Definition and Instantiation of a Reference Model for Problem Specifications*

Martin Kronenburg Christian Peper
University of Kaiserslautern, Erwin Schrödinger Straße,
D-67663 Kaiserslautern, Germany
E-mail: {kronburg|peper}@informatik.uni-kl.de

Abstract

Based on the reference model for requirements and specifications of Jackson, Zave, et al. a reference model for problem specifications *of reactive systems is defined. Considering the requirements specification technique FOREST, it is shown that both reference models can be instantiated using a real–time temporal logic and that they are compatible with structural concepts like inheritance and aggregation.*

1. Introduction

If a customer asks a developer of (software) systems to produce a machine that solves a problem, the first and often most important activity is to *understand* and precisely *specify* the problem. The customer typically formulates a list of requirements that describe what the machine should do. However, requirements by themselves are not sufficient for the understanding of the complete problem.

Obviously, it must be distinguished between the *requirements* containing the desired part of a system and the *domain knowledge* containing the already existing part of a system. Even more, if the developer was able to decide whether the requirements are satisfiable based only on these two descriptions, these requirements would be a direct consequence of the domain knowledge. In other words, the requirements by themselves would describe something already existing and nothing has to be done. Generally, there must exist a further description that provides enough additional information for the developer to build the machine that satisfies the requirements. This description is called *specifications*. We call a document comprising all three descriptions a *problem specification*.

In Section 2 we will introduce a reference model for problem specifications that is based on existing approaches to the closely related specification of requirements. Section 3 contains a brief introduction to the FOREST approach as a concrete specification method for reactive systems. In Section 4 we discuss to which extent the FOREST approach is an instantiation of the defined reference model.

*This work was supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 501, "Development of Large Systems with Generic Methods".

2. A Reference Model for Problem Specifications

The goal of this section is to provide a reference model for problem specifications. A problem specification is a document comprising all information necessary to understand *which* system should be built. It does not contain information on *how* the system should be developed, for example which development process should be applied or when the system should be available.

Existing approaches to the definition of the notion *requirement specification* either focus on the *content* of a document containing requirements ([9, 12, 3]) or on certain *properties* such a document should satisfy ([11, 6, 4]). We regard both aspects as important. Nevertheless, due to space limitations we concentrate here only on the content. In Section 2.2 we will recapitulate the main aspects of the reference model for requirements and specifications proposed by Jackson, Zave, et al. ([12, 3]). Based on this reference model we define our reference model for problem specifications in Section 2.3.

2.1. Basic Notions

A *system* is a part of the real world that is for some reason considered as a unit. In a system several *phenomena* are collected and combined. A *phenomenon* is an aspect in the real world that is essential for the system. Phenomena are for examples states of a system (e.g. the temperature in a room), events occurring in the real world (e.g. the switching on of a light), or objects (e.g. sensors and actuators) and individuals (e.g. a facility manager of a building). *Terms* (e.g. words in natural language) are used to designate specific phenomena of a system. They are a prerequisite for each conversation about real world phenomena. We demand that there is a one–to–one relation between phenomena and terms, i.e., for each phenomenon there is exactly one term designating it, and each term designates exactly one phenomenon. *Statements* express relationships between several phenomena. Each statement is constructed of the terms representing the considered phenomena. A set of statements is called a *description*.

2.2. A Reference Model for Requirements and Specifications

The reference model of Jackson, Zave, et al. extends the introduced basic notions by several relationships, classifications and constraints:

Constraints on Phenomena and Terms

If a specification is intended to be the basis for the development of a system, it is necessary to distinguish between the part of the system that already exists, the *environment*, and the part of the system that has still to be developed, the *machine*. Each phenomenon/term can uniquely be assigned to one of these parts. Based on this distinction, the phenomena/terms are classified w.r.t. the following two aspects:

control: It is demanded that each phenomenon/term is controlled either by the environment or by the machine.

visibility: It is demanded that each phenomenon/term controlled by one of the parts is also visible to this part.

From these constraints concerning control and visibility the following kinds of phenomena/terms can be derived:

- **e**nvironment controlled, **h**idden for the machine (**eh**)
- **e**nvironment controlled, **v**isible for the machine (**ev**)
- **m**achine controlled, **v**isible for the environment (**mv**)
- **m**achine controlled, **h**idden for the environment (**mh**)

eh, *ev*, *mh*, and *mv* is called the *scope* of a phenomenon/term. It is required that each phenomenon/term has exactly one of these scopes. Figure 1 displays the relationship among the four different scopes, cf. [3].

