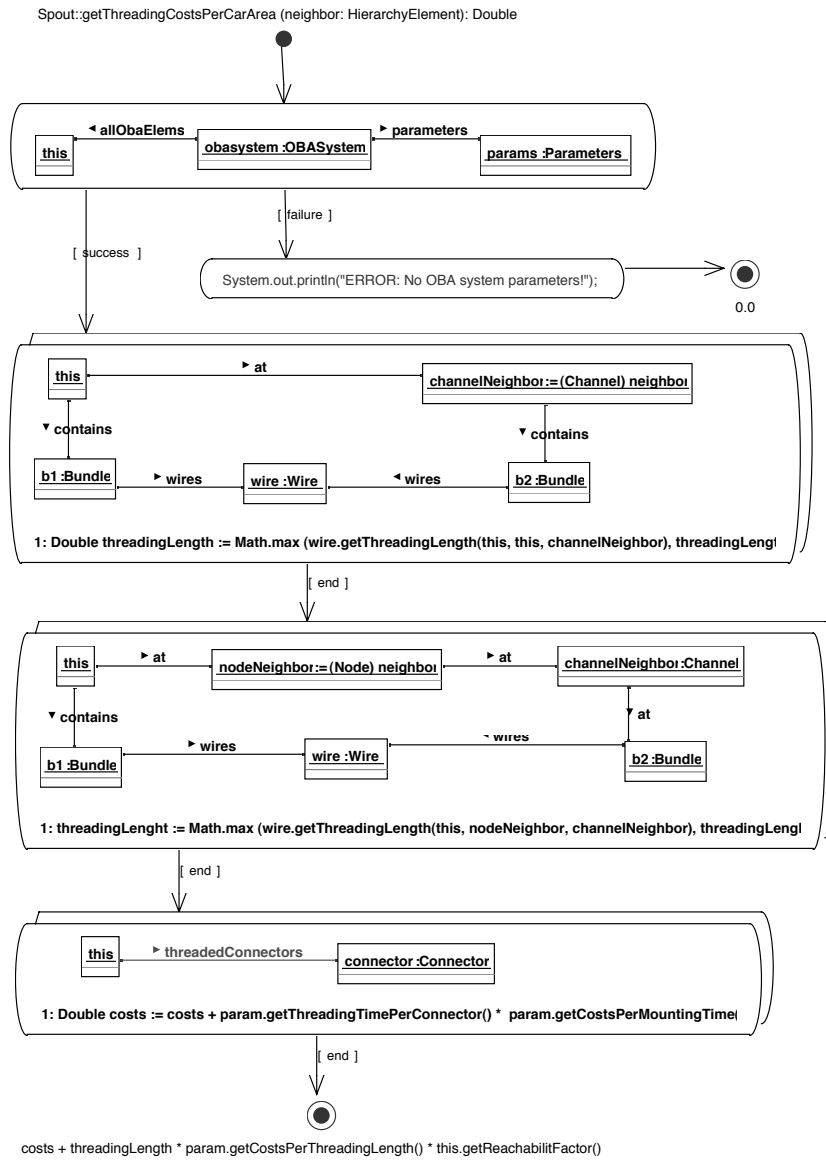costs + threadingLength * param.getCostsPerThreadingLength() * this.getReachabilitFactor()

Figure 6: Computation of the threading costs

two domain experts from the research group of car systems. For the design and implementation work, there was only two PhD student from the software engineering group of University Kassel. At the beginning, the software engineering group thought that one domain expert and three software developers would make a better team. However, after roughly one and a half years of software development, we have to state that the emphasis on domain experts was actually a key to success for the project. More precisely, since we had only two software developers, we were forced to involve the domain experts into the development process as much as possible. Using object diagrams we were able to do this in the requirements analysis. After a short learning curve, the domain experts were actually driving the requirements process. For each new aspect, they developed object diagram examples. These were discussed with the software experts in some iterations. Frequently, the domain experts from University of Kassel had to interview the domain experts in the contributing enterprizes in order to clarify details. Thus, the domain experts at University Kassel served as some kind of mediator between the actual customers and the software developers. Through object diagrams, this worked out very well. Similarly, the contribution of detailed implementation reviews by our domain experts was of unmeasurable value for the project. Again, this was enabled by the object diagram like notation used in story diagrams.

References

[BRJ99]   Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide; Addison Wesley, ISBN 0-201-571168-4, 1999

[DGZ05]   I. Diethelm, L. Geiger, and A. Zündorf: Applying Story Driven Modeling to the Paderborn Shuttle System Case Study; book chapter in S. Leue and T.J. Systä (Eds.): Scenarios, LNCS 3466, pp. 109133, 2005. Springer-Verlag 2005

[Fu02]   Fujaba Homepage, Universität Paderborn, http://www.fujaba.de/.

[GSZR et al. 05] R. Gemmerich, S. Semmelrodt, A. Zündorf, C. Reckord, J. Leohold, J. Trippler, L. Brabetz, D. Müller, U. Schrey, H.-G. Weil: An integrated approch for the generation and optimization of car electric systems. VDI (Hrsg.): 12th International Conference and Exhibition Electronic Systems for Vehicles Baden-Baden. 2005, pp. 597-608

[JBR99]   Ivar Jacobson, Grady Booch, James Rumbaugh: The Unified Software Development Process; Addision Wesley, ISBN 0-201-57169-2, 1999

[KNNZ00]   H. Köhler, U. Nickel, J. Niere, A. Zündorf: Integrating UML Diagrams for Production Control Systems; in Proc. of ICSE 2000, Limerick, Ireland, acm press, pp. 241-251(2000)

[Zün02]   Albert Zündorf: Rigorous Object Oriented Software Development with Fujaba. http://www.se.eecs.uni−kassel.de/se/fileadmin/se/publications/Zuen02.pdf 2002