Specifying Building Automation Systems with PROBAnD, a Method Based on Prototyping, Reuse, and Object-orientation

Andreas Metzger, Stefan Queins

Department of Computer Science, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
{metzger, queins}@informatik.uni-kl.de

Abstract: In this article, the PROBAnD requirements engineering method, which is specialized towards the domain of building automation systems, is presented. The method bases on object-orientation to handle complexity, reuse to gain efficiency as well as product quality, and prototyping to enable test-based verification and validation early in the development process. To demonstrate the applicability and efficiency of this method, the results of an extensive case study are introduced.

1 Introduction[1]

The development process of embedded real-time systems differs from that of other applications in many ways. One difference can be the small number of identical systems that are created. Especially in the building automation domain a specific system is delivered only once. Another difference lies in the fact that the life cycle of such systems is relatively long. Therefore, the development process needs to be very efficient and traceability has to be established during the whole life cycle, which includes development as well as maintenance. A further property of building automation systems that has to be dealt with is the complexity that results from the integration of control strategies for different physical effects (e.g., illuminance and temperature). This integration is required because of the physical coupling of these effects (e.g., artificial or natural lighting might heat up an area). Additionally, the complexity stems from the huge number of objects that have to be regarded (e.g., the number of components can range from few hundreds to many thousands).

In this article, we focus on the first phase of the development process, the requirements engineering phase. In this phase, the often-vague needs of the stakeholders have to be transformed into formal system requirements. To gain efficiency in performing this step, the development method needs to be specialized towards a specific domain. This specialization allows the definition of the method by describing the different development activ-

1. This paper has been published in P. Hofmann, A. Schürr (Eds.) *OMER – Object-Oriented Modeling of Embedded Real-Time Systems*. GI-Edition, Lecture Notes in Informatics (LNI), P-5. Bonn: Köllen Verlag. 2002. pp. 135–140.

ities and products in a very fine-grained manner. Further, the specialization permits the precise definition of guidelines for each activity.

The *PROBAnD* method is one example of such a domain-specific requirements engineering method. It relies on the application of three basic techniques: object-orientation, reuse, and prototyping. Object-orientation is employed to handle the huge number of objects. Reuse allows the reduction of effort by producing products of high quality. Prototyping permits the early validation of the system together with stakeholders as well as the verification of development decisions by developers.

2 Related Work

For the specification of embedded real-time systems, several techniques exist; e.g., Statecharts [Ha90] or real-time UML [Se99], which is a combination of ROOM [Se94] and the universal Unified Modeling Language, UML [Bo99]. Although Use Cases and Interaction Diagrams are widely used in the UML community, they are not qualified for completely and precisely describing the behavior of a system. A notation with well defined semantics is the Specification and Description Language, SDL [Ol97], which therefore is supported by commercial code-generators. In research projects, several of the above techniques have been extended to allow the specification of non-functional aspects. One example is SDL* [Sp97], which adds non-functional annotations to SDL.

Current development methods for embedded real-time systems often base on one of the modeling languages that are depicted above. Some universal methods, like the Unified Software Development Process [Ja99] or the Object-Engineering-Method [Ru01], claim to be applicable in the domain of real-time systems. Such universal modeling methods are specialized for certain development domains by applying derivations of universal modeling notations. Examples for such specialized methods are ROPES [Do01], which is based on real-time UML, and the Real-Time Object-Oriented Modeling Method [Se94], which is based on the ROOM notation.

Despite this specialization, these methods still approach a broad range of problem domains (e.g., the domain of real-time systems), and therefore have to be adapted to specific development domains (e.g., the domain of automotive control systems) before they can reasonably be executed. We have carried out this adaption step, for which we have explored the influence of characteristic properties of the building automation domain. This adaption lead to the definition of the PROBAnD method, which allows the efficient derivation of the requirements of building automation systems.