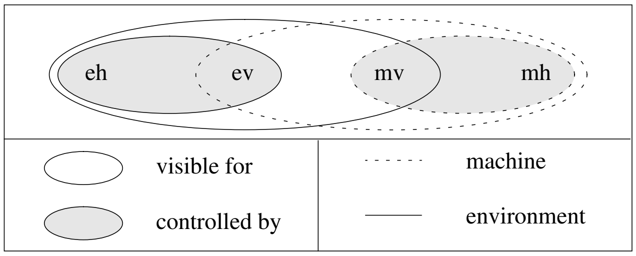

Figure 1. The possible scopes

Constraints on Statements and Descriptions

Each statement must be either in the *indicative* mood or in the *optative* mood. A statement in indicative mood describes something in the real world as it *is*. A statement in optative mood describes something in the real world as it *should be*. When combining the mood of statements with the scope of terms, a large variety of different kinds of statements is possible. Three combinations of mood and scope are reasonable:

domain statement: a statement in indicative mood where all terms are visible for the environment, i.e. have the scope *eh*, *ev*, or *mv*.

requirement statement: a statement in optative mood where at least one term of the scope *eh* is used.

specification statement: a statement in optative mood where all terms are visible for the machine, i.e. have the scope *ev*, *mv*, or *mh*.

Note that the combination of a statement in indicative mood with a term of scope *mh* is not reasonable: a statement in indicative mood describes something that already exists. Since the machine is the part of a system that has still to be developed a term of scope *mh* can merely represent a phenomenon that does not exist up to now. Even in an optative statement, a term of scope *mh* should only be allowed if this is explicitly desired by the customer. In general, a customer should make requirement statements, whereas specification statements should be restricted to the machine developer.

Since descriptions are sets of statements this classification can easily be extended to descriptions (cf. Figure 2):

domain knowledge: a description containing only domain statements.

requirements: a description containing only requirement statements.

specifications: a description containing only specification statements.

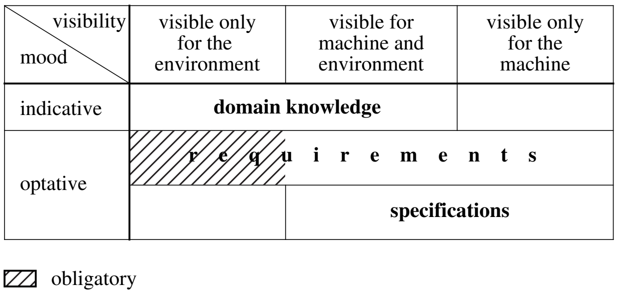

Figure 2. Classification of descriptions

2.3. A Reference Model for Problem Specifications

In the introduction of Section 2 we have described a problem specification as a document containing all the information necessary to understand *which* system should be built. Using the introduced terminology we can now refine the notion *necessary information*.

A problem specification is a document consisting of the following four components, where domain knowledge and specifications together must satisfy the requirements:

- a list of terms with a specification of their scopes and an explanation of the phenomena they represent
- domain knowledge using only these terms
- requirements using only these terms
- specifications using only these terms

The list of terms provides the vocabulary that can be used in the description parts. The domain knowledge represents the situation as it is without or in spite of the machine. The requirements describe how the environment should be under influence by the machine.

The information necessary for subsequent development steps is contained in the specifications. For the developer the specifications are an important information to assess the completeness of the problem specification and to determine the end of the problem analysis.

In addition to these aspects concerning the content of a problem specification such a document should be validated, pre– and post–traceable, testable, precise, comprehensible, consistent, realizable, complete, implementation-independent, adaptable, analyzable (cf. [11, 6, 4]). These properties are not elaborated here in more detail.

3. The FOREST Approach

In this section we introduce the FOREST approach. FOREST is an acronym for **Fo**rmal **Re**quirements **S**pecification **T**echnique and was designed as an approach to the creation of problem specifications of reactive systems. This approach comprises a process and a product model. The process model is described in detail in [10] and [2].

Here, we will give an informal overview of the concepts and techniques used in the FOREST product model. In order to enhance the compactness of a description and the reusability of parts of a description the following structuring concepts are provided in the FOREST approach:

- modularization
- aggregation
- parameterization
- inheritance

These concepts are illustrated by a running example. The example is an excerpt of a case study in which a problem specification for the light and temperature control in 20 different rooms equipped with over 100 sensors and actuators has been created.

Basic Formal Description Technique

The basic formal description technique used in the FOREST approach is a temporal logic based on the temporal logics described in [7, 8, 1] extended with some tailored operators (see [5]). We have chosen a temporal logic mainly for two reasons. Firstly, temporal logics have been proven to be well suited description techniques for modeling the behavior of reactive systems. Secondly, since a temporal logic allows a declarative and property–oriented description it is especially suitable for the specification of requirements.

