A Comparison of the Specification Methods

M. Frappier[1] and H. Habrias[2]

[1] Université de Sherbrooke, Département de mathématiques et d'informatique, Sherbrooke, Québec, Canada, J1K 2R1, Marc.Frappier@dmi.usherb.ca
[2] Institut de Recherches en Informatique de Nantes, 3 rue Maréchal Joffre, 44041 Nantes cedex 1, France, henri.habrias@irin.univ-nantes.fr

In this chapter, our goal is to provide a *qualitative* comparison of the specification methods introduced in this book. Our intention *is not to rank* methods in terms of quality or productivity. It would require a much larger sample of specifications, developed within a controlled experiment, to reach valid conclusions about quality or productivity.

We have selected a set of attributes which describe several properties of specification methods. The definitions of these attributes are provided in the first section. In the second section, attributes are evaluated for each method; the results are described in a set of tables. The reader may then compare methods by comparing the values of their attributes.

1 Attributes of Specification Methods

Table 1 enumerates the attributes and their possible values. Attributes are then defined and illustrated with simple examples.

No.	Attribute	Values
1	paradigm	state machine, algebra, process algebra, trace
2	formality	informal, semi-formal, formal
3	graphical representation	yes, no
4	object-oriented	yes, no
5	concurrency	yes, no
6	executability	yes, no
7	usage of variables	yes, no
8	non-determinism	yes, no
9	logic	yes, no
10	provability	yes, no
11	model checking	yes, no
12	event inhibition	yes, no

Table 1. List of specification method attributes

1. **paradigm** : This attribute characterizes how the specification *notation* describes the system behavior. We identified four general paradigms: trace, state machine, algebra and process algebra.

To illustrate this attribute, we have chosen a very simple system, a latch (i.e., a Boolean device), for which example specifications are provided. A latch has three operations: $reset$, which sets the latch to zero; $flip$, which changes the internal state of the latch, and $valueOf$, which returns the internal state of the latch.

state machine : the specification describes a transition relation on a set of *states*. Transitions are labeled by events. The transition relation may take several forms. The set of states can be enumerated (as in an automaton) or it can be described by a set of variables with their possible values.

Example 1 *A Mealy finite state machine [6] is based on set enumeration. In the example below, the symbol λ can be interpreted in a number of ways, including that the output is a* don't-care *value, a mute message, or the absence of output.*

$Inputs = \{reset, flip, valueOf\}$
$Outputs = \{0, 1, \lambda\}$
$States = \{zero, one\}$
$InitialState = zero$
$TransitionFunction = \{\ (zero, reset) \mapsto zero,$
$(zero, flip) \mapsto one,$
$(zero, valueOf) \mapsto zero,$
$(one, flip) \mapsto zero,$
$(one, reset) \mapsto zero,$
$(one, valueOf) \mapsto one\ \}$
$OutputFunction = \{\ (zero, reset) \mapsto \lambda,$
$(zero, flip) \mapsto \lambda,$
$(zero, valueOf) \mapsto 0,$
$(one, flip) \mapsto \lambda,$
$(one, reset) \mapsto \lambda,$
$(one, valueOf) \mapsto 1\ \}$

Example 2 *State machine based on state variables.*

List of operations : reset, flip, valueOf
List of variables : s of type Boolean
Initialisation of variables : s := 0
Definition of operations
reset $\triangleq$ *s := 0*
flip $\triangleq$ *if s = 0 then s := 1 else s := 0*
valueOf $\triangleq$ *return s*

algebra : the specification describes a set of *operations* defined on a set of types (also called *sorts* or *carrier sets*) [14]. An event is represented by a function (also called an *operation*). The behavior of functions is given by a set of equations (axioms) stating how functions are related.

Example 3 *Algebraic specification of a latch.*

Sorts : Latch, Nat
Signatures of functions :
zero : a constant of sort Latch

reset : a function from Latch to Latch
flip : a function from Latch to Latch
valueOf : a function from Latch to Nat
0 : a constant of sort Nat
1 : a constant of sort Nat
Axioms
for all x of sort Latch
$reset(x) = zero$
$flip(flip(x)) = x$
$valueOf(zero) = 0$
$valueOf(flip(zero)) = 1$

process algebra : it is a special kind of algebra. Its operations are applied to elementary processes and events to describe how events may occur.

Example 4 *Process algebraic specification of a latch using the CSP notation [8].*

List of events (alphabet) : reset, flip, valueOf, 0, 1
Definitions of processes

$$\begin{aligned} LATCH &\triangleq ZERO \\ ZERO &\triangleq (reset \rightarrow ZERO \mid \\ &\quad\; flip \rightarrow ONE \mid \\ &\quad\; valueOf \rightarrow 0 \rightarrow ZERO) \\ ONE &\triangleq (reset \rightarrow ZERO \mid \\ &\quad\; flip \rightarrow ZERO \mid \\ &\quad\; valueOf \rightarrow 1 \rightarrow ONE) \end{aligned}$$

trace : the specification describes a *sequence of events* that the system may engage in. Events are sometimes classified into *input* events (i.e., events for the system: they are submitted by the environment to the system) and *output* events (i.e., events for the environment : they are produced by the system for the environment). A trace specification does not describe the internal system state. It only describes the interaction between the system and the environment.