3 The PROBAnD Method

To introduce the PROBAnD method, an informal description of the underlying process

model is given, which includes the most relevant documents and activities (c.f. Fig. 1). This introduction can only give an overview, because the chosen level of abstraction for this article does not expose all development artefacts and fine-grained activities. For a comprehensive and in-depth description of the PROBAnD method please refer to [Qu02].

It should be noted that this process model does not enforce a strict *phase-oriented* execution, where each activity relates to a specific phase and the whole project may only be in one phase at the same time. Rather, a *workflow-oriented* style is preferred, where the activities are assigned to workflows, which can be executed simultaneously.

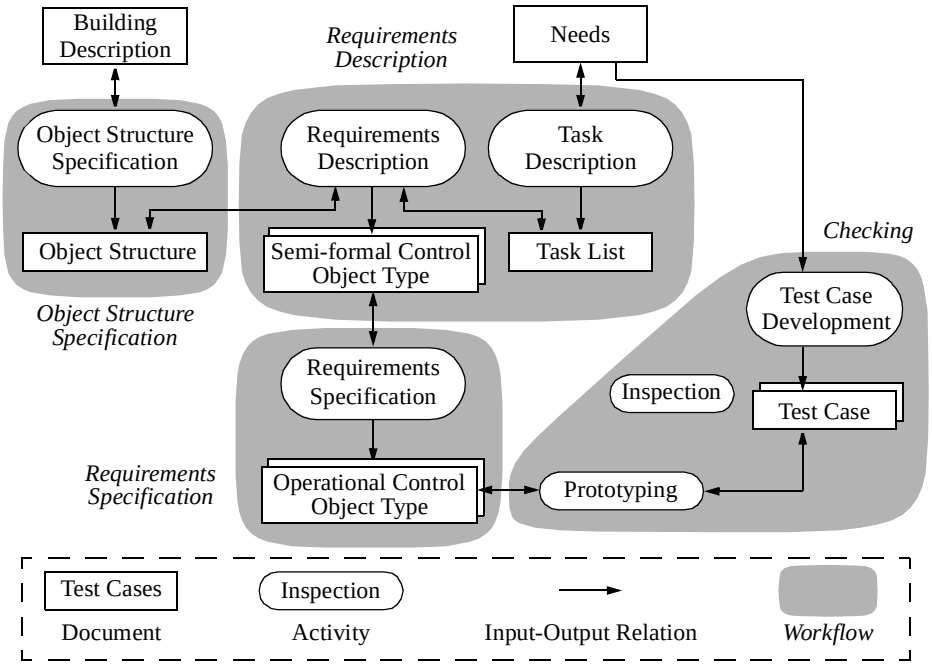

Fig. 1: Overview of the PROBAnD Process

The input for the PROBAnD method is the description of the problem, which can be divided into the *needs* and the *building description*. The needs informally describe the requirements from the point of view of the stakeholders. In the *requirements description* workflow, these needs are split into manageable *tasks*, which depict the requirements from the point of view of the developers and which have to be assignable to one *control object type*.

These control object types form the structure of the system, which is created in the *object structure specification* workflow. An initial *object structure* can easily be derived from the building description because the control object types are most often identical to the building's object types or are a sub-set of these (e.g., lighting needs to be controlled only within the boundaries of one room, and therefore the control object type that is responsible for lighting control can be identified with a room control object type). Only a strict aggrega-

tion hierarchy is allowed for the object structure, which follows the hierarchy within the building (e.g., the control object type for a floor aggregates control object types for the rooms of this floor). As requirements engineering proceeds, this control system structure can be refined, as long as the strict aggregation hierarchy of control object types is maintained.

As a guideline for the above activities, we suggest that the tasks that are assigned to a control object type should easily be realizable with the information that is available at that respective level in the hierarchy, which reduces the overall flow of information. Communication between control objects is realized through the exchange of (asynchronous) messages that are only allowed to travel along the aggregation hierarchy. This type of communication together with the strict aggregation hierarchy help maintaining the easy comprehensibility of the specification and allow the creation of documents from a set of few templates to simplify recurring activities.

