A Catalogue of Criteria for Evaluating Formal Methods and Its Application

Thomas Deiß, Martin Kronenburg, Dirk Zeckzer

{deiss|kronburg|zeckzer}@informatik.uni-kl.de

04/1997

Sonderforschungsbereich 501

Universität Kaiserslautern
Erwin-Schrödinger Straße
D-67653 Kaiserslautern

A Catalogue of Criteria for Evaluating Formal Methods and Its Application*

Thomas Deiß, Martin Kronenburg, Dirk Zeckzer

Abstract

A large set of criteria to evaluate formal methods for reactive systems is presented. To make this set more comprehensible, it is structured according to a Concept-Model of formal methods. It is made clear that it is necessary to make the catalogue more specific before applying it. Some of the steps needed to do so are explained. As an example the catalogue is applied within the context of the application domain building automation systems to three different formal methods: SDL, statecharts, and a temporal logic.

1 Introduction

Today, more and more software systems are constructed which are part of other systems. Typically, these software systems maintain an ongoing interaction with their environment. This class of systems is termed *reactive systems* since they typically react on stimuli of their environment. Such systems often have to show a complex behavior to fulfill the tasks they are intended for.

To make precise statements about such systems and especially about their behaviors *formal description techniques* can be used. Thereby we mean description techniques which are based on well-defined syntactical constructs which have a well-defined semantics. Besides being unambiguous, formal descriptions have further advantages. For example, they can be analyzed with mathematical rigor or they can be used to generate test cases.

In addition to a precise definition of the syntax and semantics of a formal description technique, it should be defined precisely what should be the content of a formal description and how this content should be arranged in a formal description. Also, the activities working on such descriptions should be defined precisely, i.e. *products* and *processes*, as understood e.g. in [Ost87], have to be defined precisely. A formal description technique together with such descriptions of products and processes will be called a *formal method* here.†

Up to now a broad spectrum of formal methods for describing systems and their properties has been developed. This rises the problem which of them is the most helpful one when developing a specific system in a specific context. In literature, this problem has already been considered several times throughout the last years. E.g. in [Bro96] requirements for

*This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Sonderforschungsbereich (SFB) 501, *Development of Large Systems with Generic Methods.*

†In the following, we will use the term *formal method* also for description techniques with not strictly formally or even informally described syntax, semantics, products, and processes.

formal description techniques and some models to be expressed by them are identified. In [ACJ+96] a set of criteria is listed and applied to seven different formal methods, including even a programming language. From these and other publications a large set of criteria has been compiled. This set has been extended based on experience gained by the authors when performing several case studies on the formal specification of reactive systems.

In general, by a criterion one aspect of a formal method is investigated. For example, the readability of descriptions written by using a specific formal description technique can be examined. In the catalogue, the criteria are given in the form of questions, and examples of useful answers representing possible attributes of formal methods are proposed. But it is left as an open problem, how a specific attribute contributes to the quality of a formal method.

Since the catalogue consists of a large set of criteria investigating very different aspects of formal methods, e.g. their semantics or the processes related to a formal method, the catalogue is structured and criteria covering similar aspects are arranged in groups. This structure is based on a *Concept-Model* of formal methods. Furthermore, some of the criteria investigate the same aspect, but with a varying degree of detail. Where appropriate, such criteria are arranged with respect to a refinement relation.

Because the collected criteria cover a broad range of aspects related to formal methods, there is the problem, that they are too general to be really useful in a specific application. As a first step to make the catalogue more specific the kind of systems described and the kind of descriptions considered are restricted. The criteria presented here are intended for the evaluation of formal methods for describing reactive systems and their behaviors. Descriptions considered directly are specifications of systems and their behaviors and designs of systems. Implementations are considered only indirectly by investigating the relationship between specifications and designs on the one hand and implementations on the other hand. This means for example to ask, whether an implementation can be derived from a specification by successive refinement steps.

Although the criteria become more specific by these restrictions, they and the spectrum of aspects covered by the catalogue are still too general. As an example, it is not specific enough to ask whether descriptions are readable. It has to be stated precisely on the one hand, when a description shall be considered readable and how this can be measured. On the other hand it has to be made clear, for whom such a description shall be readable and which background such a person has. It might be even necessary to rephrase the question or to replace it by several more specific ones to get the information needed for the evaluation. These problems and partial solutions for some of them are discussed in this report.

Therefore, the criteria catalogue as presented here should be considered only as a first step towards evaluating formal methods in a precise and well-defined way. It should be clear, that such evaluations are necessarily based to some degree on the subjective experiences and opinions of the individuals which contribute the information about the formal methods. But on the other hand, the criteria themselves, their application in a specific context, and their contribution to an overall evaluation have to be as precise as possible. In its current state, as presented here, the catalogue is no more than a structured collection of criteria. Making the catalogue and its application more specific and precise is left open to future work.

As an example, the catalogue is applied to three different formal methods used by the authors for specifying control systems in the application domain building automation systems. The formal methods used are SDL [SDLa], statecharts [Har87], and a real–time temporal logic called tRTTL [KPG96]. Two purposes are pursued by this application of the catalogue.

Mainly, the experience of the authors made by using these formal methods are recorded. At second, as far as possible, the formal methods are compared to detect some relative strengths and weaknesses.

The paper is structured as follows: In section 2 the Concept-Model of formal methods, which is used to structure the criteria catalogue, and the main notions related to formal methods are introduced. In section 3 the notion *criterion* and its use is explained. The problems arising from the generality of single criteria and the complete catalogue and some possible solutions are discussed in this section, too. The criteria catalogue itself and its structure is presented in section 4. The application of the criteria catalogue is illustrated by an example in section 5. The report ends with some remarks which describe how this catalogue can be improved by further applications of it, see section 6.

2 Concept-Model of Formal Methods

In this section the main notions and concepts relevant in the context of a formal method are clarified and comprised in the *Concept-Model of formal methods* which is depicted in figure 1. This model is used to structure the criteria catalogue.

Figure 1 has to be seen as an entity-relationship diagram [Che76] where each box represents a set of entities and each arrow represents a set of relationships. Since all relationship sets are $n:m$ mappings no mapping is depicted in this diagram in order to keep it clear. In the following the notion *concept* instead of *entity* is used.

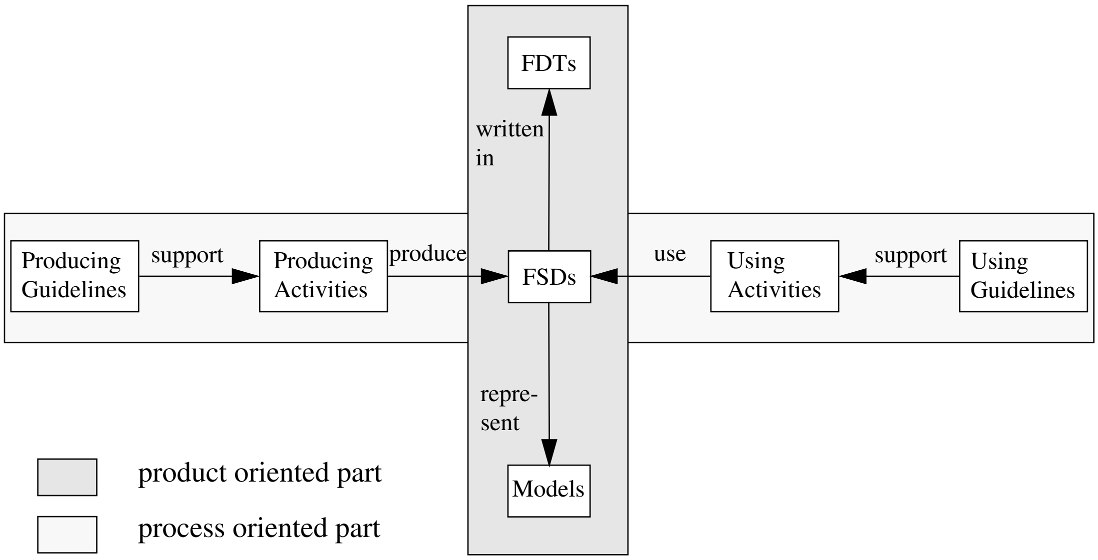

Figure 1: Concept-Model of formal methods

The basic concept within this model is that of *formal description techniques (FDTs)*. The other concepts are considered with respect to FDTs. An FDT provides a *syntax* and a *semantics*. The syntax is a formal language, i.e. an alphabet of atomic symbols and a set of rules defining in which way the words of the formal language can be constructed. The semantics defines mathematically the meaning of these words.

FDTs are used to write down *formal system descriptions (FSDs)*. A formal system description is a document written in one or more FDTs representing *models* of a system. A model is an abstraction of a specific system. Such abstractions of a system are needed since usually a system is too complex to describe all its features in detail. A system can be described either explicitly by presenting models of it or implicitly by stating its *properties*. The concepts FDTs, formal system descriptions, and models constitute the *product oriented part* of the Concept-Model.

In the *process oriented part* of the Concept-Model the *activities* dealing with the formal system descriptions are considered. Most of the activities can be assigned to two main classes of activities: *producing* and *using* activities. By *producing activities* formal system descriptions are produced as outputs. We regard activities changing formal system descriptions also as producing activities. *Using activities* use formal system descriptions as inputs. Moreover, there are activities, for example learning how to apply an FDT, which are not directly related to one of these two main classes. In the criteria catalogue they will be investigated separately.

Activities mainly specify *what* has to be done with a formal system description. Additionally, it has to be stated, *how* activities based on one or more FDTs can or should be performed. This should include both, a well-defined description how to perform *processes* and a precise explanation how to structure *products* and what information they should contain, see [Ost87]. Today, for most formal methods neither well-defined and precisely described processes nor precise descriptions of the content and structure of formal descriptions exist. This information is mostly given in the form of more informal *guidelines* and *rules*. They can be of a more general nature or specific for one formal description technique. We will call the combination of an FDT with a set of guidelines or rules, even if they are given in an informal way, a *formal method*, see also [BH93].

In figure 1 two concepts are omitted: *Persons* and *Tools*. They are not depicted since there is a large number of different relations between these two concepts and the other ones of the Concept-Model: Persons play a lot of different roles and tools can support different kinds of activities. Depicting all relations would make the Concept-Model in figure 1 too complex and confusing. Nevertheless, we will take the different roles and relations into account when presenting the catalogue in section 4.

A similar approach to clarify the principles relevant in the context of a formal method is given in [Bro96]. There, the author presents the following principles for a formal method[‡] to *make it a useful tool in system development*:

- formal syntax,
- formal semantics,
- clear conceptual system model,
- uniform notion of an interface,
- sufficient expressiveness and descriptive power,
- concept of development techniques with a proper notion of refinement and implementation.

[‡]Note, that in [Bro96] the notion FDT is used for that, what we have defined as formal method.

The first two principles are comprised by our concept *FDTs*, the following three principles are part of our concept *Models*, and the last principle corresponds to the *process oriented part* of our Concept-Model.

3 Annotations to the Criteria Catalogue

Before the criteria catalogue is presented in section 4.2 several annotations have to be made in order to make clear how this catalogue has to be read and utilized. Therefore we first explain the notion *criterion*; in the second part we point out what has to be done and considered when applying the criteria catalogue to a formal method.

3.1 Criterion

In order to evaluate a formal method it has to be characterized, i.e. the *attributes* of the considered formal method have to be found out and stated explicitly. To support this task *criteria* are used that investigate different aspects of formal methods, so-called *attribute types*. In the following we define what a criterion is and explain in which way a criterion will be represented in the catalogue of section 4.2.

Let $\mathcal{F}$ be a set of formal methods and $\mathcal{A}$ a set of *attribute values*. Then a *criterion* $\mathcal{C}$ is a function $\mathcal{C} : \mathcal{F} \rightarrow 2^{\mathcal{A}}$, where $2^{\mathcal{A}}$ is the power set of $\mathcal{A}$. By this definition it is made clear that a criterion $\mathcal{C}$ assigns a specific *attribute* to a formal method $f \in \mathcal{F}$, namely the attribute $\mathcal{C}(f) \subseteq \mathcal{A}$. An example for a criterion is a function $\mathcal{C}$ that assigns to a formal method f the set of metrical temporal properties that can be expressed using f, e.g. $\mathcal{C}(f) = \{\textit{bounded response}, \textit{bounded invariance}\}$.

Considering this definition there are three problems that have to be solved. Firstly, it has to be defined what contributes to a formal method f, i.e. what syntax and semantics is considered and what rules and guidelines are taken into account. Secondly, the set $\mathcal{A}$ of the attribute values of a criterion has to be determined. This includes a precise definition of each attribute value. Finally, the function $\mathcal{C}$ itself has to be defined, this means it has to be stated which subset of $\mathcal{A}$, i.e. which attribute, is assigned to which formal method f.

Usually, it is easy to make clear what contributes to a formal method. In contrast to this, up to now, very often it is neither possible to enumerate all attribute values of $\mathcal{A}$ exhaustively nor it is possible to define them precisely. Considering for example the effort needed to learn to write good system descriptions using a formal method, no-one would deny that this is an attribute of a formal method. But there is still the problem to define what is meant by *effort* and *good system descriptions*. Moreover, it is necessary to characterize the background of the person learning to use the formal method. As a consequence of this difficulty the assignment of an attribute to a formal method, i.e. the definition of a criterion, is a difficult task.

Although these problems obviously exist, we think that it is already now possible and useful to present a criteria catalogue. This structured collection of criteria has to be seen as an interim report and as a hint to what has to be done in the future. In order to represent a criterion $\mathcal{C}$ in the catalogue a *question* is given that focuses on the attribute type that is investigated by the criterion. A question instead of a description of the attribute type investigated by a criterion $\mathcal{C}$ is utilized in order to simplify the assignment of attributes to a formal method f. By using a question an answer given to this question with respect to f

can be interpreted as the attribute $\mathcal{C}(f)$.

Obviously, this kind of determining $\mathcal{C}(f)$ depends on the subjective view of the person answering the question, but it is a first step to the characterization of a formal method using criteria. An important goal is to replace as many questions as possible by precise definitions of possible attribute values, resulting in a well-defined function $\mathcal{C} : \mathcal{F} \rightarrow 2^{\mathcal{A}}$. Nevertheless, we are just at the beginning and therefore we have to use the question-variant.

