,,,,,,

Formally-Driven Friendly Specifications of Concurrent Systems: A Two-Rail Approach*

Egidio Astesiano – Gianna Reggio
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova – Italy

Introduction and outline

The advertisement of this workshop rightly mentions that "perceived disadvantages of formal methods are the increased investments and the skills required for their application". We have experienced these problems in some attempts at applying a formal method for the specification of concurrent systems together with industry people. It was realized by both sides that, though the advantages of formal methods are clear, it is necessary to give much importance, together with appropriate tools, to the informal presentation of the formalism.

Thus we have experimented an approach where informal and formal specifications proceed in parallel with a rather rigorous matching, following what we call the "two-rail principle". In other words we propose to make explicit the intermediate stage between a natural language description and its formal counterpart, in order to favour the interaction with the client who states the problem in a natural language and to avoid discouraging the non-specialist in formalities (readers, users and developers), who have to pass abruptly from natural to formal descriptions.

Notice that the above principle concerns with having at hand both informal and formal matching specifications. Thus it also accommodates the derivation of a formal method from an existing informal one, which seems to be the predominant approach in the literature (see e.g. [Hus95]).

Here we present an experience in another direction, which seems to be somewhat new: how to derive from a formal method a rather rigorous informal counterpart. Warning: the derivation is referred to the metalevel, the one of methods; when developing a specification we start with the informal one and then give the matching formal one (with some feedback) in parallel.

We started with a formal algebraic approach for the specifications, at the design level (specifications essentially defining one model), of concurrent systems (the

*This work has been partially funded by the "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of C.N.R. (Italy), EEC-HCM Medicis and by Esprit-BRA W.G. n.6112 Compass.

SMoLCS approach, see e.g. [AR87, RBM94]), later upgraded to a method for covering the various phases of the development from requirement to implementation. The development process is supported by a formal specification language METAL [PR94], that, together with borrowing from algebraic specification languages various structuring features, allows, e.g., for temporal and event logic specifications [CR91, Reg93], and is endowed with various software tools.

In our case we soon realized that, being our models for concurrent systems highly structured, it was possible to provide a simple common guide for structuring both informal and formal specifications, delaying the difference (sentences in a natural language vs. formulae in a logical language) to some last steps of the specification development. Moreover the schema is common both to the requirement and to the design specifications.

In order to exploit the analogy with program practice guided by a well-structured language, we have embodied the model-driven structure of the informal and formal specifications, together with a simple step-by-step development model, into a language that we call "documentation language". That language guides the production at every stage of the informal and formal specifications, alongside with a natural description, which represent the interface with the client, who is actually providing the initial one as a description of the system to build. If we take out just the informal specifications, we are left with a formally-driven method for informal specifications, that could be the principal vehicle for those system engineers who are not trained in formalities, leaving to the specialists the job of writing the formal specifications, as it is already done in some pilot cases.

In the following section we explain how we derive an informal specification schema for the formal models and specifications and in the other one we present some significant fragments of the documentation language. For absolute lack of room no examples are reported here; see, e.g. [AR94] for a complete two-rail specification the problem recently proposed by Lamport and Broy as a common case study at a Dagsthul Seminar, [BL94].

1 Formally-Driven Informal Specifications

The key idea is that the structure of our formal models (of the concurrent systems) defines a common structure for formal and informal specifications, the difference being delayed until some terminal steps, where the correspondence is based on a menu of correspondences (often, though not always, formalizable).

Formal models and specifications

We briefly outline some terminology and the basic concepts of our approach; as much as possible we will adopt simplifications and avoid formalities; full definitions can be found in the references, and many concepts are already well-known.

Our specifications are written in the above mentioned METAL. As usual, a specification SP has a semantics $Mod(SP)$ consisting of a class of models. The models are taken in a universe of structures, classified in *static structures* and *dynamic systems*.

A *static structure* corresponds to the classical concepts of (concrete) data type and models things like numbers, bar codes, queues, bulletin boards and the like. Formally a static structure is a many-sorted first-order structure (or algebra with predicates).

A *dynamic system* models things like an electric plant, a lift system, a computer net and the like.

Formally, as in CCS and similar approaches [Mil89], a dynamic system is seen as a *labelled transition system*, which is a triple (S, L, T), where S is the set of the states, L the set of the labels and T a set of triples (s, l, s'), called transitions, also written $s\ -\ l->\ s'$, where s and s' are states and l a label. The states model the *situations* (along its evolution) of a dynamic system, the labels the nature of its possible *interactions* with the external world and the transitions its possible *activity*, in the sense that each transition $s\ -\ l->\ s'$ corresponds to an action capability.