After the control object types and their tasks have been elicited, the control object types are refined by defining strategies for how the tasks should be solved. These strategies are described in natural language, leading to *semi-formal control object type documents* that are expressed in HTML. Therefrom, each HTML control object type document is transformed into an operational (i.e. formal) document that is specified in SDL [Ol97] in the *requirements specification* workflow. In these documents, the aggregation of control object types is represented by a block hierarchy and strategies are expressed by extended finite state machines, which is a suitable notation for the chosen application domain, as states of control object types can easily be identified (e.g., a luminaire is turned on or off) and the change of states mostly occurs due to events (e.g., the pressing of a button).

Besides inspecting a document for verification purposes, *test cases* are developed that describe typical user interaction scenarios against which prototypes of the system are tested. Further, prototypes are employed for validating the system together with the stakeholders. To benefit from prototyping in the context of an efficient development process, the construction of the prototypes must be possible with little effort. In our approach, this is realized by *generating* prototypes from the set of available documents, which additionally guarantees consistency between the specification and the prototype. Besides SDL documents, HTML documents should also be accessible for prototype generation. To be able to generate prototypes from such semi-formal documents, the PROBAnD method's fine-grained and formally defined product model is employed [MeQ02]. Further, the generated prototypes can be used to produce *traces*, with which the dynamic behavior of the system can automatically be analyzed [Qu99] leading to results that can either provide feedback for requirements engineering activities or that can support decisions during the design phase.

In addition to prototyping, reuse is an important technique to achieve high quality products. Therefore, we allow access to all artefacts that were developed in earlier projects or that are being developed in the current project. These reuse artefacts can range from very small document parts up to complex collections of development products, like the whole

system requirements specification. All reuse artefacts are accessible through a so called *dictionary*, which allows the intuitive search for reuse candidates by employing the general structure of a building (e.g., reuse candidates for room control object types can be found by navigating through the dictionary's structure starting at the building and traversing over floor via section to room). Further, domain knowledge that was not packaged in a reusable artefact resides in this dictionary.

4 Case Study

To demonstrate the applicability of the PROBAnD method and to illustrate some of the development documents that were introduced in Section 3, one case study is presented in detail [QZ99]. In this case study an integrated heating and lighting control system for a floor of a university building was to be developed. This floor, which is separated into three sections, consists of 25 rooms of different types, which are connected by a hallway.

The description of the problem consisted of 68 different needs and a building description, which included the description of the installed sensors and actuators. Typical needs were

- "shortly before a persons enters a hallway section, the light should be turned on, if necessary",
- "the use of solar radiation for heating should always be preferred to the usage of the central heating unit", and
- "in the case of malfunctions the system shall provide a stepwise degradation of functionality".