To overcome this lack of precision and objectivity the following aspects have to be considered. It is necessary to define formally in a precise way the attribute values of each criterion. This can be done by introducing appropriate mathematical models. See for example [Bro96] where several mathematical models of systems and their description have been presented. Based on such mathematical models there are a lot of criteria for which an assignment of an attribute for a specific formal method is easily possible just by analyzing it. But there are also several criteria that require empirical data, for example the one mentioned above dealing with the effort needed to learn to write good system descriptions. In such cases we suggest to perform controlled experiments in the sense of [Bas96] in order to acquire the data needed.

3.2 Application of the Criteria Catalogue

When applying the criteria catalogue to a formal method two tasks have to be performed: *the tailoring of the criteria catalogue* and *the acquisition of data*. Because of the current representation of the criteria catalogue, i.e. the question-variant, the acquisition of data just means to answer the given questions. In the following we will therefore concentrate on the tailoring of the criteria catalogue.

Besides the general problems described in the previous section, e.g. the general difficulty to enumerate the possible attribute values of a criterion, there is another reason that necessitates a tailoring of the criteria catalogue. The criteria catalogue presented in section 4.2 is a general one. This means that a wide spectrum of attributes of formal methods is investigated in a general manner. Since the catalogue is usually applied with respect to a specific purpose and within a given context, it has to be adapted to become specific for an application. The steps that have to be performed in order to tailor the criteria catalogue are described in the following.

Define the Purpose of the Application of the Criteria Catalogue
At first it is stated which formal methods are investigated and for which purpose; for example in order to record in a structured way experiences made in a previous project or in order to determine whether a set of formal methods is suited for the use in a planned project.

Define the Context of the Application of the Criteria Catalogue
To define the context of an application several aspects have to be considered, for example:

- the application domain and the relevant models within this domain,
- the kind of descriptions that are created, e.g. specification or design documents,
- intended uses of a formal system description, e.g. verification or documentation,
- the background of the persons intended to use a description.

This list contains only some examples of aspects that contribute to the context; many further aspects can be relevant for a specific application of the criteria catalogue.

This information about the context as well as the purpose is needed in order to adapt the criteria catalogue in the next tailoring step.

Adapt the Criteria Catalogue

Based on the defined purpose and context two adaptions have to be made: the set of relevant criteria and their corresponding attribute values have to be determined. Criteria considered not to be relevant for the defined purpose and context are deleted. For example, if the purpose of the application of the criteria catalogue is the recording of experiences, and none were made with respect to changing a formal system description, then criteria investigating this aspect are not relevant. Furthermore, although the criteria catalogue is considered to be comprehensive for evaluating formal methods used for the specification and design of reactive systems, it might be necessary to add new criteria in order to capture attribute types of formal methods which are not taken into consideration by any of the criteria of the catalogue.

The second adaption concerns the set of attribute values of each relevant criterion. As far as it is possible these sets should be stated and the meaning of each attribute value defined. As already mentioned this is a very difficult task and for a large number of criteria this can not be done as precisely as it should be. Nevertheless, it should be explained what it means that a specific attribute is assigned to a formal method.

Assess the Criteria

The three steps described so far are sufficient for the tailoring of the criteria catalogue in order to characterize formal methods. This step is only needed if the purpose of the application of the criteria catalogue implies a valuation of a formal method, for example because it is to be compared with other formal methods.

The assessment of the relevant criteria determined in the previous step is performed on two levels. At first, the possible attributes, i.e. the subsets of the set $\mathcal{A}$ of attribute values, of each criterion are assessed. Next, the criteria are classified according to their importance with respect to the considered purpose and context.

To assess the several attributes we suggest to establish a partial order on the set $2^{\mathcal{A}}$. This order is in general a partial one, because there might be attributes that are not comparable. Based on these partial orders it is possible to assess a single formal method or to compare several formal methods with respect to one criterion.

In order to be able to assess or compare formal methods on a more global level the criteria can be classified in different categories representing the importance of a criterion. In [ACJ$^+$96] for example the classification *fundamental selection criteria* and *important selection criteria* is proposed. Again, it should be made clear why a criterion is assigned to a specific category.

Note, that as long as the definition of a criterion can not be done in a precise and formal way the assessment is a problematic task, since it depends on the person performing the evaluation. Each establishment of an order on the possible attributes that can be assigned by a criterion or the classification of the criteria depends on the subjective view of the person doing this, on his likes and dislikes, his background, and so on. Thus, no objective view is given.

Moreover, it can be asked whether it is really necessary to get an answer of the question: *Which method is objectively seen the best one within a given context?* It might be sufficient just to ask whether a formal method is well-suited for the context it is used in. To find the *best one* may be too cost expensive.

4 The Criteria Catalogue

This section is the central part of the report. Here we present a structured catalogue of criteria for classifying and evaluating formal methods. As already mentioned, these criteria are intended for the evaluation of formal methods for the specification and design of reactive systems.

The criteria presented have been collected from three different kinds of sources. Similar collections of criteria have been published, see for example [AB96, ABL96, ACJ+96, Bro96, CGR93, dR91, Fau95, HL94, Zav91]. These have been looked through, and a lot of the criteria in these publications have been taken over to the catalogue presented here. Also, colleagues from all subprojects of the Sonderforschungsbereich (SFB) 501 contributed to a list of criteria, which served as a first version of this catalogue [BDD+94]. The criteria gained from these sources have been complemented by further ones based on the experience of the authors in specifying reactive systems, see e.g. [Dei94, Dei96, PGK97, DH97], and the homepages of two *Teams*[§] within the SFB 501.

The structure of the criteria catalogue is described in section 4.1. The catalogue itself is presented in section 4.2. Note, that this catalogue has to be tailored to a specific purpose before it can be applied. This is described in more detail in section 3.

4.1 Structure of the Criteria Catalogue

In order to simplify the readability and the application of the criteria catalogue it is structured hierarchically, whereby two structuring principles are used. At first the set of all criteria is divided into subsets aggregating criteria that investigate similar aspects. This principle is used hierarchically, i.e. a subset can be split into further subsets, resulting in a hierarchy of so called *groups* (of criteria). Such a division into groups of criteria is not necessarily disjoint. In the catalogue presented in section 4.2 there are some criteria which are mentioned several times but explained and refined only once in a general manner, for example the criteria investigating tool support and learning support. Note, that a group is not a criterion itself but a collection of criteria.

The groups and subgroups are established according to the structure of the Concept-Model. The set of all criteria is divided into four main groups: **Product Oriented**$_{\mathrm{A}}$ criteria, **Process Oriented**$_{\mathrm{B}}$ criteria, criteria with respect to **Tools**$_{\mathrm{C}}$, and criteria concerning **Persons**$_{\mathrm{D}}$. In accordance with the Concept-Model the set of product oriented criteria is divided into the subgroups **FDT**$_{\mathrm{Aa}}$, **FSD**$_{\mathrm{Ab}}$, and **Models**$_{\mathrm{Ac}}$, and the set of process oriented criteria into the subgroups **Producing**$_{\mathrm{Ba}}$ and **Using**$_{\mathrm{Bb}}$. The complete group-hierarchy of the criteria catalogue is depicted in figure 2. Note, that the proposed division into groups and subgroups is not the only possible one, it results from the structure of the Concept-Model.

[§]`http://wwwsfb501.informatik.uni-kl.de:8080/team1doc` and `/team2doc`

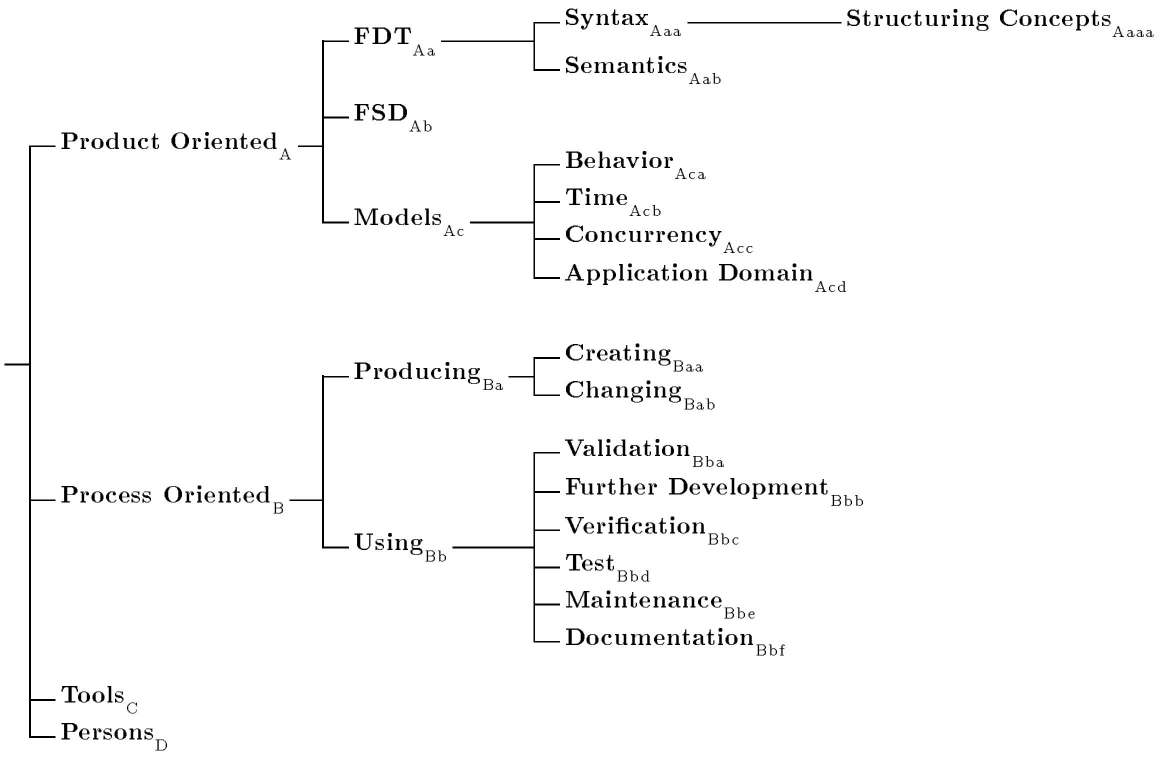

Figure 2: Group hierarchy of the criteria catalogue

Between some criteria of a group a *refinement relation* can exist. This is the second structuring principle. A criterion $\mathcal{D}$ is a *refinement* of a criterion $\mathcal{C}$ if the application to a formal method f is only meaningful if a specific attribute $\mathcal{C}(f)$ is assigned to f by $\mathcal{C}$, and if $\mathcal{D}$ focuses on a special aspect of the attribute type of $\mathcal{C}$. For the criteria presented here, it will be obvious or stated explicitly under which conditions a refining criterion is meaningful.

For easier reference each criterion and each group of criteria is given a name and a unique label. To distinguish criteria and groups, criteria names are written in a sanserif font and labeled by sequences of numbers, group names are written in **bold** font and labeled by sequences of letters. These sequences correspond to the location in the group-hierarchy or in the refinement-hierarchy of criteria, respectively.

4.2 The Criteria Catalogue

A: Product Oriented$_{\text{A}}$

This group comprises the criteria concerning the product oriented part of the Concept-Model. According to the structure of the Concept-Model this group is split into the subgroups **FDT**$_{\text{Aa}}$, **FSD**$_{\text{Ab}}$, and **Models**$_{\text{Ac}}$.

Note, that the group **Models**$_{\text{Ac}}$ contains criteria, which investigate which models can be described in principle by an FDT. The models relevant in a specific application domain have to be considered when applying the catalogue to a specific purpose and context. This is discussed in section 3.2.

Aa: FDT$_{\text{Aa}}$

Here all criteria are mentioned that are directly related to an FDT. This group contains the criteria examining which syntactical constructs are provided to structure a formal system description. They are collected in the group **Syntax**$_{\text{Aaa}}$. By the criteria of the group **Semantics**$_{\text{Aab}}$ it is considered how the semantics of the syntactical elements is defined. Additionally, there is one criterion dealing with the Specification Style$_5$ of an FDT. The structure of this group is depicted in figure 3.

Aaa: Syntax$_{\text{Aaa}}$

This group deals with the structuring concepts that are provided by an FDT. Furthermore, the notational style is considered.

1 Notation$_1$

Which notation-style is provided by the FDT?

This criterion deals with the basic symbols provided by the syntax of an FDT to construct descriptions and which are the carrier of information. Since the syntax of an FDT is a formal language the questions asks for the kind of alphabet of the FDT. Note, that this question is independent of the criterion Degree of Formality$_4$ where we consider how formally the interpretation of the formal language is performed. Examples of attribute values are:

textual: Natural language is used.

mathematical: The alphabet of the FDT includes mathematical symbols like $\forall$, $\sum$ that are not used in natural languages.

tabular: Tables are provided, for example for structuring a description (see for example [CP93]).

graphical: Circles, rectangles, and other graphical symbols can be used in a description.

Obviously, one FDT can offer different notation-styles.

The notation-style of an FDT is considered, since it can influence the readability of formal system descriptions. For example it is often claimed that a graphical or tabular notation can more easily be understood by a person not familiar with an FDT than a mathematical one. See also Readability$_{38}$.

Aaaa: Structuring Concepts$_{\text{Aaaa}}$

The criteria of this group deal with the syntactical concepts provided by the FDT to structure a description. The semantics of the structuring concepts has to be well-defined by the FDT.

2 Modularity$_2$

Can a formal description be structured and decomposed into manageable parts?

To answer this question with *yes* the FDT has to provide concepts for building a system description from several smaller pieces of descriptions. The meaning of each part of the description as well as the meaning of the composition of some parts has to be defined by the FDT.
Note, that the structure of a description does not need to coincide with the structure of the system, although this usually simplifies the understanding of the description. See also the criteria Interface Model$_{12}$ and Structure$_{13}$.

2.1 Interface$_{2.1}$

Is the notion of an interface supported as a syntactical construct by the FDT?