Our first–order temporal logic consists of a set S of sorts, a set V of typed variables, a set F of function symbols, and a set P of predicate symbols. For each symbol f there is a declaration, e.g. $f : s_1, \ldots, s_n \rightarrow s$, where $s, s_1, \ldots, s_n$ are sorts. Based on the sets V, F, and P, the propositional operators ($\neg, \vee, \wedge, \ldots$), the first order quantifiers ($\exists, \forall$), and temporal operators ($\Box, \Diamond, \ldots$), the set of formulae $\mathcal{F}$ are defined as usual. A formula $\varphi \in \mathcal{F}$ is a *sentence* if it contains no free variables. As time domain we use the non–negative real numbers $\mathbb{R}_0^+$.

Finally, we outline the meaning of the temporal operators appearing in the following examples. $\Box\varphi$ is valid if φ is continuously valid in the future. $\triangleright$ is a tailored operator. $in \ \triangleright^f_{\leq t} \ out$ means that the value of *out* is derived from the value of *in*. The transformation of the value of *in* to the value of *out* is according to the function f. The constant t represents the maximal delay between a change of the value of *in* and a corresponding change of the value of *out*.

Modularization

To enhance the compactness and reusability of descriptions it is desirable to split a description in several parts representing specific parts of a system. The FOREST approach yields a document that consists of several so called *description classes*. Each description class consists of five sections:

- SIGNATURE
- DOMAIN KNOWLEDGE
- REQUIREMENTS/SPECIFICATIONS
- BASE CLASSES
- FORMAL PARAMETERS

The SIGNATURE section is a set of symbol entries. Each symbol entry contains

- a formal declaration of a function or predicate symbol
- an explanation of what phenomenon is represented
- the scope of the symbol in the sense of Section 2.2

Figure 3 contains an example of a description class for a (generic) sensor that is already installed. Note that there is a great difference between whether a sensor already exists or whether it still has to be developed. In the first case it is part of the environment, in the second case it is part of the machine. Which case is given is explicitly expressed by the scopes of the corresponding symbols.

The sections DOMAIN KNOWLEDGE and REQUIREMENTS/SPECIFICATIONS are sets of property–entries.[1] Each property entry consists of

- a property identifier
- a formula in temporal logic defining a property
- a translation of this formula in natural language

Description Class Sensor

Intention: This class provides all definitions to describe the behavior of a *sensor* that already exists in the environment.

FORMAL PARAMETERS

Domain ENV_D
Intention : possible values of the measured phenomenon

Domain $MEAS\_D$
Intention : possible values returned by the sensor

SIGNATURE

Function $envEntity : TIME \to ENV\_D$
Intention : value of the measured phenomenon
Scope : eh

Function $measEntity : TIME \to MEAS\_D$
Intention : measured value of the phenomenon
Scope : ev

Function $modifyDuringMeas : ENV\_D \to MEAS\_D$
Intention : function representing the modification of a value during measurement
Scope : eh

Function $reactionTime : \to TIME$
Intention : *reaction time* of the sensor
Scope : eh

DOMAIN KNOWLEDGE

Property $DomSens_1$
Formal : $\Box(envEntity \rhd^{modifyDuringMeas}_{<reactionTime} measEntity)$
NL : The measured value is always derived from the value in the real world using the function *modifyDuringMeas*. A change of the value in the real world is always propagated within *reactionTime* time units.

Figure 3. Example: Description class Sensor

Inheritance, Aggregation, and Parameterization

There are two kinds of relations between description classes: *aggregation* and *inheritance*. If a description class R is a refinement of a description class B, i.e. R inherits from B, this is expressed in the BASE CLASSES section of R. The meaning of such a refinement is that all symbols and properties introduced in the base class B are also valid in the refined class R. All symbols and property–identifiers have to be unique. In Figure 4 the description class *Binary Sensor* is displayed as a refinement of the *Sensor* of Figure 3.

Classes can be aggregated by using the keyword *Object* in the SIGNATURE section. In Figure 5 two motion detectors are aggregated as instantiations of the description class *Binary Sensor*. They are contained in the SIGNATURE section of a class describing the light control for a room.

[1] We expect that this usage of the notion *property* is not confused with its usage in the sense of Section 2.3.

Description Class BinarySensor

Intention: This class provides all definitions to describe a *binary sensor*. A binary sensor is a sensor that can return a 0 or a 1.

BASE CLASSES

Class $Sensor(ENV\_D = BINARY, MEAS\_D = BINARY)$
Intention : All definitions of *Sensor* are also provided here using 0 and 1 as the possible values of the measured phenomenon and as the possible values returned by the sensor.

Figure 4. Example: Inheritance

Description Class RoomLight

Intention: This class provides all definitions to describe the light environment and requirements common to any kind of room.

SIGNATURE

Object $md : BinarySensor[2]$
Intention : These objects represent the two room motion detectors.
⋮

Figure 5. Example: Aggregation

The meaning of aggregating an object *ob* in class A as an instantiation of class B is that all symbols and property identifiers of class B get "ob." as prefix. The resulting substituted symbols and properties are then valid in class A.