A dynamic system may be *structured or concurrent*, when there are (at least two) component dynamic systems in parallel, and *simple* otherwise.

The components of a dynamic system may be *active* (dynamic systems) or *passive* (just static structures). Formally the structure of a dynamic system is reflected in the structure of its states, that may be be built out of components (either states of dynamic systems or static data) by a parallel operator.

We distinguish between *requirement* and *design* specifications; requirement specifications define general requirements of a product for a certain level of abstraction, while design specifications define essentially one product. Formally the difference is that $Mod(SP)$ consists just of one model (up to isomorphismn) for design specifications, while it may consist of classes of non-isomorphic models in the other case.

In METAL the design specifications are always amenable to (dynamic) positive conditional specifications, i.e. first-order specifications where the axioms are Horn clauses, guaranteeing the existence of a model (the initial one), which is defined on the basis of logically deducible atoms (equality and elementary predicative assertions).

For requirement specifications it is convenient (and often necessary) to use not only first-order logics but also various kinds of temporal logics, for expressing modalities about system behaviours, like eventually, always, etc. For the purposes of the present paper the kind of language is not much relevant, except that, as we will see, the kind of formulae used in the formal part determines the schemas of the sentences in the informal one.

Correspondence between formal and informal specifications

A product grammar The structure of our models for the ultimate product (a static structure or a dynamic system) suggests a common structure for formal and informal specifications. In order to favor acceptance by system engineers we exploit the analogy with programming practice and describe that structure by means of a simple grammar, where the nonterminals correspond to the concepts introduced so far and the productions to our way of structuring things.

The purpose of this grammar is mainly explanatory; a typical use is in courses teaching the method to system engineers. Here we present it just for understanding the structure of the documentation language, where that grammar will be instantiated for supporting the various development phases. A typical feature of our approach is that the grammar is the same for the requirement and the design phase, which experimentally is proven to be a great simplification.

What we present here is a simplified version, just to give the flavor.

Prod ::= *Stat_Str* | *Dyn_Sys*
Dyn_Sys ::= *Simple_Sys* | *Struct_Sys*
Simple_Sys ::= *States Interactions Activity*
States ::= *Stat_Str*
Interactions ::= *Stat_Str*
Struct_Sys ::= $Comp^+$ *States Interactions Activity*
Comp ::= *Stat_Str* | *Simple_Sys* | *Struct_Sys*

Informal sentence schemas and line-by-line comments By structuring the models and the corresponding common grammar, the difference between informal and formal specifications is confined to the a late stage of detail, when a a natural language sentence has to be matched by a formula in some logical language. The correspondence is built depending on the logical language for the formulae and formalized by a corresponding similar grammar for informal sentences. Below we give the flavor of the correspondence by considering the schemas for the fragment of the specifications at design level dealing with the activity of a simple system.

Formal schema A list of positive conditional formulae of the form:

`if` $atom_1(x_1,\ldots,x_m)$ `and ... and` $atom_n(x_1,\ldots,x_m)$ `then`
$st_1(x_1,\ldots,x_m)$ `--` $inter(x_1,\ldots,x_m)$ `-->` $st_2(x_1,\ldots,x_m)$

where $n \geq 0$, for $i = 1,\ldots,n$ $atom_i(x_1,\ldots,x_m)$ has form $t = t'$ or $Pr(t_1,\ldots,t_q)$, $st_1(x_1,\ldots,x_m)$, $st_2(x_1,\ldots,x_m)$ are terms of sort state and $lab(x_1,\ldots,x_m)$ is a term of sort interaction.

Informal schema A list of sentences having form:

if *atomic-condition*$_1$, ... and *atomic-condition*$_n$ then
the system in the state *state-descr*$_1$ may perform *interaction-descr* passing
in the state *state-descr*$_2$

where $n \geq 0$, for $i = 1,\ldots,n$ *atomic-condition*$_i$ is an informal atomic condition, *state-descr*$_j$ $j = 1, 2$, and *interaction-descr* are informal sentences describing respectively states and interactions, whose possible forms are determined by the informal specifications of the static structures of the states and of interactions.

2 Documentation Language

The specification structure presented in the previous section guides and is incorporated into a language supporting the development process, that we call documentation language (DOC METAL, in this case), since its purpose is to produce a document for each stage of the development. The document includes also various aside comments.