To get an overview and to support consistency and completeness of formal system descriptions it is helpful to have a clear notion of an interface as a syntactical object.

For example, a list of input and output channels together with the sort of messages sent over them, can be considered as the interface of a component of a system. Which information can be described in an interface is investigated by criterion Interface Model$_{12}$.

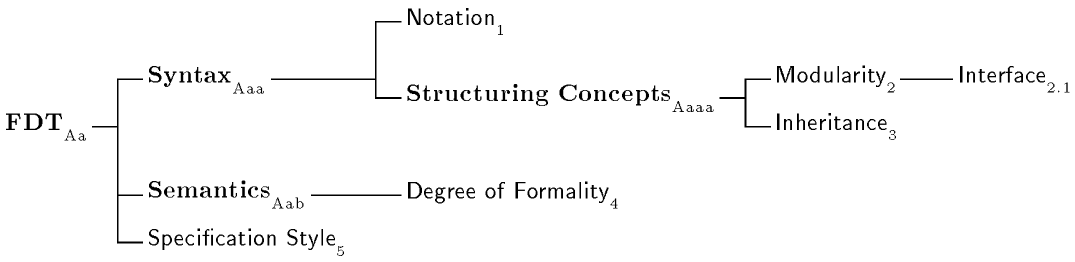

Figure 3: Structure of the group **FDT**$_{\mathrm{Aa}}$

This example is biased towards a correspondence between parts of a description and components of a system. But interfaces are also useful for other kinds of descriptions, e.g. signatures of an abstract data type can also be considered as an interface.

3 Inheritance$_3$

Which kind of inheritance is supported by the FDT?

Similar to modularity inheritance can be a powerful concept for structuring a description. We distinguish between the following attribute values:

no inheritance: Inheritance is not supported by the FDT.

single: An object can inherit features from at most one other object.

multiple: An object can inherit features from more than one other object.

Since modularity and inheritance are the main structuring concepts provided by object orientation this concept is not mentioned as a separate criterion.

Aab: Semantics$_{\mathrm{Aab}}$

This group contains only one criterion that deals with the way in which the semantics of the FDT is defined.

4 Degree of Formality$_4$

In which way is the semantics of the FDT defined?

By this criterion it is examined how formally the basic symbols of the syntax, see group **Syntax**$_{\mathrm{Aaa}}$, are interpreted. Examples of attribute values are:

informal: The semantics is only given in natural language, so that one gets a rough idea of what the symbols mean.

formal: Each word of the formal language of the FDT has a mathematically well defined, i.e. unique and unambiguous, meaning.

rigorous: This is the case if the definition of the semantics is given in a mixture of natural language and mathematical notions.

no way: The meaning of the formal language is in no way explained but left open to the intuition of the persons using the FDT.

Note, that in this criterion the notion *formal* description technique is used in a broader sense than it was defined in section 2. Here we also consider description techniques that have no mathematically defined semantics. In some subsequent criteria this broader sense is used, too.

5 Specification Style$_5$

Which specification style is supported by the FDT?

We want to distinguish between two specification styles, which can be mixed:

property oriented: This style is also called *declarative style* [dR91] or *extensional model* [CS96].

model oriented: This style is called by the same authors *operational style* and *intensional models*, respectively.

According to [dR91] the distinction is whether the specification of a system itself models a computational process (model oriented) or not (property oriented).

Note, that this criterion does not belong to the group **Semantics**$_{\mathrm{Aab}}$.

Ab: FSD$_{\mathrm{Ab}}$

This group consists of all criteria related to formal system descriptions written in an FDT. The criteria Internal Completeness$_9$ and Internal Consistency$_{10}$ are refined and structured similarly. This structure and the refined criteria are explained only once for the criterion Internal Completeness$_9$. The entire structure of this group is depicted in figure 4.

6 Combination of Notations$_6$

Is it possible to use different notations in one formal system description?

This criterion is only meaningful if more than one notation style has been mentioned by criterion Notation$_1$. The question should be answered with yes only if a relation between the different styles is provided, for example translation rules. This criterion is similar to the criterion Application of Domain Models$_{25}$. Here, the emphasis is on different notation-styles in one description whereas criterion Application of Domain Models$_{25}$ focuses on the combination of models of the domain and the system.

7 Abstraction$_7$

Is it possible to look at a system on various levels of abstraction in one formal system description?

What details can be omitted between different levels of abstraction depends on the considered FDT and the application domain. This criterion is related to the criterion Orthogonality$_8$.

7.1 Refinement$_{7.1}$

Is there a refinement relation between formal system descriptions on different levels of abstraction?

By a refinement relation it is defined how different levels of abstraction are related.

8 Orthogonality$_8$

Is it possible to have different views on a system on the same level of abstraction in one formal system description?

In contrast to the criterion Abstraction$_7$ the same level of abstraction is considered but from different points of view. If an FDT allows to represent several models of a system, for example the control and data flow, there are automatically different views. Furthermore, even within one such model different views can be expressible. For example, one can describe the behavior of a system separately for each of its interfaces. By this criterion both cases are considered.

8.1 Compatibility$_{8.1}$

Is there a notion of compatibility between different views of one system (expressed in the same formal system description)?

A *notion of compatibility* means that it is defined precisely under which conditions two different views are not contradictory. This criterion is related to the criterion Internal Consistency$_{10}$.

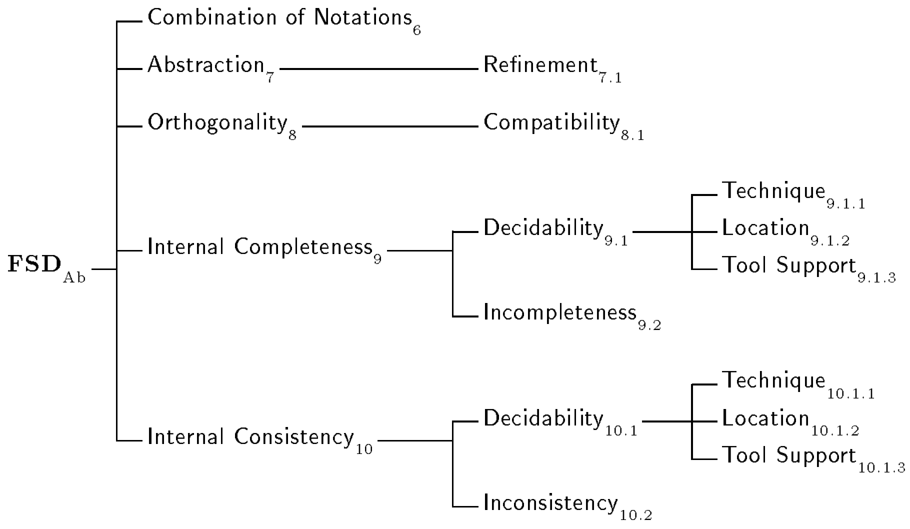

Figure 4: Structure of the group **FSD**$_{\mathrm{Ab}}$

9 Internal Completeness$_{9}$

Is there a notion of internal completeness of formal system descriptions defined?

Examples (not meant as criteria) for the notion internal completeness are:

- In each state a reaction on the occurrence of each input event is defined.
- All inputs are used somewhere, all possible outputs can be produced. This is supported by having syntactical constructs for interfaces, see criterion Interface$_{2.1}$.
- The transition function of a state transition system is total.

9.1 Decidability$_{9.1}$

Is it possible to decide whether a formal system description is internally complete?

This is a more theoretical criterion pointing out how hard the problem is.

9.1.1 Technique$_{9.1.1}$

By which technique can it be decided whether a formal system description is internally complete?

Examples of attribute values are:

reviews: It can be found out by reading the description, perhaps according to a specific reviewing strategy (see criterion Reviewing Strategy$_{46.1}$), whether the description is complete or not.

manual proof: A proof to show or disprove the completeness can be constructed with paper and pencil.

automatic proof: The completeness property can be checked automatically by an appropriate tool.

9.1.2 Location$_{9.1.2}$

Is it possible to locate the incomplete parts of a formal system description?

It can not only be decided whether a description is complete or not, but if it is not, there is a technique allowing to find the incomplete parts.

9.1.3 Tool Support$_{9.1.3}$

Are there tools supporting the completeness check?

9.2 Incompleteness$_{9.2}$

Is it possible to work in a useful way with a formal system description, which is not internally complete?

This criterion is important for several reasons: Since it is sometimes not possible to decide whether a description is complete or not and since usually a lot of work has to be done in order to achieve a complete description it is often necessary to continue the development process although a description is incomplete. In the case of the execution of an incomplete automaton it is for example possible to provide the missing parts manually throughout the execution when they are needed.

10 Internal Consistency$_{10}$

Is there a notion of internal consistency of formal system descriptions defined?

There are a lot of different notions of internal consistency in the literature. Examples for the notion internal consistency are (not meant as criteria):

- absence of non-determinism, i.e. disambiguity
- logical consistency, i.e. satisfiability
- compatibility between different views, see Compatibility$_{8.1}$

10.1 Decidability$_{10.1}$

Is it possible to decide whether a formal system description is internally consistent?

10.1.1 Technique$_{10.1.1}$

By which technique can it be decided whether a formal system description is internally consistent?

10.1.2 Location$_{10.1.2}$

Is it possible to locate the inconsistent parts of a formal system description?

10.1.3 Tool Support$_{10.1.3}$

Are there tools supporting the consistency check?

10.2 Inconsistency$_{10.2}$

Is it possible to work in a useful way with a formal system description, which is not internally consistent?

Ac: Models$_{\mathrm{Ac}}$

This group collects all criteria investigating which models can be described in principle by the FDT. This means that there are appropriate syntactical elements with a well-defined semantics that can be used to describe the models. In figure 5 the structure of this group is depicted.

11 Data$_{11}$

What aspects of data can be expressed?

The following list of attribute values to this question is not meant to be exhaustive, but as a proposal for some aspects which are relevant concerning the representation of data.

abstract data types: It is possible to define abstract data types and operations on these data types.

refinement: The FDT supports a refinement between data types. An example of a refinement of a data type is that the abstract data type `stack` can be refined either to an `array` or a `list`.

specialization: It is possible to express specialization/generalization of data. Thereby similarity between data types can be expressed. For example a `sorted list` is a specialization of `list`. See also criterion Inheritance$_{3}$.

12 Interface Model$_{12}$

What aspects of an interface between components can be described?

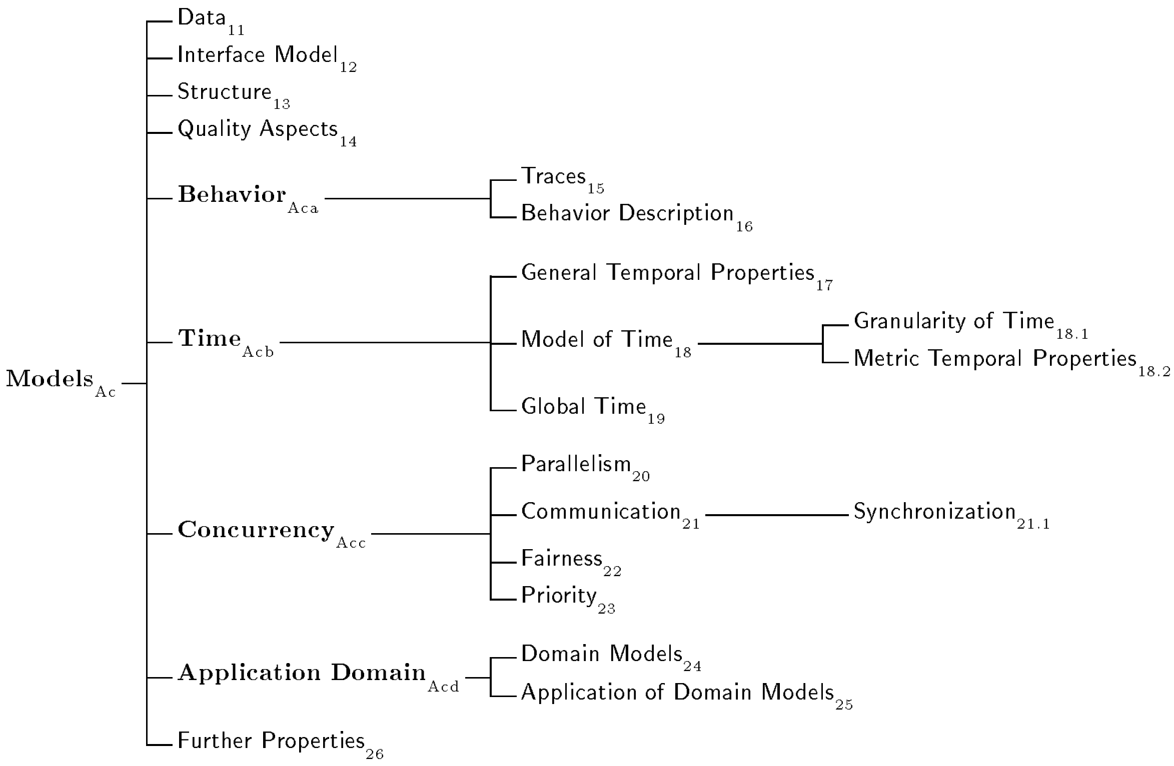

Figure 5: Structure of the group **Models**$_{Ac}$

Note, that this question deals with the interfaces between components of the system and not between the system description and the description of the environment. This is considered by criterion Application of Domain Models$_{25}$.
Obviously, this question is only meaningful if the FDT supports modularity and interface descriptions, see the criteria Modularity$_{2}$ and Interface$_{2.1}$. The following attribute values are taken from [Che70].

data type: Data type of the information exchanged by the components.

monitored/controlled: It can be expressed whether a value is monitored or controlled by a component.

magnitude: The magnitude of the values can be described.

probability: The probability that certain values occur can be described.

time: It is possible to specify the time a value occurs.

frequency: It is possible to specify the frequency a value occurs.

13 Structure$_{13}$

What structural aspects of a system can be expressed?

This criterion examines what static aspects of a system structure can be described. Examples of attribute values are:

hierarchical: It is possible to express the structure of a system as a set of hierarchically related components.

flow of data: It is possible to describe the flow of data or information between various components of the system.