Example 5 *Some traces of the latch.*

List of events : reset, flip, valueOf, 0, 1
trace 1 = $\langle valueOf, 0, flip, valueOf, 1, flip, valueOf, 0\rangle$
trace 2 = $\langle flip, reset, flip, flip, valueOf, 0\rangle$

The set of traces for the latch is infinite. A trace-based specification notation provide mechanisms to completely define the set of traces.

Note that the attribute paradigm should not be confused with the *semantics* of a notation. The semantics of a notation is concerned with the *meaning* of specification texts. For instance, the meaning of an operation in the B notation can be described using a set of axioms relating the precondition and the postcondition of an operation. This style of semantics is called a *weakest precondition semantics* [7]. Another style is to associate a mathematical object to each specification text. For instance, the semantics of a B operation can also be given by a pair (D, R), where D is the set of initial states for

which the operation terminates and R is the set of pairs (s, s') such that the operation started on s may terminate on s'.

2. **formality** : This attribute characterizes the syntax and the semantics of the notation. It has three possible values.

   informal : the syntax is not formally specified. Typically, the specification is given in natural language.

   semi-formal : the syntax of the notation is formally specified, but it does not have a formal semantics.

   Example 6 *A data flow diagram used in SAZ [12] includes processes, arrows, data stores and external entities. There are precise rules describing how these elements can be combined. For instance, an arrow must have a source and a destination. The source and the destination must be a process, a data store or an external entity. The notation does not have a formal semantics. For instance, there are no rules to determine if two diagrams are equivalent (aside from being syntactically identical), or if an implementation satisfies a data flow diagram.*

   formal : the syntax of the notation is formally specified, and it has a formal semantics.

   Example 7 *An operation in the B notation [1] has a weakest precondition semantics. It is possible to determine (prove) that two operations are equivalent, or that an implementation satisfies a specification.*

3. **graphical representation** : This attributes determines if the notation includes a graphical representation (e.g., pictures, diagrams, graphs). It has two possible values: yes or no.
4. **object oriented** : This attributes determines if the notation uses the following concepts : inheritance, class, polymorphism, and encapsulation [9]. It has two possible values: yes or no.
5. **concurrency** : A notation allows the modeling of concurrency if a system can be described in terms of processes which may communicate. It has two possible values: yes or no.

   Example 8 *CSP allows the modeling of concurrency. The system $P \parallel Q$ is made of processes P and Q.*

6. **executability** : This attribute determines if the specification *can* be executed to simulate the system behavior. Note that specifications can be non-deterministic. In such a case, the interpreter either computes the set of possible outcomes and asks the users to pick one, or it picks an outcome in a random manner. Note also that the interpreter does not need to be efficient. This attribute has two possible values: yes or no.
7. **usage of variables** : This attribute determines if variables can be used to denote values in the system. It has two possible values: yes or no. The use of variables fosters abstraction, because it avoids the enumeration of all possible values.

   Example 9 *Examples 3 and 2 use variables. CSP[8] allows variables, but we have not used any in Example 4. A Mealy state machine (Example 5 does not allow variables.*

8. **non-determinism** : This attribute determines if the notation allows non-determinism. It has two possible values.
   yes : a specification may offer a choice between several outcomes for a computation.
   Example 10 *Consider the specification of a machine that must change a dollar for a set of coins. The specification may leave the choice of the algorithm for the selection of the set of coins to be made at the design stage. The specification may look like this: let C be a set of coins returned by the machine.*
   $$\sum_{c \in C} valueOf(c) = 1$$
   no : a specification is always deterministic.
9. **logic** : This attributes determines if the notation includes some first-order (or higher-order) logic notation. Logic is useful to express properties in an abstract manner. This attribute has two possible values: yes and no. Note that a notation may not include logic, but its semantics may be defined in some logic.

   Example 11 *CCS [10] does not include a logic notation. However, its semantics is defined in terms of a set of inference rules expressed in a logic notation. There is also a logic for reasoning about the equivalence of CCS specifications, or proving properties about CCS specifications.*

   Example 12 *B includes a logic notation; hence the value of attribute logic for B is yes.*

10. **provability** : This attributes determines if *properties* about the specification can be proved using a formal proof system. It has two possible values: yes and no.

    Example 13 *In the latch specification of example [14], one may prove that*
    $$valueOf(flip(reset(x))) = 1$$

11. **model checking** : This attributes determines if *properties* about the specification can be checked by enumeration of the system states. It has two possible values: yes and no.
12. **event inhibition** : An event is the execution of an operation. A system is said to allow the inhibition of events if the set of operations it offers to the environment may be different from one point in time to another. If several operations are offered, the environment picks one of them (external non-determinism) and the operation is executed (i.e., the event occurs). If two events are independent (i.e., they do not share variables), they can be executed in parallel.