Clearly here the emphasis is not on the adopted development process, which is a simple one, but on the way the specification structure and the development phases contribute to a language supporting companion informal and formal specifications.

We start with a purely requirement phase (PHASE 1), followed by many steps of the second phase (PHASE 2), where each step is an implementation of the previous one, so that the corresponding document has to include a part concerning the verification of the implementation.

By implementation we mean the combination of refinement and realization, following the implementation notion developed by Wirsing and Sannella [Wir90]: *SP is implemented via* α *by* SP', where α is a function transforming specifications iff $Mod(\alpha(SP')) \subseteq Mod(SP)$.

Actually that notion has proven to be quite useful in many concrete case studies.

The following are some significant fragments of DOC METAL.

$$\mathit{Prod\_Doc} \ ::= \ \mathit{Prod\_Name} \quad \mathit{Doc\_Ph1} \quad \mathit{Doc\_Ph2}^{+}$$

All PHASE 1 documents have a common structure, different from that of the PHASE 2 ones.

$$\begin{array}{lll} \mathit{Doc\_Ph1} & ::= & \mathit{Prod\_Name}.\ \mathsf{PHASE\ 1} \\ & & \mathit{Nat\_Descr} \\ & & \mathit{Border\_Determ} \quad \mathit{Inf\_Spec} \quad \mathit{Shadow\_Spots} \\ & & \mathit{Form\_Spec} \end{array}$$

The above structure corresponds to the following structuring of the tasks during PHASE 1.

$$\begin{tikzcd}
 & \mathit{Border_Determ} & \\
\mathit{Nat_Descr} \arrow[r, "\text{Analysis}"] \arrow[ur] \arrow[dr] & \mathit{Inf_Spec} \arrow[r, "\text{Formalization}"] & \mathit{Form_Spec} \\
 & \mathit{Shadow_Spots} &
\end{tikzcd}$$

Nat_Descr is a natural language text describing the product to be realized, it is produced by the client and does not follow any format. It is included in the document and cannot be modified by the developers, since it represents the starting point of the development.

The analysis task consists of examining the natural description trying first to understand if the product is either a static structure or a dynamic system; after the activity goes on differently in the two cases; and also the format of the produced informal specification is different.

Border_Determ gives the motivations for deciding what parts of the product have been included in the specification, i.e. how it has been determined the border of the product. An important feature of our method is that it is not possible to give requirements on the structure and on the behaviour of anything external to the specified product. Thus the choice of where to place the border, i.e. which parts of the outside world to specify, depends on the relevance of the requirements about such parts (e.g. if a user of a system can perform at most 100 times a certain activity, then this condition should be taken into account by specifying also the users, since the developed system is ok only with such user).

Shadow_Spots reports those parts of the product that for some reason cannot be completely defined starting from the natural description or whose only possible interpretation consistent with the natural description does not seem very sound. They may be errors in the natural description or features which must be investigated in the following developments or aspects of the product which are underdefined since they are not relevant.

The schema of the informal specifications follows the structure of the products:

Inf_Spec ::= *Stat_Str_Inf_Spec* | *Simple_Sys_Inf_Spec* | *Struct_Sys_Inf_Spec*
Simple_Sys_Inf_Spec ::= *Prod_Name*. PHASE 1: Informal Specification
[*Basic_Stat_Str_Inf_Spec*]
States_Inf_Spec *Interactions_Inf_Spec* *Activity_Inf_Spec*
Struct_Sys_Inf_Spec ::= *Prod_Name*. PHASE 1: Informal Specification
[*Basic_Stat_Str_Inf_Spec*]
Components *Comp_Inf_Spec*$^+$
States_Inf_Spec *Interactions_Inf_Spec* *Activity_Inf_Spec*
Comp_Inf_Spec ::= *Stat_Str_Inf_Spec* | *Simple_Sys_Inf_Spec* | *Struct_Sys_Inf_Spec*

The basic static structures part contains the informal specifications of those static structures used by the dynamic system, whose use it is not localized in a particular subpart. The role of the other three parts *States_Inf_Spec*, *Interactions_Inf_Spec* and *Activity_Inf_Spec* is clear; the first two are just static structures.

Doc_Ph2 ::= *Prod_Name*. PHASE 2 . step *Num*
Nat_Descr
Inf_Spec *Form_Spec* *Verif*

The above structure of the PHASE 2 documents corresponds to the following structuring in tasks of the activity during a PHASE 2 step.

$$\mathit{Nat\_Descr} \xrightarrow{\text{Analysis}} \mathit{Inf\_Spec} \xrightarrow{\text{Formalization}} \mathit{Form\_Spec} \xrightarrow{\text{Verification}} \mathit{Verif}$$

Here *Nat_Descr* describes using the natural language which choices have been done in this step for the development of the product.