14 Quality Aspects$_{14}$

What quality aspects can be expressed?

Quality aspects correspond to non-functional requirements that are not temporal requirements. Since there is a large number of different quality aspects the following list of attribute values is not meant to be exhaustive:

performance of a system or its components

availability of a system or its components

fault-tolerance of a system or its components

robustness of a system or its components

In dependence on a specific application domain this list has to be adapted (see also section 3.2).

Aca: Behavior$_{Aca}$

The two criteria of this group investigate how the behavior of a system can be described by an FDT.

15 Traces$_{15}$

Which kind of elements constitute the observable traces?

The behavior of a reactive system is usually described by a trace resulting from the observation of the environment and/or the system. We want to distinguish between two kinds of basic elements such traces can be composed of. It is also possible that both kinds occur in the same trace.

states: An element of a trace describes the values of observable variables; this can include control variables. Normally, a state is expected to be valid for a period of time greater than zero.

signals: An element of a trace describes that a variable has a specific value. A signal is often also called an *event* and is considered as occurring exactly at one instant of time, having no temporal extension.

16 Behavior Description$_{16}$

How can the dynamic behavior of the system or its components be described?

The following examples of attribute values can be supported both by one FDT:

blackbox: The behavior of a system is described by viewing the system as a black box, i.e. the internal structure of the system is not considered when describing the behavior.

whitebox (glassbox): The behavior of the system is described by displaying the behavior of the individual components, thereby using internal structure of the system.

Note, that in [Bro96] the notion *blackbox* is used in a different way. There a blackbox description specifies only the interface of a system component, but not its behavior.

Acb: Time$_{Acb}$

When specifying reactive systems it is substantial to be able to express temporal aspects. For that reason we consider several criteria directly related to time.

17 General Temporal Properties$_{17}$

What kind of general temporal properties can be expressed?

According to [AS85] the following two classes of temporal properties can be distinguished:

safety: Informally, a safety property prescribes that something bad never happens.

liveness: Informally, a liveness property prescribes that something good will eventually happen.

In [AS85] these classes are also defined formally. In [MP90] further classes of temporal properties are introduced and analyzed formally.

18 Model of Time$_{18}$

In which way is time represented?

We distinguish between a *quantitative* and *qualitative* model of time, depending on whether it is possible to express metrical aspects of time. In this case it is for example possible to express not only that event A occurs before event B, but also that it occurs δ time units earlier.

The following two refinements of this criterion are only meaningful if there is a quantitative notion of time.

18.1 Granularity of Time$_{18.1}$

What granularity has the time domain?

Examples of attribute values are:

discrete: Between two time points there is always a finite number of further time points. A typical time domain with this property is $\mathbf{N}$, the set of natural numbers.

dense: Between two different time points there are always infinitely many time points. Typical time domains having this property are $\mathbf{Q}^+$ and $\mathbf{R}^+$.

18.2 Metric Temporal Properties$_{18.2}$

What kinds of metric temporal properties can be described?

The following list of attribute values is not meant to be exhaustive:

bounded response: Between two observable elements of a trace only a maximal period of time is allowed to pass.

minimal separation: Between two observable elements of a trace there must be a minimal time period.

duration: A property has to be valid for a given amount of time.

frequency: An observable element has to occur n times during m time units.

timeouts: A signal is triggered after a specified amount of time.

19 Global Time$_{19}$

Is time considered as being global or is time considered local to individual components?

Time is called *global* in a system if there is the same time for all its components. If time is *local*, i.e. each component can have its own time, this implies that time can proceed differently in the components.

Acc: Concurrency$_{\text{Acc}}$

When specifying reactive systems criteria concerning the description of concurrency are as important as those for temporal aspects considered in the previous group.

20 Parallelism$_{20}$

How are traces of concurrently proceeding components composed?

Usually two kinds of parallelism are distinguished, see e.g. [CS96]:

interleaving: In this case *concurrency is 'reduced' to nondeterminism by treating the parallel execution of actions as the choice between their sequentialisations.* Each choice establishes a total order on the union of the observable elements of the several traces.

maximal parallelism: This is also called *true concurrency.* In this case *concurrency is treated as a primitive notion.* Besides the orders on the single traces there is no additional ordering relation between observable elements of different traces.

21 Communication$_{21}$

What communication-concept between components is used?

Examples of attribute values are *shared variables* and *message passing*.

21.1 Synchronization$_{21.1}$

Is communication between components synchronous or asynchronous?

This criterion is only meaningful when *message passing* is provided as communication-concept.

22 Fairness$_{22}$

What kind of fairness is supported?

Fairness is important if there is a restricted resource which two or more components of system want to use simultaneously. This is a typical situation when specifying reactive systems. Usually two kinds of fairness are distinguished, see e.g. [MP93]:

weak: It is forbidden that an action or transition *is continually enabled beyond a certain point but taken only finitely many times.*

strong: It is forbidden that an action or transition *is enabled infinitely many times, but taken only finitely many times.*

23 Priority$_{23}$

Is it possible to express priority between components?

The reason why it can be useful to express priority between components is the same as for Fairness$_{22}$: resolve conflicting access to restricted resources.

Acd: Application Domain$_{\text{Acd}}$

Up to now we have only considered criteria concerning the FDT which are related to the description of a *system*. Since it is important to embed and analyze a reactive system in the environment in which it should operate, in this group criteria dealing explicitly with the application domain are presented.

24 Domain Models$_{24}$

Which models of the application domain can be expressed in the considered FDT?

Obviously this question can only be answered with respect to a specific application domain.

25 Application of Domain Models$_{25}$

How can formal descriptions and models of the application domain be combined?

Usually a system description has to be connected with descriptions of the application domain in order to get a meaning. The combination of a system description with a description of the application domain written in a *different* FDT is normally more difficult than the combination with a description written in the *same* FDT.

26 Further Properties$_{26}$

What further properties can be described?

Besides the non-functional properties considered by the criterion Quality Aspects$_{14}$ and properties of data (criterion Data$_{11}$) and behavior (group **Behavior**$_{\text{Aca}}$) and temporal properties (group **Time**$_{\text{Acb}}$) there can be further properties that can be expressed using the FDT. These properties are taken into account by this criterion.

Note, that this criterion does not belong to the group **Application Domain**$_{\text{Acd}}$ but is a direct successor of the group **Models**$_{\text{Ac}}$, see figure 5.

B: Process Oriented$_{\text{B}}$

This group comprises the criteria related to the *method* part of a formal method. This part corresponds to the process-oriented part of the context model depicted in figure 1. The criteria investigating these methodical aspects are grouped according to the kind of processes which deal with formal system descriptions. The processes are distinguished mainly whether they are used to produce formal system descriptions (**Producing**$_{\text{Ba}}$) or whether they use them as input (**Using**$_{\text{Bb}}$).

The *method* part of a formal method is seen as a set of guidelines and rules which describe what should be done and how this should be done. There are general guidelines which can be applied to any FDT. Examples of such guidelines are '*Use meaningful names*' or '*Decompose a system into manageable pieces, describe them separately and provide clear interfaces between them*'. Besides these general guidelines and rules there should be some which are tailored to the specific FDT. Such guidelines and rules are important as they allow to use an FDT in a cost-effective way.

Ba: Producing$_{\text{Ba}}$

The processes used to produce formal system descriptions can be divided roughly into those to create formal system descriptions from other descriptions and processes to change formal system descriptions. It should be clear that the separation between these processes is not a strict one. In general, when creating a formal system description it can be assumed that this is done in an iterative way. Therefore, aspects of changing descriptions have to be considered here, too. Nevertheless, it has been tried to separate these aspects as clear as possible.

Therefore, the criteria in the group **Creating**$_{\text{Baa}}$ are concerned more with producing a formal system description from other kinds of descriptions, such as e.g. an informal problem description. The criteria in the group **Changing**$_{\text{Bab}}$ are concerned more with making changes of formal system descriptions throughout regular maintenance processes.

An overview of the criteria considered in this group is given in figure 6.

Baa: Creating$_{\text{Baa}}$

27 Guidelines$_{27}$

To which extent is creating formal system descriptions supported by guidelines and rules?

There is a broad range of possible applications of guidelines and rules on the creation of descriptions. Therefore, there are several refinements of this criterion.

27.1 Process$_{27.1}$

To which extent is the process of creating a formal system description supported by guidelines and rules?

Here guidelines and rules on *how* and *when* to perform individual steps are considered.

As an example of the first kind of guidelines and rules consider the process of capturing requirements. One guideline could be to describe typical interaction sequences of the system and a user of it.

On the other hand, since creating a formal description can be considered as an iterative process or can consist of several activities performed in parallel, there should exist guidelines which describe when to start a further iteration or a new activity.

27.2 Embedding$_{27.2}$

To which extent can the guidelines and rules be embedded in existing development processes?

Most formal methods today do not cover the complete development of a system, but concentrate on specification and design. Therefore, they should allow for a smooth transition from and to other phases of the development. An example for such an embedding is the combined use of SDL [SDLa] and OMT [RBP+91] as described e.g. in [Ree96].

By this criterion it is considered, whether there are guidelines and rules which support such a transition. By the related criteria Understanding$_{42}$ and Implementation$_{43}$ it

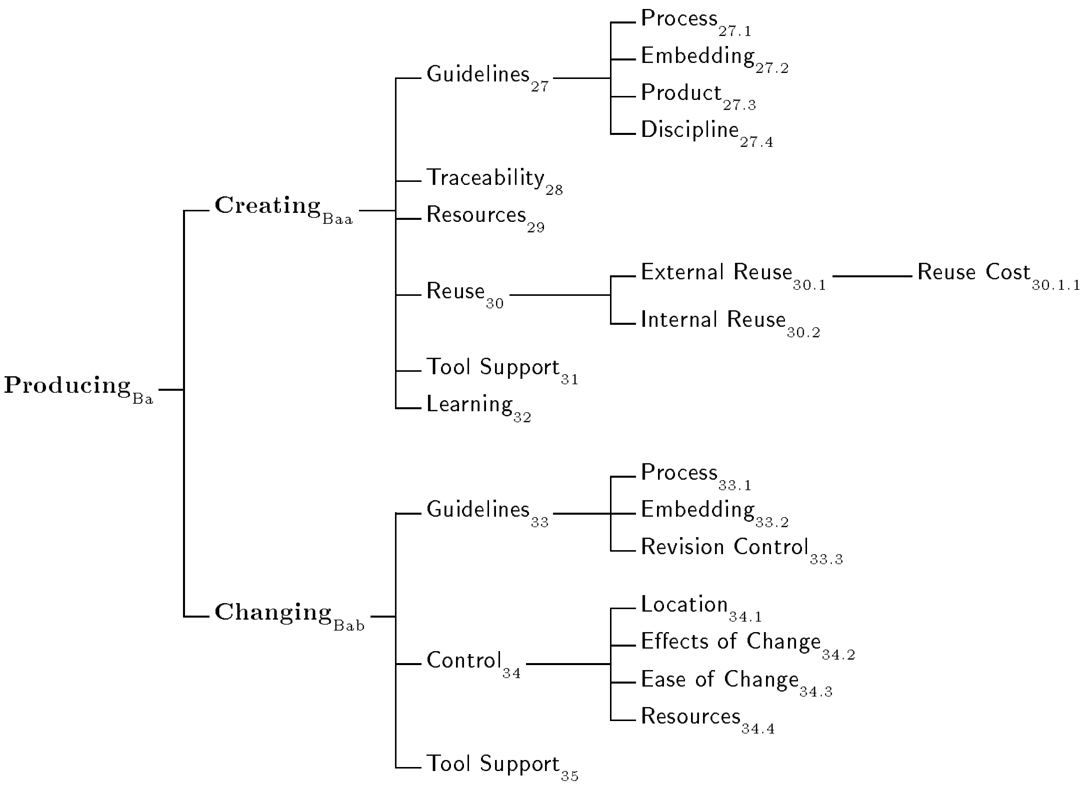

Figure 6: Structure of the group **Producing**$_{Ba}$

is considered whether the descriptions produced allow such a transition.

This criterion is important, because a formal method allowing such an embedding can be introduced in industrial practice with little additional effort besides introducing the method itself.

27.3 Product$_{27.3}$

Are there guidelines and rules on the structure and appearance of a formal system description?

Notational and structuring guidelines and rules can be an aid in creating and understanding formal system descriptions. By guidelines and rules on the structure it is made clear where in a description which kind of information is located. Notational guidelines and rules support a uniform appearance of descriptions, making it easier to understand them. For example, there exist a lot of notational rules on the appearance of descriptions written in SDL in [BH93].

Another point which can be considered here is whether there are standard ways or patterns for expressing models or properties of an often occurring type. Such a set of patterns for specifying real-time properties of systems is described in [PGK97].

27.4 Discipline$_{27.4}$

Which guidelines and rules exist whose application leads to reasonable formal system descriptions with respect to non-functional properties?

An example of such a rule is to specify the reaction of a system on every input in each state, even on the unexpected inputs. If it is applied, it contributes directly to the robustness of the implemented system. Further rules of this kind can be found e.g. in [JLHM91].

28 Traceability$_{28}$

Does the formal method support establishing a traceability relation?

The traceability relation is between parts of the documents from which a formal system description is derived and the description itself. Here, traceability between requirements stated in an informal way and their formal counterparts in a specification is considered. Traceability between requirements stated formally and those parts of design documents which describe how they should be realized are considered by criterion Traceability$_{45}$.

29 Resources$_{29}$

Are there cost models on the resources needed to create a formal system description?

This means, can it be estimated a priori how much resources are needed to create a formal system description? It should also be considered whether it is possible to supervise whether such a process is still within schedule. Resources used can be e.g. the number of persons, the total amount of time they spent on the work, or the usage of computer resources.

30 Reuse$_{30}$

Is it possible to reuse formal system descriptions or parts of them?

30.1 External Reuse$_{30.1}$

Is it possible to reuse formal system descriptions from previous development projects?

30.1.1 Reuse Cost$_{30.1.1}$

How much effort is necessary to reuse a formal system description from a previous development process?