In the FORMAL PARAMETERS section parameters can be specified. These parameters can be instantiated when this class is refined or when it is aggregated. See for example Figures 3 and 4. When refining class *Sensor* to class *BinarySensor* the sort parameters *ENV_D* and *MEAS_D* are instantiated by the sort *BINARY*.

4. Instantiating the Reference Model for Problem Specifications

We now show to which extent the product model of the FOREST approach can be seen as an instantiation of the reference model for problem specifications. Again, we focus on the content part of a problem specification and neglect the properties required for such a document. Note that by this we also present an instantiation of the reference model of Jackson, Zave, et al. using a temporal logic.

Terms and Their Scopes

In the FOREST approach a term is either a function or a predicate symbol. The list of all terms is not given as one block. Instead it is distributed among the SIGNATURE sections of the description classes. In these sections also the scope of a term as well as an explanation of the corresponding phenomenon are given. Some remarks have to be made on the aggregation and inheritance of description classes.

The one–to–one relation between terms and represented phenomena is guaranteed by an extension of the symbol (term) in the class that inherits or aggregates another description class. In the case of aggregation this extension is done explicitly as described in Section 3 on the syntactical level. Concerning the inheritance this is done implicitly on the semantical level and is not explained here in more detail.

The scope of a term is not affected by inheritance or aggregation. Therefore, one has to take care whether the terms given in the refined or aggregated description class have the right scopes. Consider for example the difference between an already installed sensor and a sensor that is itself part of the development, cf. Section 3.

Due to these considerations we can summarize that the product model of the FOREST approach provides a list of the used terms accompanied with an explanation of the phenomena they represent. Additionally, a function $sc : F \cup P \to \{eh, ev, mv, mh\}$ can be specified assigning a scope to each symbol.

Statements and Their Moods

The notion *statement* of the reference model is in the FOREST approach instantiated by the notion *sentence*, a formula without free variables. The mood of a sentence φ is given by the section of the description class in which φ appears. If φ belongs to the DOMAIN KNOWLEDGE section it is an indicative sentence. If φ belongs to the REQUIREMENTS/SPECIFICATIONS section it is an optative sentence.

Whether an indicative sentence φ is a domain statement and whether an optative statement ψ is a requirement or specification statement can be decided using a function $symb : \mathcal{F} \to F \cup P$. $symb(\varphi)$ denotes the set of all function and predicate symbols occurring in φ.

Hence, we can easily identify the four components required of a problem specification. We emphasize that the considerations above also show that the classification mechanisms introduced in the reference model of Jackson, Zave, et al. can be combined with structuring concepts like modularization, parameterization, aggregation, and inheritance.

5. Conclusions

We have defined a reference model for problem specifications based on the reference model for requirements and specifications proposed by Jackson, Zave et al. A problem specification should contain all the information necessary to understand which system should be developed.

For the FOREST approach we have shown that its resulting products satisfy the constraints imposed by both reference models up to a large degree. Especially, we have shown in which way the reference models can be instantiated using a temporal logic. We have further emphasized that the structuring mechanisms scope and mood proposed in the reference models are well suited for the specification of systems. Furthermore, we have provided evidence that these classifications mechanisms can be combined without any problem with other structuring concepts like parameterization, inheritance or aggregation.

Currently we are developing a tool that is intended to support the creation of problem specifications following the FOREST approach and to perform several checks.

References

[1] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. of 5th LICS*, pages 390–401. IEEE Computer Society Press, 1990.

[2] R. Gotzhein, M. Kronenburg, and C. Peper. Reuse in requirements engineering: Discovery and application of a real-time requirement pattern. In A. P. Ravn and H. Richel, editors, *Proc. of the 5th Intl. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems, Lyngby, Denmark*, pages 65–74. Springer, 1998.

[3] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. http://www.cis.upenn.edu/~gunter/hol/papers/refmod.ps, 1998.

[4] P. Jalote. *An integrated approach to software engineering*. Springer, 1997.

[5] M. Kronenburg, C. Peper, and R. Gotzhein. A tailored real time temporal logic for specifying requirements of building automation systems. SFB 501 Report 16/96, University of Kaiserslautern, 1996.

[6] P. Loucopoulos and V. Karakostas. *System Requirements Engineering*. International Series in Software Engineering. McGraw Hill, 1995.

[7] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[8] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[9] D. L. Parnas and J. Madey. Functional documentation for computer systems. *Science of Computer Programming*, 25:41–61, 1995.

[10] C. Peper, R. Gotzhein, and M. Kronenburg. A generic approach to the formal specification of requirements. In *Proc. of the 1st IEEE Intl. Conf. on Formal Engineering Methods (ICFEM'97), Hiroshima, Japan*, pages 252–261, 1997.

[11] I. Sommerville. *Software Engineering*. Addison-Wesley, 1996.

[12] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997.