    Example 14 *The B notation as defined in [1] does not allow event inhibition. All operations of a B machine are always offered to (or can be invoked by) the environment. If an operation is invoked, it may either terminate or fail.*

Example 15 *A Mealy machine (see example [6]) does not allow event inhibition. Its transition function and its output function must be total; hence, it always offers all its operations to the environment.*

Example 16 *Process algebras, action systems [3], and the extension of the B notation as defined in [2, 5] allow event inhibition.*

2 A Qualitative Description of the Methods

Tables 2, 3, and 4 provide a description of each method with respect to the attributes defined in the previous section. Note that these tables refers to the methods as they are described in the book; they do not consider their various extensions.

References

1. Abrial, J.-R.: *The B-Book*, Cambridge University Press, 1996.
2. Abrial, J.R., Mussat, L.: Introducing Dynamic Constraints in B, in *B'99: Recent Advances in the Development and Use of the B Method*, D. Bert, Ed., LNCS 1393, Springer-Verlag, 1998, 83–128.
3. Back, R. J. R., Kurki-Suonio, R: Decentralization of process nets with centralized control. In 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, Montréal, Québec, Canada, 1983, 131–142.
4. Booch., G.: *Object-Oriented Analysis and Design with Applications*, 2nd edition, Benjamin-Cummings, 1994.
5. Butler, M., Waldén, M.: Distributed System Development in B, in *First Conference on the B Method*, H. Habrias, Ed., Institut de Recherche en Informatique de Nantes, Nantes, France, 1996, 155-168.
6. Cassandras C.G., Lafortune S.: *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999,97–100
7. Dijkstra, E.W.: *A Discipline of Programming*, Prentice Hall, 1976.
8. Hoare, C.A.R.: *Communicating Sequential Process*, Prentice Hall, 1985.
9. Meyer, B.: *Object-Oriented Sotware Construction*, Second Edition, Prentice Hall, 1997.
10. Milner, R.: *Communication and Concurrency*, Prentice Hall, 1989.
11. Morgan, C.: *Programming from Specifications*, Prentice Hall, 1990.
12. Polack F.: SAZ: SSADM version 4 and Z in *this book*, M. Frappier H. Habrias (Ed.), Springer Verlag, 1999 244–250
13. Waldén M.:Layering Distributed Algorithmes within the B Method in *B'98: Recent Advances in the Develpment and Use of the B Method,LNCS 1393*, D. Bert, Ed., LNCS 1393, Springer-Verlag, 1998, 244–250
14. Wirsing: Algebraic Specification in *Handbook of Theretical Computer Science, Vol. B* , J. van Leewen (Ed.), Elsevier, 1990,675–788
15. Yourdon, E.: *Modern structured analysis*, Yourdon Press, 1989.

method name	**paradigm**	**formality**	**graphical representation**	**object-oriented**
Action Systems	state transition	formal	no	no
B	state transition	formal	no	no
CASL	algebra	formal	no	yes
Cleanroom & JSD	traces & process algebra	formal	yes	no
COQ	state transition	formal	no	no
Estelle	state transition	formal	no	no
LOTOS	process algebra	formal	no	yes
OMT & B	state transition	formal	yes	yes
Petri Nets	state transition	formal	yes	no
Petri Nets with Objects	state transition	formal	yes	yes
SART	state transition	informal & semi-formal	yes	no
SAZ	state transition	semi-formal & formal	yes	no
SCCS	process algebra	formal	no	no
SDL	state transition	formal	yes	yes
UML	state transition	informal & semi-formal	yes	yes
VHDL	state transition	formal	no	no
Z	state transition	formal	no	no

Table 2. List of specification method attributes - part 1

method name	**concurrency**	**executability**	**usage of variables**	**non-determinism**
Action Systems	no	yes	yes	yes
B	no	yes	yes	yes
CASL	no	yes	yes	no
Cleanroom & JSD	no	yes	yes	yes
COQ	no	yes	yes	yes
Estelle	yes	yes	yes	no
LOTOS	yes	yes	yes	yes
OMT & B	no	yes	yes	yes
Petri Nets	yes	yes	no	yes
Petri Nets with Objects	yes	yes	yes	yes
SART	yes	no	no	yes
SAZ	no	yes	yes	yes
SCCS	yes	yes	yes	yes
SDL	yes	yes	no	yes
UML	yes	no	no	no
VHDL	yes	yes	yes	no
Z	no	yes	yes	yes

Table 3. List of specification method attributes - part 2

method name	**logic**	**provability**	**model checking**	**event inhibition**
Action Systems	yes	yes	yes	yes
B	yes	yes	yes	no
CASL	yes	yes	yes	no
Cleanroom & JSD	yes	yes	yes	no
COQ	yes	yes	yes	no
Estelle	no	no	yes	yes
LOTOS	yes	yes	yes	yes
OMT & B	yes	yes	yes	no
Petri Nets	no	yes	yes	no
Petri Nets with Objects	no	yes	yes	no
SART	no	no	no	no
SAZ	yes	yes	yes	no
SCCS	yes	yes	yes	yes
SDL	yes	yes	yes	yes
UML	no	no	no	no
VHDL	no	no	yes	no
Z	yes	yes	yes	no

Table 4. List of specification method attributes - part 3