This includes the expense of packaging a formal system description, retrieving it from a library, and modifying it such that it can be used in the current project.

30.2 Internal Reuse$_{30.2}$

Is it possible to reuse parts of formal system descriptions from the current development project?

Building up specialization hierarchies can serve as an example for expressing and using similarity between objects, thereby contributing to internal reuse. Inheritance as a concept is considered by criterion Inheritance$_{3}$.

31 Tool Support$_{31}$

Which of the guidelines and rules are supported by tools?

More detailed criteria concerning tools are collected in the group **Tools**$_{C}$.

32 Learning$_{32}$

How difficult is it to learn to create formal system descriptions?

For a more detailed exposition of this criterion see Learning$_{61}$.

Bab: Changing$_{Bab}$

Some of the criteria below are very similar to those in the group **Creating**$_{Baa}$. Therefore, their explanations are not repeated here.

33 Guidelines$_{33}$

To which extent is changing formal system descriptions supported by guidelines and rules?

33.1 Process$_{33.1}$

To which extent is the process of changing a formal system description supported by guidelines and rules?

33.2 Embedding$_{33.2}$

To which extent can the guidelines and rules be embedded in existing development processes?

33.3 Revision Control$_{33.3}$

Are there guidelines and rules with respect to the management of changes?

34 Control$_{34}$

How difficult is it to change a formal system description?

To perform a single request for change of a formal system description it is necessary to locate the parts of it which have to be changed. Furthermore, the changes have to be performed in a controlled way. This means, that it must be possible to ensure, that the changes do not affect other parts in an unforeseen way.

34.1 Location$_{34.1}$

How difficult is to locate which parts of a formal system description have to be changed?

34.2 Effects of Change$_{34.2}$

How difficult is it to predict the effects of a change on a formal system description?

34.3 Ease of Change$_{34.3}$

How easy is it to change a formal system description syntactically?

For some FDT it might be even more costly to change a description than to produce a new description from scratch. If editing of descriptions is supported by tools, then this is closely related to the quality of the tool used.

34.4 Resources$_{34.4}$

Are there cost models on the resources needed to change a formal system description?

35 Tool Support$_{35}$

Is changing a formal system description supported by tools?

Tools are investigated in more detail in the group **Tools**$_{\mathrm{C}}$.

Bb: Using$_{\mathrm{Bb}}$

The processes which are based on already produced formal system descriptions are investigated by the criteria of the groups **Validation**$_{\mathrm{Bba}}$, **Further Development**$_{\mathrm{Bbb}}$, **Verification**$_{\mathrm{Bbc}}$, **Test**$_{\mathrm{Bbd}}$, **Maintenance**$_{\mathrm{Bbe}}$, and **Documentation**$_{\mathrm{Bbf}}$.

The aspects related to maintenance processes are very similar to those of changing formal system descriptions. Therefore, the group **Maintenance**$_{\mathrm{Bbe}}$ is not given in detail, but refers simply to the group **Changing**$_{\mathrm{Bab}}$.

An overview of the criteria considered here is given in Figure 7.

Bba: Validation$_{\mathrm{Bba}}$

When creating a formal system description, it has to be ensured that this description reflects correctly the information used as input to the creation process. The term *validation* is used here for the process of checking that this is actually the case. This process cannot be seen in isolation from the process of creating a description. But because it is based on already created formal system descriptions the corresponding criteria are presented here and not in the group **Producing**$_{\mathrm{Ba}}$.

36 Requirements Validation$_{36}$

To which extent is the validation process supported by guidelines and rules?

Examples of such guidelines are descriptions of appropriate review techniques for formal system descriptions.

37 Traceability$_{37}$

Is validation supported by a traceability relation?

Validation of a formal system description is supported by providing explicitly the correspondence between elements of the formal system description and of those descriptions from which it was derived. It should be possible to use this relation in both directions, from the system description back to its ori-

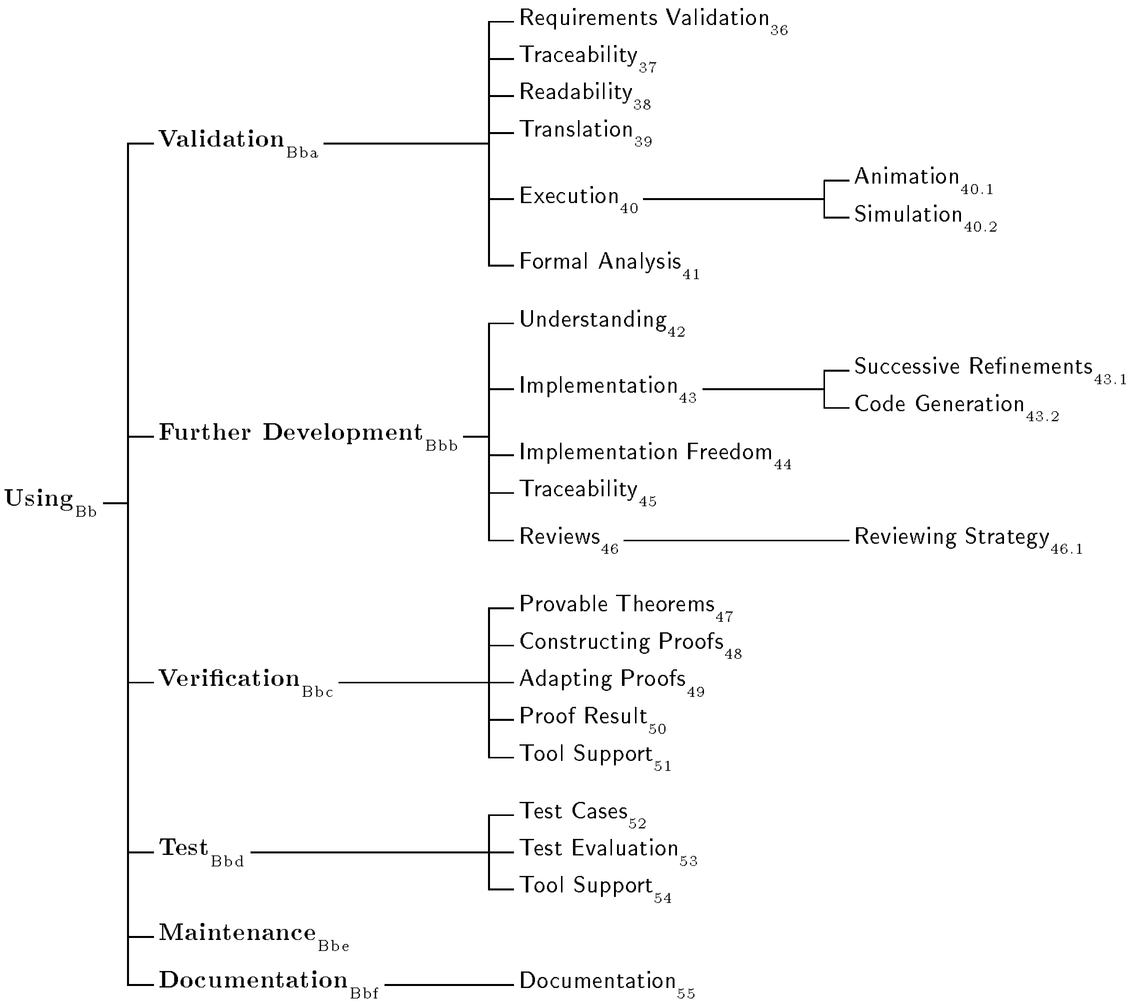

Figure 7: Structure of the group **Using**$_{Bb}$

gins, but also vice versa. This is clearly related to Traceability$_{28}$.

38 Readability$_{38}$

Which persons are able to understand a formal system description directly?

To validate a formal system description different kinds of persons besides the *experts in the method* have to be involved. These can be e.g. the *owner* of the system to be build, *users* of this system, *application experts*, *personal of sub-contractors*, and *management personal of the contractor*. Each of them has a different background on the FDT, in mathematics, engineering disciplines, in the application domain, etc. Note, that one person can belong to more than one of these groups of persons, for example the owner can also be a user of the system. The corresponding aspect of the concrete notation is investigated in criterion Notation$_{1}$.

39 Translation$_{39}$

To which languages better understood by non-experts can a formal system description be translated?

It can be assumed, that not all persons mentioned in the criterion Readability$_{38}$ are able to understand a formal system description directly. One way to make such a description understandable for them is to translate it to another language. The meaning of the description should not be changed by such a translation. Examples of such languages are *natural language* or *description techniques of the application domain*.

40 Execution$_{40}$

Can a formal system description be executed?

To validate the dynamic behavior of a system it is helpful to produce traces of the system starting from specific situations and to check these traces. This can be seen as a method to test a formal system description.

40.1 Animation$_{40.1}$

In which way can a formal system description be animated?

This means, which possibilities exist to visualize an execution of a formal system description? This can be e.g. a visualization of the currently active states of a state-transition system or a graph showing the value of a variable over time.

40.2 Simulation$_{40.2}$

Can a formal system description be linked to the environment of the system or a simulation of it?

To execute a formal system description of a dynamic system it is necessary that there is also an (executable) description of its environment, such that the control loop of the system can be closed. Instead of producing a formal description of the environment and executing it, it can be useful to link the description of the system directly to the environment or a simulation of it.

41 Formal Analysis$_{41}$

Is it possible to analyze a formal system description to derive further information?

Given a formal system description, it is sometimes useful to check whether the system described has further properties besides those stated explicitly. Examples of such properties are:

- Given a description of a communication protocol in the form of communicating processes. Is the size of buffers used in this description always below some given threshold?
- Given a description of a distributed system, again as communicating processes, which use a common resource. Is always at most one process using this resource?

As one can see from the examples given, this criteria is intended primarily for model-oriented descriptions. There, deriving further properties can be used to check whether the model describes the system as intended.

This criterion is related to the criteria in the group **FSD**$_{\mathrm{Ab}}$. It can be defined e.g. that a formal system description is consistent if the processes in the description cannot deadlock. But this is also a property in the sense used here. Furthermore, this criterion is directly related to those in the group **Verification**$_{\mathrm{Bbc}}$.

Bbb: Further Development$_{\mathrm{Bbb}}$

One of the most important purposes of producing a formal system description is that it is the starting point of the further development of the system. A design or an implementation has to be derived from such a description. Throughout the following criteria, a formal system description is mainly seen as a specification.

42 Understanding$_{42}$

Does a specification provide the information needed by a designer or programmer?

To be useful, a specification has to state the task to be solved in a clear and precise way. Furthermore, constraints on the design or implementation of the system have to be stated. This has to be done in a way which is understood by the persons involved, therefore this criterion is related to the criterion Readability$_{38}$.

43 Implementation$_{43}$

Which guidelines and rules exist to derive an implementation from a specification?

These can be guidelines and rules for deriving an implementation manually, for example:

- how to derive a hierarchy of classes in an object oriented programming language from a class model of object oriented analysis,
- how to derive an efficient implementation of a state-transition system.

43.1 Successive Refinements$_{43.1}$

Is it possible to derive an implementation by successively refining a specification?

An advantage of implementing a system by successive refinements is, that the refinement relation is usually formally defined and that corresponding proof obligations can be derived. If these can be proven, correctness of the implementation is established. As a byproduct, the refinement relation can be seen as a special kind of a traceability relation, see Traceability$_{45}$.

43.2 Code Generation$_{43.2}$

Is it possible to generate code from a specification?

In its most advanced form this means to produce good code automatically from a specification. But it is already useful if e.g. signatures of functions, class templates, etc. can be derived automatically. Besides the savings of the time needed to implement the system it is also possible to concentrate the verification efforts on the code generator instead of verifying the correctness of each system independently.

44 Implementation Freedom$_{44}$

Does the formal method preserve implementation freedom?

This means, does it preserve freedom to realize and satisfy non-functional requirements, e.g. performance or robustness requirements? This criterion can be seen as a contradiction to the criterion Implementation$_{43}$, but this is not the case. Here, it is considered, whether possible solutions are excluded already by the way the problem is stated or not.

45 Traceability$_{45}$

Is it possible to establish a useful traceability relation between a specification and an implementation?

46 Reviews$_{46}$

Is a specification useful as a reference document for reviews of an implementation?

46.1 Reviewing Strategy$_{46.1}$

Does there exist a reviewing strategy tailored to the formal method?

Bbc: Verification$_{\mathrm{Bbc}}$

The term *verification* is used here for the process of deriving properties of systems within a formal system and with mathematical rigor. Therefore it is not a direct counterpart to *validation*. Essentially the proof of correctness of an implementation with respect to a specification is considered here, other aspects of verification are considered in the criterion Formal Analysis$_{41}$.

47 Provable Theorems$_{47}$

What can be proven?

The theorems to be proven usually state properties of the system. These properties can refer to each of the different kinds of models of the system mentioned in the group **Models**$_{\mathrm{Ac}}$.

48 Constructing Proofs$_{48}$

How much ingenuity is needed to construct a proof?

In some formal systems it is possible to derive proofs automatically, in other ones it is in general impossible or intractable to find proofs automatically. In these cases it should be possible to prove simple properties in a routine way without requiring much ingenuity of the person carrying out the proof.

49 Adapting Proofs$_{49}$

How difficult is it to adapt a proof?

Often proofs reveal errors in the descriptions of a system or in the properties to be proven. These descriptions have to be changed accordingly and these proofs and other already succeeded ones need to be redone. In this case it is helpful if the proofs can be adapted to the changed descriptions in an easy way or even better if it is possible to redo them automatically.

50 Proof Result$_{50}$

What is the result of a proof?

The most basic result of a proof should be whether the theorem to be proved is correct or not. But further information is desirable. In the case of a positive answer, one can ask, whether a proof gives additional insight into the system and the reasons why the theorem holds. In the case of a negative answer it is helpful if one can come up with a counterexample or if it is possible to make clear why the proof failed. For example, it would be extremely helpful if it is possible to trace this back to e.g. a wrong assumption. This is related to the criteria Location$_{9.1.2}$ and Location$_{10.1.2}$.

51 Tool Support$_{51}$

In which way is verification supported by tools?