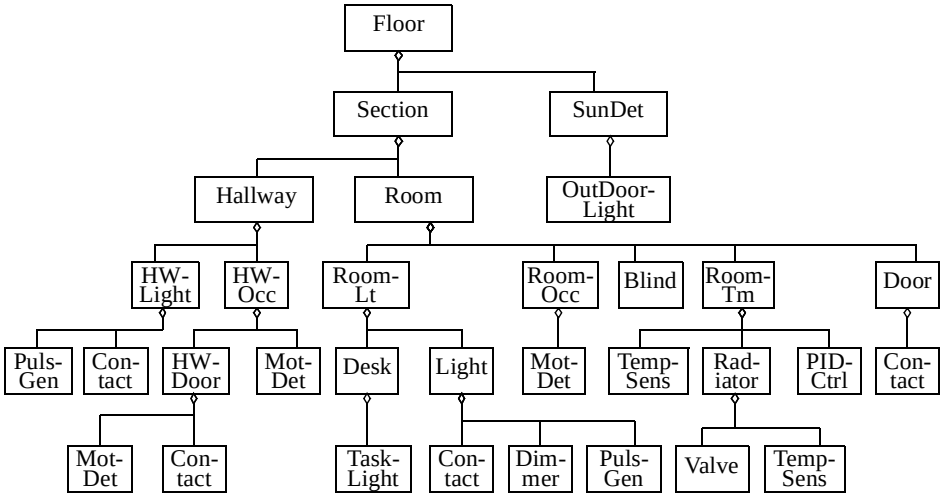

Fig. 2: Object Structure

These needs were split into 126 different tasks, which were assigned to 37 control object types (see simplified object structure in Fig. 2), which are instantiated to 920 instances. To give an impression of the complexity of the control object type documents, the respective

HTML files contained 1,280 lines of text. The total effort for the specification of the control system was approximately ten person weeks. Half of the effort was spent for the requirements specification workflow. In this workflow, 118 reuse operations were carried out, where in 81 cases an existing artefact could be reused. Again, to give an idea for the complexity of the resulting SDL specification, the textual representation (SDL-PR) contained 46,000 lines of text. All development products can be accessed on-line at

http://wwwagz.informatik.uni-kl.de/d1-projects/Projects/Floor32/

5 Conclusion

In this article, the efficient requirements engineering method PROBAnD has been presented. The efficiency of this method, which was illustrated by the results of a complex case study, is gained through the application of reuse and generator-based prototyping and the specialization towards a specific domain, the domain of building automation systems.

References

[Bo99] Booch G.; Jacobson, I.; Rumbaugh, J.: The Unified Modeling Language User Guide. Addison Wesley Longman, Reading (Mass.), 1999

[Do01] Douglass, B.P. : Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks and Patterns. 4th Print, Addison-Wesley, Boston (Mass.), 2001

[Ha90] Harel, D.; Lachover, H.; Naamad, A. et al.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems. IEEE Transactions on Software Engineering, Vol.16, No.4, 1990

[Ja99] Jacobson, I.; Booch, G.; Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (Mass.), 1999

[MeQ02] Metzger, A.; Queins, S.: Early Prototyping of Reactive Systems Through the Generation of SDL Specifications from Semi-formal Development Documents. In Proceedings of the 3rd SAM (SDL And MSC) Workshop, Aberystwyth, Wales. SDL Forum Society; University of Wales. 2002

[Ol97] Olsen, A.; Færgemand, O.; Møller-Pedersen, B. et al.: System Engineering Using SDL-92. 4th Edition, North Holland, Amsterdam, 1997

[Qu99] Queins, S.; Schürmann, B.; Tetteroo, T.: Bewertung des dynamischen Verhaltens von SDL-Modellen. SFB 501 Report 9/99, University of Kaiserslautern, 1999

[Qu02] Queins, S.: PROBAnD – Eine Requirements-Engineering-Methode zur systematischen, domänenspezifischen Entwicklung reaktiver Systeme. Ph.D. Thesis, Department of Computer Science, University of Kaiserslautern, 2002

[QZ99] Queins S., Zimmermann, G.: A First Iteration of a Reuse-Driven, Domain-Specific System Requirements Analysis Process. SFB 501 Report 13/99, University of Kaiserslautern, 1999

[Ru01] Rupp, C.: Requirements-Engineering und -Management. Carl Hanser-Verlag, München, 2001

[Se99] Selic, B.: Turning clockwise: Using UML in the Real-time Domain. Communications of the ACM, Vol. 42, No. 10, 199; pp. 46–54

[Se94] Selic, B.; Gullekson, G.; Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley & Sons, New York, 1994

[Sp97] Spitz, S.; Slomka, F.; Dörfel, M.: SDL* – An Annotated Specification Language for Engineering Multimedia Communication Systems. Sixth Open Workshop on High Speed Networks, Stuttgart, 1997