The analysis task in this case consists in trying to express the new product obtained by following the development choices reported in the natural description, but starting from the specification given in the step before; thus, e.g., here we have no to classify the product, since that it is already reported in the previous specification. Notice that now there is no the part about the border determination, since it cannot change along the development, except in case of error.

But now the (informal/formal) specifications, of the whole product or of its parts, may be also design. In general in the first steps of PHASE 2 they are still requirement, while in the last step they are all design; in the intermediate steps some parts may be at requirement level and others at design level, since the development of some parts may be speedy than that of other ones.

The *Verif* parts contains a justification that the formal specification of the product given at this step is a correct implementation of the formal specification of the previous step; it follows the modular structure of the product reflected in the formal specification, and consists of two parts:

- an *implementation function*, i.e. a description of how the parts of the specified product) has been realized in the new one.
- *a proof (justification)* that each property expressed in the old specification, where the old parts have been replaced by the corresponding new ones using the implementation function, is valid in the new one. The check is done using the properties of the new specification.

Conclusion

We have sketched some basic ideas of a method for developing concurrent systems, that combines formal and informal specifications, in the sense that a schema for informal specifications is driven by the formal method. The approach is presented as an experience, since it comes out of a project where some significant test cases have been developed jointly with people from industry. Actually we have produced a rather extensive guide for the developers on the basis of the mentioned approach; the guide extends the formal/informal correspondence to much deeper levels and includes many informal comments for the developer.

We are aware that much more should be done for obtaining a method which hides formalities at any possible level, though being firmly based on them. But, before exploring further this direction, we feel that we should raise the question of the value and flexibility of the "two-rail principle", well outside our own experience. For example it is unclear to us whether in all cases the correspondence between model structure and specification structure is applicable. Moreover, depending on the development model, the structure may change in the various phases.

Also we do not underestimate the importance of graphical notations; their exact role within the two-rail approach has still to be understood. It is also clear that our approach can be combined with the methods supporting the development process, like the one by Souquières and Lévy [SL93], which supports the building of specifications.

Acknowledgments. We want to acknowledge the benefit of many discussions and comments by Valeria Filippi and Ernani Crivelli of ENEL-CRA, Milano.

References

[AR87] E. Astesiano and G. Reggio. An outline of the SMoLCS approach. In *Mathematical Models for the Semantics of Parallelism, Roma, 1986*, number 280 in L.N.C.S. Springer Verlag, 1987.

[AR94] E. Astesiano and G. Reggio. A case study in friendly specifications of concurrent systems (Lamport & Broy's specification problem presented at the Dagsthul seminar "Specification and refinement of reactive systems – A case study"). Technical Report DISI–TR–94–21, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1994.

[BL94] M. Broy and L. Lamport. Specification problem. Distributed to the partecipants of the Dagsthul Seminar on "Specification and Refinement of Reactive Systems: A case Study", 1994.

[CR91] G. Costa and G. Reggio. Abstract dynamic data types: a temporal logic approach. In *Proc. MFCS'91*, number 520 in L.N.C.S. Springer Verlag, 1991.

[Hus95] H. Hussmann. Axiomatic specification of large information systems: Experiences and consequences. In *Recent Trends in Data Type Specification*, number 906 in L.N.C.S. Speringer Verlag, 1995.

[Mil89] R. Milner. *Communication and concurrency*. Prentice Hall, London, 1989.

[PR94] F. Parodi and G. Reggio. Metal: a metalanguage for SMoLCS. Technical Report DISI–TR–94–13, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1994.

[RBM94] G. Reggio, D. Bertello, and A. Morgavi. The reference manual for the SMoLCS methodology. Technical Report DISI–TR–94–12, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1994.

[Reg93] G. Reggio. Event logic for specifying abstract dynamic data types. In *Recent Trends in Data Type Specification*, number 655 in L.N.C.S. Springer Verlag, 1993.

[SL93] J. Souquières and N. Lévy. Description of specification and developments. In *Proc. of International Symposium on Requirements Engineering RE'93*. IEEE Computer Society, Los Alamitos, CA, 1993.

[Wir90] M. Wirsing. Algebraic specifications. In *Handbook of Theoret. Comput. Sci.*, volume B. Elsevier, 1990.