More detailed questions concerning tools can be found in the group **Tools**$_{C}$.

Bbd: Test$_{Bbd}$

Testing is a common technique to search for errors in an implementation. It can be supported by formal system descriptions by using them as a clear and precise reference for evaluating tests and for generating test cases in a systematic way.

52 Test Cases$_{52}$

Is it possible to derive test cases from a formal system description?

For example, if the system is described by a state-transition system, then there should be testcases corresponding to the initial states and to transitions between states.

53 Test Evaluation$_{53}$

Is it possible to use a formal system description to evaluate tests?

This requires that, given a test case and the result of performing it, it is possible to give a definitive answer whether the result of this test corresponds to the formal system description.

54 Tool Support$_{54}$

Which of these tasks are supported by tools?

More detailed questions concerning tools can be found in the group **Tools**$_{C}$.

Bbe: Maintenance$_{Bbe}$

See group **Changing**$_{Bab}$.

Bbf: Documentation$_{Bbf}$

55 Documentation$_{55}$

Can a formal system description serve as a reference document of a system?

This means, can a formal system description be the basic source of information about a system or is it necessary to resort to other descriptions, e.g. the code itself?

C: Tools$_{\text{C}}$

Tools play a crucial role to make tasks like syntax-checking less tedious or tasks like verification less error-prone. Therefore, tools can help a lot when applying a specific formal method. Furthermore, insufficient tool support is considered an impediment to introducing formal methods into industrial practice.

Since this group is referenced several times in this catalogue, it is necessary to apply the criteria in this group to each of the corresponding aspects. This does not exclude, that when appropriate, several of them can be considered together.

The structure of this group is depicted in figure 8.

56 FDT Support$_{56}$

Which syntactic features of the FDT are handled by the tool?

This criterion investigates whether the tool does support all syntactical elements or not.

57 Process Support$_{57}$

Which activities are supported by the tool?

A possible answer can be that the tool is just a comfortable (syntax oriented) editor. All activities supported by the tool should be enumerated in the answer. See also the criterion Tool Support of the following criteria and groups: Internal Completeness$_{9}$, Internal Consistency$_{10}$, **Creating**$_{\text{Baa}}$, **Changing**$_{\text{Bab}}$, **Verification**$_{\text{Bbc}}$, and **Test**$_{\text{Bbd}}$.

58 Tool Maturity$_{58}$

How mature is the tool?

The answer shall point out whether the tool does what it is expected to do.

58.1 Bugs$_{58.1}$

How often does the tool reveal a bug?

58.2 Interface Evaluation$_{58.2}$

How good is the user interface?

Important aspects are:

- Compliance to interface standards.
- User guidance.
- Learning effort. See also Learning Tool Interface$_{59.1}$.

58.3 Tool Feature Evaluation$_{58.3}$

Which features are particularly good or bad?

59 Learning$_{59}$

How much time is necessary to learn to use the tool efficiently?

59.1 Learning Tool Interface$_{59.1}$

To which extent is the learning of the tool supported by the conceptual design of the tool and the user interface?

Compliance with standards for user interfaces makes learning easier and is one point to be taken into consideration here. See also Interface Evaluation$_{58.2}$.

59.2 Learning Support$_{59.2}$

Which kinds of support for learning the tool exist?

See Learning Support$_{61.3}$.

60 Data Interchange$_{60}$

Which part of information can be interchanged between different tools?

Information to be interchanged can be:

textual information: A representation of the content of a formal system description (e.g. a postscript file containing a state diagram).

semantic information: The content of a formal system description (e.g. a list of states and transitions).

visual information: The spatial arrangement and connectivity of graphical

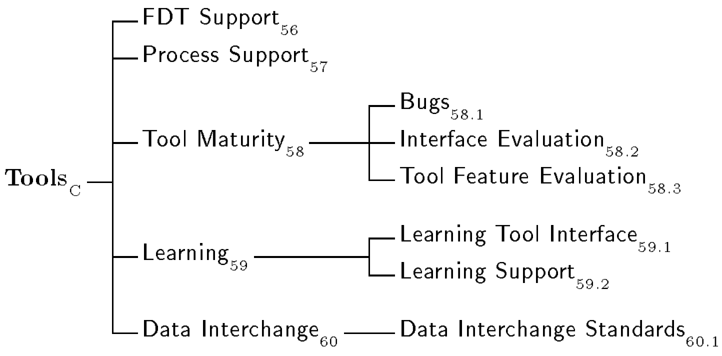

Figure 8: Structure of the group **Tools**$_{C}$

elements contained in a formal system description.

60.1 Data Interchange Standards$_{60.1}$

Which standards to exchange data are supported?

Examples of such standards are FrameMaker-documents, Tex-documents or standards for the particular formal method.

D: Persons$_{D}$

61 Learning$_{61}$

How difficult is it to become familiar with a formal method?

In the first two refinements of this criterion the difficulty is related to the time needed to understand the concepts and to learn to produce good formal system descriptions, respectively. As long as no controlled experiments have been conducted, the attributes can only be assigned subjectively.

61.1 Understanding the FDT$_{61.1}$

How long does it take to understand the concepts of the FDT?

61.2 Creating Descriptions$_{61.2}$

How long does it take to learn to produce good formal system descriptions?

61.3 Learning Support$_{61.3}$

Which kinds of support for learning the formal method exist?

Examples of attribute values are: *text books*, *experience reports*, *courses*, *online tutorials*.

5 Evaluation of Formal Methods

In this section we will give an exemplary application of the criteria catalogue. First we describe the tailoring of the criteria catalogue in section 5.1, then we give a short introduction to the considered formal methods in section 5.2. In section 5.3 we are recording the experiences made using these formal methods. Finally, we compare the formal methods with respect to selected criteria in section 5.4.

5.1 Tailoring of the Criteria Catalogue

In the following the catalogue is tailored as described in section 3.

Purposes and Context
The *main purpose* for which we want to use the criteria catalogue is to *record* in a structured way the experiences we made using the formal methods SDL, statecharts, and a temporal logic for specifying control systems in the domain building automation systems. Based on this, it is possible to compare them with respect to the context in which they have been utilized. This *comparison* is our *second purpose*, but it is only of minor importance.

The context in which we utilized the three formal methods has been the specification of control systems in the domain building automation systems. Among the tasks of a building automation system are the control of temperature, humidity, light, air flow, security, and safety. For specifying control systems in this domain it is substantial to be able to express the behavior of a system, i.e. functional requirements, and temporal properties. According to our experience the ability to express data is of minor importance. Probably, this is caused by the fact that we have considered only the control part of a system and not for example management aspects or a (graphical) interface to the user. Note, that this experience is not related to a specific formal method, but made with respect to the application domain.

Adaption to the Main Purpose
When adapting the proposed criteria catalogue to our main purpose it was not necessary to refine criteria; only irrelevant criteria had to be deleted. The deleted criteria deal with activities we have not performed up to now, so that there exists no experience that can be recorded. Furthermore, most of the criteria for which only subjective experience was available have not been considered. In order to demonstrate the problem of subjective information the criteria shown in table 1 have not been excluded. The following list contains the criteria or groups of criteria we will not consider:

- criteria External Reuse$_{30.1}$ and Reuse Cost$_{30.1.1}$
  Since there were no previous projects, reuse was not possible in our case studies.
- group **Changing**$_{\mathrm{Bab}}$
  This group is deleted because we did not have to change a description due to new requirements, but only in order to eliminate detected errors or to improve the readability and intelligibility of a description. By the criteria of the group **Changing**$_{\mathrm{Bab}}$ mainly modifications of the first kind are considered.
- groups **Further Development**$_{\mathrm{Bbb}}$, **Verification**$_{\mathrm{Bbc}}$, **Test**$_{\mathrm{Bbd}}$, **Maintenance**$_{\mathrm{Bbe}}$, and **Documentation**$_{\mathrm{Bbf}}$
  The criteria of these groups investigate processes we did not perform in the case studies.
- groups **Tools**$_{\mathrm{C}}$ and **Persons**$_{\mathrm{D}}$,
  criteria Tool Support$_{9.1.3}$, Tool Support$_{10.1.3}$, Tool Support$_{31}$, and Learning$_{32}$
  Only subjective information is available.

The set of possible attribute values of those criteria for which we have only subjective information is shown in table 1. By these criteria it is investigated to which extent the corresponding processes are supported by guidelines and rules. The attribute value *average* means that the processes are supported to an extent which was considered helpful by the authors.¶

criteria	possible attribute values
Guidelines$_{27}$, Process$_{27.1}$, Embedding$_{27.2}$ Requirements Validation$_{36}$	{ *full*, *large*, *average*, *small*, *no* }

Table 1: Assignment of attribute values to criteria

Adaption to the Second Purpose

First, we have to decide which criteria are taken into account when comparing the formal methods. Obviously, this must be a subset of the criteria considered for the recording of the experience. Furthermore, only criteria for which different attributes can be compared with each other are applied. The following list contains all the criteria for which there is no meaningful ordering on the set of possible attributes with respect to our second purpose:

Notation$_{1}$, Specification Style$_{5}$, Traces$_{15}$, Behavior Description$_{16}$, Granularity of Time$_{18.1}$, Global Time$_{19}$, Parallelism$_{20}$, Communication$_{21}$, Synchronization$_{21.1}$, Animation$_{40.1}$.

Assessment

For the remaining set of criteria we now present for each criterion a partial order on its set of possible attributes. Most of the partial orders are simple and obvious. But there are also some which result from our observation which models and properties are relevant or not so relevant in our application domain.

Normally, we will use the notation $a_1 > a_2$ to denote that attribute a_1 is better suited than attribute a_2 with respect to the focus of our comparison. In the case of a yes-no-question the relation is always *yes*>*no*. Criteria of this category are:

Modularity$_{2}$, Interface$_{2.1}$, Combination of Notations$_{6}$, Abstraction$_{7}$, Refinement$_{7.1}$, Orthogonality$_{8}$, Compatibility$_{8.1}$, Internal Completeness$_{9}$, Decidability$_{9.1}$, Location$_{9.1.2}$, Incompleteness$_{9.2}$, Internal Consistency$_{10}$, Decidability$_{10.1}$, Location$_{10.1.2}$, Inconsistency$_{10.2}$, Priority$_{23}$, Product$_{27.3}$, Discipline$_{27.4}$, Traceability$_{28}$, Resources$_{29}$, Reuse$_{30}$, Internal Reuse$_{30.2}$, Traceability$_{37}$, Execution$_{40}$, Simulation$_{40.2}$, Formal Analysis$_{41}$.

There are also a lot of criteria for which the attributes assigned by a criterion usually contains more than one element of the set of possible attribute values. In this case we will use the principle *"the more the better"* to denote that the ordering relation among the possible attributes is based upon the number of attribute values contributing to a attribute. Criteria of this category are:

Data$_{11}$, Interface Model$_{12}$, Structure$_{13}$, Quality Aspects$_{14}$, General Temporal Properties$_{17}$, Metric Temporal Properties$_{18.2}$, Fairness$_{22}$, Domain Models$_{24}$, Application of Domain Models$_{25}$, Further Properties$_{26}$, Readability$_{38}$, Translation$_{39}$.

¶One can see here, that it is necessary to define the attribute values more precisely in order to be able to characterize and evaluate a formal method in an objective way.

For the following criteria the set of possible attribute values is given in table 1. The attributes are ordered in the obvious way.

Guidelines$_{27}$, Process$_{27.1}$, Embedding$_{27.2}$, Requirements Validation$_{36}$.

Finally, we list the partial orders for those criteria which can not be assigned to one of the categories mentioned so far:

- Inheritance$_{3}$:
  single>no inheritance and *multiple>no inheritance*
  This order is established since we think inheritance is a potentially good structuring concept. However, it is unclear whether the larger expressiveness of multiple inheritance outweighs its larger complexity; therefore *single* and *multiple inheritance* are not ordered.

- Degree of Formality$_{4}$:
  formal>rigorous>informal>no way
  This order is used, since we made the experience, that every time a syntactical feature is not well defined, this is a source of confusion and time consuming discussions because of different interpretations of the persons working together.

- Technique$_{9.1.1}$ and Technique$_{10.1.1}$:
  Here the principle *the more the better* is applied and if two sets are incomparable with respect to this principle the relation *automatic proofs>manual proofs* is used if possible.
  We do not compare *reviews* with *automatic* or *manual proofs* because these are completely different techniques. However, if the tool is mature an automatic proof is more reliable than a manual one.

- Model of Time$_{18}$:
  quantitative>qualitative
  This order results from the models needed in our application domain, where quantitative timing requirements have to be expressed.

Due to these partial orders it is possible to compare the formal methods with respect to one criterion. In order to get a more global comparison of the considered formal methods we classify the criteria into the three classes: *very important*, *important*, and *not so important*. We decided to utilize only this coarse grained rating, since we have not enough information to establish a finer classification. The importance of a criterion corresponds to the relevance of the investigated aspect with respect to the application domain considered here.

Very important criteria are:

General Temporal Properties$_{17}$, Model of Time$_{18}$, Metric Temporal Properties$_{18.2}$, Priority$_{23}$, Readability$_{38}$.

The criteria dealing with time are classified as *very important* since timing aspects play a substantial role in our application. The ability to express priority is very important since we made the experience in our application domain that there is a large number of conflicting requirements which could be resolved by priority. Readability is regarded as very important since several different groups of persons are involved in the developing of a building automation system, especially persons that are not familiar with any formal method.

As already mentioned, according to our experience the ability to express data is of minor importance in this application domain. Therefore the criterion Data_{11} is classified as *not so important*.

All other criteria are classified as *important*.

5.2 Introduction to the Formal Methods Evaluated

5.2.1 Statecharts

Statecharts [Har87] are an extension of state-diagrams by the concepts *depth*, *orthogonality*, and *broadcast-communication*. By *depth* hierarchy is added to state-diagrams. Thereby it becomes possible to view a state-diagram on several levels of abstraction. By *orthogonality* it becomes possible to write down concurrently running automata separately instead of their product automaton. By the term *broadcast-communication* it is described how information is exchanged between such concurrent automata, i.e. information is not send to a specific automata, but presented simultaneously to all of them. This is done instantaneously, thereby following the so-called *synchrony hypothesis*, see e.g. [BG92]. Furthermore, it is possible to refer to data and to use timeouts to describe metric temporal aspects.

