

WORKSHOP NOTES

INTERNATIONAL WORKSHOP ON
MODELS AND LANGUAGES FOR SOFTWARE
SPECIFICATION AND DESIGN


March 30 1984

Orlando, Florida

EDITED BY

Robert G. Babb II
Oregon Graduate Center


Ali Mili
Université Laval

Sponsored by:  IEEE COMPUTER SOCIETY

LCRST OF JAPAN

 UNIVERSITÉ
LAVAL


OREGON GRADUATE CENTER

In cooperation with:  ACM SIGSOFT

A SYSTEM DESCRIPTION METHOD

Bertrand Meyer
Department of Computer Science
University of California
Santa Barbara, California 93106

(on leave from Electricite de France)

1. INTRODUCTION

Specification techniques are used for several purposes: documenting projects, managing their evolution, proving the correctness of programs. We view specifications as being primarily a design tool: having a precise description of the objects to be manipulated by the system is an essential step towards understanding the problem at hand and solving it.

From the literature on specification, it is clear that two quite different schools of thought exist:

- In the methods most widely used by industry, on the one hand, emphasis is placed on the management and documentation aspects; this has resulted in very successful systems (e.g. PSL/PSA, ISDOS, SREM, etc.), which are very strong on aspects such as configuration control, adequate supporting tools, production of up-to-date documents and so forth.

- On the other hand, many of the studies undertaken by universities and research laboratories concentrate on formal notations which can be used to produce complete and unambiguous descriptions of systems. Progress in recent years has proved that such formal methods (e.g. VDM, HDM, FDM, AFFIRM etc.) can be applied successfully to the specification of quite complex systems; it is fair to say, however, that use of such techniques remains rather rare in industry, and still requires more effort than most companies are willing to undertake, except in cases where exceptional security requirements clearly justify it.

In this paper, we present a specification system (where we take the word "system" in software engineering to mean a set of associated methods, notations and tools). This system, called M, has the ambitious goal of trying to combine the best of the above two worlds. The main idea is that people should be permitted to be just as formal as they can afford to be in each particular case. Of course, no miracle should be expected: the less you express in a specification, the less help you will receive from the system. But it is extremely nice to have a system which will allow you to gain some benefits from having written a specification even if

you have not completed it down to the last quantifier.

The design of M resulted from a careful study of the various specification methodologies mentioned earlier. It has also benefited from previous work on the specification language Z.

As mentioned above, M as a system has three facets: a set of methods and principles, a linguistic basis and associated tools. We shall now sketch them in this order.

2. PRINCIPLES AND METHODOLOGICAL BASIS

2.1. Implicitness

In our view, the single most important feature of specifications (as opposed to design documents, code etc.) is that they describe objects implicitly, not explicitly; in other words, a specification should state properties of objects, but not give a way to construct these objects, even if this construction is an abstract one, using mathematical concepts. This may also be expressed by saying that the role of a specification is to say what objects have, not what they are.

As an example of this distinction, consider first the Pascal type definition

```
type POINT =  
  record  
    x, y, z : REAL;  
    speed : VECTOR  
  end
```

Then consider the following characterization of POINT by four functions:

```
x, y, z : POINT --> REAL  
speed : POINT --> VECTOR
```

These two ways of defining POINT may at first sight seem equivalent. The first, however, is explicit, whereas the second is implicit; this also implies that the first is "frozen" (POINT is defined as being "equal" to something); only with the second is it possible to add later a new property of POINTs, say a color:
color : POINT --> C

At a specification stage, when one is exploring issues and trying out different approaches, it is particularly important that nothing should be frozen and that new properties should be easy to add. In fact, the conclusion of the specification step can be taken to be that time when one decides to freeze all the objects involved by equating them.

with the cartesian product of their attributes as defined so far. Then implicit definitions can be transformed (manually or automatically) into explicit ones similar in spirit to the above Pascal type definition.

2.2. Syntax and Semantics

A system can be described as a set of objects upon which certain actions are performed. The description of the relational structure of the system, i.e. what objects are connected to what other objects and what operations apply to what objects, can be called the syntax of the system. Its semantics, on the other hand, is the description of the properties of the objects and the operations.

Describing semantics is a much more difficult task than describing syntax if one is to remain at the specification level. Many industrial specification systems are mostly good at describing the syntax, and their attempts at including the semantics use either natural language or an algorithmic language. On the other hand, formal specification techniques make it possible to describe system semantics while remaining at the specification level, but they require a lot of effort.

The method used in M is to describe a system by successive steps; the first stages are concerned with syntax, the later ones add semantics. It will be possible to benefit from the specification even if one only stops at one of the early stages.

2.3. Object-orientedness

The description of a system may be structured around the objects or around the operations. The former alternative seems to yield better design descriptions in terms of flexibility and evolutivity. The objects are modelled by "sorts", similar to types of programming languages; mathematically, however, they are adequately described as sets.

2.4. Functions

The basic modeling tools used in the description of systems are the simple mathematical notions of sets and functions. In particular, no notion of "type", as may be derived from category theory, was deemed necessary. Functions may be either total or partial; partial functions play an important role in connection with error situations.

3. LANGUAGE

M specifications are expressed as a set of paragraphs. These successive views of the same concept, each of which gives more detail, are a generalization of the way Ada

packages are described in two successive parts ("specification" and "body").

The successive paragraphs in the specification of a system X are as follows:

system X sorts

-- Lists all the sorts which intervene in X.

end

system X interface

-- Lists the elements which are borrowed
-- from other specifications
-- (this makes it possible to have "libraries"
-- of basic specifications, and generic specifications)

end

system X attributes

-- Lists the functions which give properties of
-- elements of the various sorts of X

end

system X invariants

-- Lists the properties of attributes
-- which must be preserved by transformations

end

system X transformations

-- Lists the functions corresponding to operations
-- modifying elements of the various sorts of X

end

system X effects

-- Defines the effect of the transformations
-- on the attributes

end

system X constraints

-- Lists the domains of all partial functions
-- (attributes or transformations)

end

Other possible paragraphs include one for error processing (error processing consists in extending the domains of non-total functions) and, possibly, implementation.

4. TOOLS

To be practical, a specification system requires adequate tools. The tools which are necessary for efficient use of M include:

- A screen-oriented syntax editor, making it easy to

enter and change specifications;

- Validation tools for checking the consistency and completeness of specifications;

- Configuration management tools, for monitoring the development and modification of specifications;

- Automated aids for the development of the programs once the specification has been completed. The most promising area here seems to be the automatic generation of data structures from descriptions of attributes and transformations.