Statecharts describe the temporal behavior of automata. They are complemented by two other types of charts: *Activity-charts* for describing the data-flow between the concurrent automata and *module-charts* for describing the structural decomposition of a system. These types of charts, together with the corresponding tool, are described in [HLN+90]. In the following, all three kinds of charts are subsumed under the term statecharts.

Due to the extensions, the semantics of statecharts is not easy to define. In literature, various different semantics are proposed, for an overview see [vdB94]. The one described in [HN96] can be considered the 'official' semantics of statecharts. In the case studies performed by the authors an older version of the semantics [stm91] is used, but this is very similar to [HN96].

5.2.2 SDL

SDL (Specification and Description Language) is an FDT for the specification of discrete reactive systems. We are considering experiences made with SDL-92 which was standardized by the ITU (International Telecommunication Union) in 1993 [SDLa]. Two notations for the description of behavior are defined: a graphical one (SDL-GR) and a textual one (SDL-PR). We only used the graphical notation. In addition there is a mathematical notation for abstract data types.

In an SDL-description a system is divided into so-called blocks. These blocks are connected via channels to each other and to the environment. Every block can be built from other blocks. On the lowest level each block consists of a set of so-called processes. Those processes are connected to each other and to processes of other blocks via signal routes. On these signal routes typed messages are transmitted without loss and without any delay in FIFO-order. The behavior of a system is described using communicating extended finite state machines. There is an unbounded FIFO-input-buffer for each process. It is possible to save specific signals in the input queue. Also it is possible to read signals from the queue with a higher priority. All SDL processes of one system proceed concurrently.

The process model we used is the one described in [BH93], but see also [OFMP+94] and [Ree96]. There, in addition to SDL, other FDTs are used, especially MSC (Message Sequence Charts) which are tightly connected to SDL and standardized by the ITU in 1993 [MSC], too.

5.2.3 Temporal Logics

Temporal logics are widely used for specifying reactive systems. A large number of different kinds of temporal logics have been developed in the last twenty years since Pnuelis pioneering work [Pnu77]; see for example [AH90, CHR91, CES86, Lam94, SMSV83]. Usually a temporal logic is based upon a classical one, like the propositional or first order logic, which is extended by special temporal operators.

The temporal logic we consider here is called *tailored real time temporal logic (tRTTL)* [KPG96]. It is a *propositional*, *linear time* temporal logic whose time domain is the set of non-negative real numbers, and it is based on the temporal logics described in [MP92, MP93, AH90]. During a case study performed in the context of the SFB 501, i.e. within the application domain building automation systems, new tailored operators have been developed in order to improve the readability and intelligibility of the formulae and in order to increase the expressiveness of the logic. Furthermore, based on these new operators so called *patterns* have been developed. These are parameterized formulae representing typical classes of requirements we detected in the considered application domain. Such patterns on the one hand allow to formalize efficiently informally given requirements by instantiation; on the other hand it is easily possible to translate the resulting formulae back to natural language in a uniform way. This work resulted in the definition of tRTTL which is described and analyzed in [KPG96].

5.3 Evaluation

SDL	statecharts	tRTTL
Product Oriented$_{\mathrm{A}}$		
FDT$_{\mathrm{Aa}}$		
Syntax$_{\mathrm{Aaa}}$		
Notation$_1$: *Which notation-style is provided by the FDT?*		
textual, *graphical*, in addition *mathematical* to describe abstract data types.	Mainly *graphical*. To describe conditions and actions a simple iterative programming language is available. This allows quantification over finite domains, i.e. over arrays.	*mathematical*
Structuring Concepts$_{\mathrm{Aaaa}}$		
Modularity$_2$: *Can a formal description be structured and decomposed into manageable parts?*		
yes	*yes*	*no*

SDL	statecharts	tRTTL
Interface$_{2.1}$: *Is the notion of an interface supported as a syntactical construct by the FDT?*		
yes, restricted to the relation between channels and blocks and signalroutes and processes, respectively.	*no*, the information is distributed over various syntactical objects.	*Does not apply.*
Inheritance$_{3}$: *Which kind of inheritance is supported by the FDT?*		
single	Not supported directly. *Parameterization* is supported by so-called *generic* charts.	*no inheritance*
Semantics$_{\mathrm{Aab}}$		
Degree of Formality$_{4}$: *In which way is the semantics of the FDT defined?*		
rigorous in [SDLa], *formal* in [SDLb]; the formal overrules the rigorous one in case of differences. Both are intended for different users; the rigorous one is being judged to be easier comprehensible (see [SDLb]).	*rigorous*	*formal*
Specification Style$_{5}$: *Which specification style is supported by the FDT?*		
model oriented (behavior); *property oriented* and *model oriented* (data)	*model oriented*	*property oriented*
FSD$_{\mathrm{Ab}}$		
Combination of Notations$_{6}$: *Is it possible to use different notations in one formal system description?*		
yes	Module-, activity-, and statecharts can be used together in a complementary way. There is only one language for each of the corresponding aspects.	*no*
Abstraction$_{7}$: *Is it possible to look at a system on various levels of abstraction in one formal system description?*		
yes	*yes*	*yes*
Refinement$_{7.1}$: *Is there a refinement relation between formal system descriptions on different levels of abstraction?*		
yes	*yes*	*no*
Orthogonality$_{8}$: *Is it possible to have different views on a system on the same level of abstraction in one formal system description?*		
yes, but only between different kinds of models. By using MSCs the behavior of a system observed at different interfaces of it can be described separately.	*yes*, but only between different kinds of models.	*yes*
Compatibility$_{8.1}$: *Is there a notion of compatibility between different views of one system (expressed in the same formal system description)?*		
yes, see Orthogonality$_{8}$.	*yes*, but only between different kinds of models.	*yes*: logical consistency

SDL	statecharts	tRTTL
Internal Completeness$_9$: *Is there a notion of internal completeness of formal system descriptions defined?*		
yes, based on syntax, i.e. are all variables defined, are all inputs consumed somewhere, etc.		*no*; therefore the following criteria do not apply.
Decidability$_{9.1}$: *Is it possible to decide whether a formal system description is internally complete?*		
yes	*yes*	Does not apply.
Technique$_{9.1.1}$: *By which technique can it be decided whether a formal system description is internally complete?*		
syntactical checks	*syntactical checks*	Does not apply.
Location$_{9.1.2}$: *Is it possible to locate the incomplete parts of a formal system description?*		
yes	*yes*	Does not apply.
Incompleteness$_{9.2}$: *Is it possible to work in a useful way with a formal system description, which is not internally complete?*		
yes	*yes*	Does not apply.
Internal Consistency$_{10}$: *Is there a notion of internal consistency of formal system descriptions defined?*		
yes: again based on syntax.	*yes*: again based on syntax.	*yes*: logical consistency, i.e. satisfiability.
Decidability$_{10.1}$: *Is it possible to decide whether a formal system description is internally consistent?*		
yes	*yes*	We have not proved this up to now.
Technique$_{10.1.1}$: *By which technique can it be decided whether a formal system description is internally consistent?*		
syntactical checks	*syntactical checks*	*manual proof*
Location$_{10.1.2}$: *Is it possible to locate the inconsistent parts of a formal system description?*		
yes	*yes*	*yes*
Inconsistency$_{10.2}$: *Is it possible to work in a useful way with a formal system description, which is not internally consistent?*		
yes	*yes*	*yes*
Models$_{Ac}$		
Data$_{11}$: *What aspects of data can be expressed?*		
abstract data types, *refinement*, *specialization*; in addition there exist some predefined data types like records and arrays and the possibility to give a subrange for numerical data types.	*no abstract data types*; only some predefined data types, records and arrays are supported; *no refinement*, *no specialization*.	*none* Data is not supported directly.
Interface Model$_{12}$: *What aspects of an interface between components can be described?*		
data type, *monitored/controlled*, *properties of abstract data types*	*data type*, *monitored/controlled*, *magnitude* (in the data dictionary), *time* (by the statecharts). Note, that components are described by activities.	Since Modularity$_2$ is not supported, this criterion does not apply.
Structure$_{13}$: *What structural aspects of a system can be expressed?*		
hierarchical, *flow of data*	*hierarchical*, *flow of data*	*none*

SDL	statecharts	tRTTL
Quality Aspects$_{14}$: *What quality aspects can be expressed?*		
none directly. Certain effects can be modeled through non-determinism e.g. channels which can lose data.	*none*	*none*
Behavior$_{\mathrm{Aca}}$		
Traces$_{15}$: *Which kind of elements constitute the observable traces?*		
states, *signals*	the *configuration* of the statecharts, i.e. which states are currently active; *events*, *values of variables*.	*states*
Behavior Description$_{16}$: *How can the dynamic behavior of the system or its components be described?*		
whitebox	*whitebox*	*blackbox*, *whitebox*
Time$_{\mathrm{Acb}}$		
General Temporal Properties$_{17}$: *What kind of general temporal properties can be expressed?*		
safety, but note, that properties cannot be stated directly, because SDL and statecharts are model oriented description techniques.		*safety*, *liveness*
Model of Time$_{18}$: *In which way is time represented?*		
mainly *qualitative*, in a restricted way also *quantitative* through the use of timers.	mainly *qualitative*, in a restricted way also *quantitative*.	*quantitative*
Granularity of Time$_{18.1}$: *What granularity has the time domain?*		
dense	*discrete*	*dense*
Metric Temporal Properties$_{18.2}$: *What kinds of metric temporal properties can be described?*		
timeouts	*timeouts*	*bounded response*, *minimal separation*, *duration*
Global Time$_{19}$: *Is time considered as being global or is time considered local to individual components?*		
local with respect to processes: timer-concept	There are two different semantics in [HN96]: in one of them time can proceed for each component, i.e. activity, independently; in the other one time proceeds at the same pace for all components.	*global*
Concurrency$_{\mathrm{Acc}}$		
Parallelism$_{20}$: *How are traces of concurrently proceeding components composed?*		
interleaving	*maximal parallelism*	Since Modularity$_2$ is not supported there are no different components. Therefore the criteria of group **Concurrency**$_{\mathrm{Acc}}$ do not apply.

SDL	statecharts	tRTTL
Communication$_{21}$: *What communication-concept between components is used?*		
message passing, *remote procedure calls*, *remote variables*; *shared variables* via reveal/view-construct (not recommended).	*shared variables*, *events* as a simple form of messages or signals.	Does not apply.
Synchronization$_{21.1}$: *Is communication between components synchronous or asynchronous?*		
asynchronous	Does not apply.	Does not apply.
Fairness$_{22}$: *What kind of fairness is supported?*		
none	*none*	Does not apply.
Priority$_{23}$: *Is it possible to express priority between components?*		
no, except it is possible to express priority on signals.	*yes*	Does not apply.
Application Domain$_{\mathrm{Acd}}$		
Domain Models$_{24}$: *Which models of the application domain can be expressed in the considered FDT?*		
structure, *data*, *interface*, *behavior*	*structure*, *data*, *interface*, *behavior* as described above.	*behavior*, especially timing aspects
Application of Domain Models$_{25}$: *How can formal descriptions and models of the application domain be combined?*		
same	Models of the *same* kind can be combined simply by considering them as concurrent modules, activities, and statecharts, respectively.	*same*
Further Properties$_{26}$: *What further properties can be described?*		
none, see remark to Quality Aspects$_{14}$	*none*	*none*
Process Oriented$_{\mathrm{B}}$		
Producing$_{\mathrm{Ba}}$		
Creating$_{\mathrm{Baa}}$		
Guidelines$_{27}$: *To which extent is creating formal system descriptions supported by guidelines and rules?*		
full	see refinements of this criterion.	*small*, only for the usage of *patterns*, see the following refinements of this criterion.
Process$_{27.1}$: *To which extent is the process of creating a formal system description supported by guidelines and rules?*		
full	*small*, there are only few guidelines and rules specific for statecharts, but more general ones can be applied to a large extent.	*small*, since there are only a few guidelines how to use the *patterns* based on the introduced tailored operators [PGK97].
Embedding$_{27.2}$: *To which extent can the guidelines and rules be embedded in existing development processes?*		
full	No experience gained in the case studies; statecharts and tRTTL have been used 'stand alone', i.e. without relation to other defined development processes.	

SDL	statecharts	tRTTL
Product$_{27.3}$: *Are there guidelines and rules on the structure and appearance of a formal system description?*		
yes	*yes*, there are some which already help a lot.	*yes* : The introduced patterns indicate the structure of a single formula.
Discipline$_{27.4}$: *Which guidelines and rules exist whose application leads to reasonable formal system descriptions with respect to non-functional properties?*		
General ones can be applied, more specific ones are proposed e.g. in [BH93].	General ones can be applied, see e.g. [JLHM91].	*none*
Traceability$_{28}$: *Does the formal method support establishing a traceability relation?*		
yes, but it is not explained how to really establish it.	*no*, this is not supported directly. But it is possible to do so manually or by using other tools.	*yes*, by translating back the instantiated patterns to natural language.
Resources$_{29}$: *Are there cost models on the resources needed to create a formal system description?*		
no	*no*	*no*
Reuse$_{30}$: *Is it possible to reuse formal system descriptions or parts of them?*		
yes	*yes*	*yes*
Internal Reuse$_{30.2}$: *Is it possible to reuse parts of formal system descriptions from the current development project?*		
yes	*yes*, e.g. by generic charts.	*yes*, e.g. by utilizing the patterns.
Using$_{\mathrm{Bb}}$		
Validation$_{\mathrm{Bba}}$		
Requirements Validation$_{36}$: *To which extent is the validation process supported by guidelines and rules?*		
no	*no*	*no*
Traceability$_{37}$: *Is validation supported by a traceability relation?*		
no, see Traceability$_{28}$.	*no*, see Traceability$_{28}$.	*no*, see Traceability$_{28}$.
Readability$_{38}$: *Which persons are able to understand a formal system description directly?*		
only experts; *most people* can understand the graphical description if the symbols are explained once.	When using only the basic concepts, *most people* can understand statecharts. When using the more advanced ones, then only *experts in the method*.	The formulae can only be understood by *experts in the method*. The translation in natural language should be understandable by *everyone*.
Translation$_{39}$: *To which languages better understood by non-experts can a formal system description be translated?*		
none	*none*	The introduced patterns can and have been easily translated in *natural language*. For an arbitrary formula this can not be done in general.
Execution$_{40}$: *Can a formal system description be executed?*		
yes	*yes*	*no*

SDL	statecharts	tRTTL
Animation$_{40.1}$: *In which way can a formal system description be animated?*		
The evolution of the system can be followed graphically by looking at the current states and values of variables. The sequence of signals can be displayed by creating MSCs.	graphical frontends, so called *panels*, can be attached to statecharts.	In no way.
Simulation$_{40.2}$: *Can a formal system description be linked to the environment of the system or a simulation of it?*		
yes	*yes*	*no*
Formal Analysis$_{41}$: *Is it possible to analyze a formal system description to derive further information?*		
yes, the usual properties of communicating automata, e.g. reachability of certain states or existence of deadlocks, can be checked for.		We think that it is possible to derive for example *timing constraints*, but we have no experience.

5.4 Summary of the Comparison

The focus of the answers presented in section 5.3 is on the main purpose of the sample application of the criteria catalogue, namely the recording of experience. Based on these answers and the assessment described in section 5.1 the second purpose, the comparison of the three formal methods, is considered in this section. In accordance with the classification of the criteria, first the *very important* criteria are analyzed, next some *important* criteria are considered for which there are differences among the three formal methods, a brief remark is made to the *not so important* criterion Data$_{11}$.

With respect to the three *very important* criteria dealing with time – General Temporal Properties$_{17}$, Model of Time$_{18}$, Metric Temporal Properties$_{18.2}$ – the temporal logic is better than SDL and statecharts: in the case of general and metric temporal properties more properties are expressible in tRTTL and concerning the criterion Model of Time$_{18}$ the better rated one, i.e. a quantitative time model, is supported. SDL and statecharts do not differ with respect to these time related criteria. This is not the case with respect to the *very important* criterion Priority$_{23}$. While in statecharts it is possible to express priority, SDL offers no comparable mechanism. For tRTTL this criterion is not applicable since modularity concepts are not supported. With respect to the *very important* criterion Readability$_{38}$, SDL and statecharts are rated better than tRTTL. Note, that the attributes are assigned on the subjective experience of the authors and have to be confirmed by controlled experiments, see e.g. [Bas96].

Due to the large number of *important* criteria, here only a few of them are discussed that reveal differences among the formal methods. With respect to the criterion Degree of Formality$_4$ tRTTL and SDL are better than statecharts, since we have established the ordering *formal>rigorous*.

For the criteria Modularity$_2$, Interface$_{2.1}$, Inheritance$_3$, Abstraction$_7$, and Refinement$_{7.1}$ the attributes assigned to SDL and statecharts are rated better or equal than those assigned to tRTTL. The same is valid for the criteria Interface Model$_{12}$ and Structure$_{13}$ of the group **Models**$_{\mathrm{Ac}}$ and the *not so important* criterion Data$_{11}$. Either the questions of these criteria are answered with *yes* or in the case of the *the more the better* criteria the number of assigned attribute values are better for SDL and statecharts. In the group $\mathbf{FSD}_{\mathrm{Ab}}$ the main difference is that there is a syntactically defined notion of completeness for SDL and statecharts while this criterion has to be answered with *no* for tRTTL.

Up to now, usually SDL and statecharts have been considered together and compared with tRTTL on the other side. An obvious difference between SDL and statecharts is revealed by the criteria of the process oriented part of the criteria catalogue. For SDL a large number of *specific* guidelines and rules exists [BH93], which is not the case for statecharts. For tRTTL few guidelines concerning the introduced patterns exist.

Nearly all attributes assigned to SDL with respect to criteria of the group $\mathbf{Process\ Oriented}_{\mathrm{B}}$ are better or equal than those assigned to statecharts and tRTTL (according to the partial orders established in section 5.1). Only in the case of the criterion $\mathsf{Translation}_{39}$ tRTTL is rated better than SDL (and statecharts).

Summing up this comparison, the most essential and obvious fact is that the two formal methods SDL and statecharts are very similar and that both differ to a high degree from the considered specific temporal logic. Furthermore, the strengths and weaknesses of the formal methods, that have been pointed out, can be hints on directions, in which it is useful to improve a formal method. Some examples are:

- Improve the expressiveness of statecharts and SDL with respect to temporal properties. This is already an area of active research, see e.g. [KP92, Dei96, Leu95]. Most temporal properties can already be described by using timeouts, but not in a very clear way. Therefore, more suitable temporal operators should be available.
- There are few guidelines and rules which are specific for statecharts. But, as can be seen from the answers given, statecharts and SDL are very similar. Therefore it seems to be possible to adapt the guidelines and rules of SDL to statecharts.
- Traceability should be improved for each of the formal methods examined. Therefore it is necessary to define precisely, which elements of a formal system description are involved in the traceability relation.
- Structuring concepts should be introduced in tRTTL or it should be combined with a FDT providing such concepts.

6 Concluding Remarks

A large set of criteria has been presented in this paper. They are intended for formal methods for specifying and designing reactive systems. The notion of a criterion has been discussed and several problems of its use in this catalogue have been identified. The problems arise because the criteria presented are still too general to be used in a specific context. Furthermore, to give an example of the application of the catalogue, three formal methods, namely SDL, statecharts, and a temporal logic called tRTTL, have been evaluated.

The structure of the criteria catalogue according to the Concept-Model of formal methods has been helpful when applying the catalogue as described in section 5. But there are further possibilities to structure the catalogue, e.g. the process oriented part can be structured more directly towards the different phases of software development. By further applications of the catalogue it can be supported, that the structure used here is a helpful one, or suggestions how to improve the structure can be derived.

After further applications of the catalogue, it can be scrutinized, whether the different aspects of formal methods are investigated with an appropriate level of detail. It can also be reviewed, whether all criteria really contribute to an evaluation of formal methods, i.e. whether they actually reveal differences between different formal methods. Note, that this will be useful mainly within one specific context of the application of the formal methods evaluated.

It is planned to extend the evaluation of SDL, statecharts, and tRTTL in two directions. On the one hand, more criteria concerning the processes shall be considered and the answers which are based on the subjective experience of the authors shall be made more objective. On the other hand, work is in progress, see [BDK+97], to apply the catalogue to the SCR description technique, see [CP93, HJL96][||]. This application is within the same context, but done independently of the authors of this report. Therefore, this contributes in an ideal way to the evaluation of the catalogue itself. The evaluations of formal methods contained in both reports will be joined to a more comprehensive evaluation of formal methods in this specific context.

Acknowledgements

A lot of people of the Sonderforschungsbereich 501 contributed to this report: by giving comments on this and earlier versions of the criteria catalogue as well as by working together with the authors on the case studies upon which the evaluation of SDL, statecharts, and tRTTL is based. Without their help and work, this report could not have been written: L. Baum, B. Dellen, B. Geppert, R. Gotzhein, T. Hillenbrand, E. Kamsties, A. v. Knethen, K. Madlener, C. Peper, M. Schütze, P. Sturm, S. Vorwieger, G. Zimmermann.

References

[AB96] J. Armstrong and L. Barroca. Specification and verification of reactive systems: The railroad crossing example. *Real-Time Systems*, 10:143–178, 1996.

[ABL96] J.-R. Abrial, E. Boerger, and H. Langmaack, editors. *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control*, volume 1165 of *LNCS*. Springer, 1996.

[ACJ+96] M. A. Ardis, J. A. Chaves, L. J. Jagadeesan, P. Mataga, C. Puchol, M. G. Staskauskas, and J. Von Olnhausen. A framework for evaluating specification methods for reactive systems, experience report. *IEEE Transactions on Software Engineering*, 22(6):378–389, 1996.

[AH90] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. of 5th Ann. IEEE Symp. on Logic in Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.

[||] Note, that when being very precise, the FDTs described in these publications have to be considered as two different ones. E.g. the granularity of the time domain is different.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

[Bas96] V. R. Basili. The role of experimentation in software engineering: Past, current, and future. In *Proc. of 18th Intl. Conf. on Software Engineering*, pages 442–449. IEEE Computer Society Press, 1996.

[BDD+94] M. Bäntsch, T. Deiß, J. Denzinger, E. Kamsties, F. Maurer, J. Paulokat, M. Schütze, P. Sturm, and M. Verlage. Kriterien für Spezifikationssprachen. in german, unpublished manuscript, 1994.

[BDK+97] L. Baum, B. Dellen, E. Kamsties, A. von Knethen, and S. Vorwieger. Modelling real-time systems with SCR – lessons learned in a building automation system project. SFB 501 Bericht 07/97, Sonderforschungsbereich 501, Fachbereich Informatik, Universität Kaiserslautern, 1997.

[BG92] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science Of Computer Programming*, 19(2):87–152, 1992.

[BH93] R. Bræk and Ø. Haugen. *Engineering Real-Time Systems, An object-oriented methodology using SDL*. Prentice Hall, 1993.

[Bro96] M. Broy. Formal description techniques - how formal and descriptive are they? In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX, Theory, application and tools*, pages 95–110. Chapman and Hall, 1996.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite–state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGR93] D. Craigen, S. Gerhart, and T. Ralston. An international survey of industrial applications of formal methods. Technical Report NIST GCR93/626, volume 1, US National Institute of Standards and Technology, 1993.

[Che70] H. Chestnut. Information requirements for systems understanding. *IEEE Transactions on Systems Science and Cybernetics*, 6(1):3–12, 1970.

[Che76] P. P.-S. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[CHR91] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

[CP93] P.-J. Courtois and D. L. Parnas. Documentation for savety critical software. In *Proc. of 15th Intl. Conf. on Software Engineering*, pages 315–323. IEEE Computer Society Press, 1993.

[CS96] R. Cleaveland and S. A. Smolka. Strategic directions in computing research – concurrency working group report. *EATCS Bulletin*, 60:97–122, 1996.

[Dei94] T. Deiß. An outsiders evaluation of PAISLey. Internal Report 250/94, Fachbereich Informatik, Universität Kaiserslautern, 1994.

[Dei96] T. Deiß. Combining a state based formalism with temporal logic. SFB 501 Bericht 05/96, Sonderforschungsbereich 501, Fachbereich Informatik, Universität Kaiserslautern, 1996.

[DH97] T. Deiß and T. Hillenbrand. A case study on the use of SDL. SFB 501 Bericht 03/97, Sonderforschungsbereich 501, Fachbereich Informatik, Universität Kaiserslautern, 1997.

[dR91] W.-P. de Roever. Foundations of computer science: Leaving the ivory tower. *EATCS Bulletin*, 44:455–493, 1991.

[Fau95] S. R. Faulk. Software requirements: A tutorial. NRL Memo Report 5546-95-7775, Naval Research Laboratory, Washington, DC, November 1995.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HJL96] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):232–261, 1996.

[HL94] C. Heitmeyer and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proc. of Real Time Systems Symposium*, pages 120–131. IEEE Computer Society Press, 1994.

[HLN+90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions of Software Engineering*, 16(4):403–414, 1990.

[HN96] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[JLHM91] M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Transactions on Software Engineering*, 17(3):241–258, 1991.

[KP92] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, *Proc. of 2nd Intl. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *LNCS*, pages 591–620. Springer, 1992.

[KPG96] M. Kronenburg, C. Peper, and R. Gotzhein. A tailored real time temporal logic for specifying requirements of building automation systems. SFB 501 Bericht 16/96, Sonderforschungsbereich 501, Universität Kaiserslautern, 1996.

[Lam94] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[Leu95] S. Leue. Specifying real-time requirements for SDL specifications - a temporal logic-based approach. In *Proc. of 15th Intl. Symp. on Protocol Specification, Testing, and Verification PSTV'95*. Chapman and Hall, 1995.

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. of 9th Ann. ACM Symp. on Principles of Distributed Computing*, pages 377–408, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, 1992.

[MP93] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[MSC] (CCITT), Z.120 (1993), Message Sequence Charts (MSC), Recommendation Z.100.

[OFMP+94] A. Olsen, O. Færgemand, B. Moller-Pedersen, R. Reed, and J. R. W. Smith. *Systems Engineering Using SDL-92.* North-Holland, 1994.

[Ost87] L. Osterweil. Software processes are software too. In *Proc. of 9th Intl. Conf. on Software Engineering*, pages 2–13. IEEE Computer Science Press, 1987.

[PGK97] C. Peper, R. Gotzhein, and M. Kronenburg. Formal specification of real-time requirements for building automation systems. SFB 501 Bericht 01/97, Sonderforschungsbereich 501, Universität Kaiserslautern, 1997.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of 18th Ann. Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, New York, 1991.

[Ree96] R. Reed. Methodology for real time systems. *Computer Networks and ISDN Systems*, 28:1685–1701, 1996.

[SDLa] (CCITT), Z.100 (1993), Specification and Description Language SDL, Recommendation Z.100.

[SDLb] (CCITT), Z.100 – Annexe F.1 (1993), Definition Formelle du Language de Description et de Specification, Recommendation Z.100 – Annexe F.1.

[SMSV83] R. L. Schwartz, P. M. Melliar-Smith, and F. H. Vogt. Interval logic: A higher–level temporal logic for protocol specification. In H. Rudin and C. H. West, editors, *Protocol Specification, Testing, and Verification III*, pages 3–18, 1983.

[stm91] i-Logix, 22 Third Avenue, Burlington, Mass. 01803, USA. *STATEMATE, The Semantics of Statecharts*, January 1991.

[vdB94] M. van der Beeck. A comparison of statechart variants. In H. Langmaack, W. P. de Roever, and J. Vytopil, editors, *Proc. of 3rd Intl. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 128–148. Springer, 1994.

[Zav91] P. Zave. An insider's evaluation of PAISLey. *IEEE Transactions on Software Engineering*, 17(3):212–225, March 1991.