Towards Formalizing Structured Analysis

Luciano Baresi and Mauro Pezzè
Politecnico di Milano

Real-time extensions to Structured Analysis (SA/RT) are popular in industrial practice. Despite the large industrial experience and the attempts to formalize the various "dialects", SA/RT notations are still imprecise and ambiguous.

This paper tries to identify the semantic problems of the requirements definition notation defined by Hatley and Pirbhai, one of the popular SA/RT "dialects", and discusses possible solutions. As opposed to other papers that give their own interpretation, this paper does not propose a specific semantics for the notation.

This paper identifies imprecisions, i.e., missing or partial information about features of the notation; it discusses ambiguities, i.e., elements of the definition that allow at least two different ("reasonable") interpretations of features of the notation; and it lists extensions, i.e., features not belonging to the notation, but required by many industrial users and often supported by CASE tools.

This paper contributes by clarifying whether specific interpretations can be given unique semantics or retain ambiguities of the original definition. The paper allows for the evaluation of formal definitions by indicating alternatives and consequences of the specific choices.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; D.2.2 [**Software Engineering**]: Tools and Techniques—*Structured programming*

General Terms: Design, Documentation

Additional Key Words and Phrases: Structured Analysis/Real-Time, Informal vs. Formal Specifications, Hatley and Pirbhai's Requirements Definition Notation.

1. INTRODUCTION

The choice between informal and formal methods for system and software development is still open. Formal methods [Clarke and Wing 1996], though broadly disseminated and used experimentally by industry [Gerhart et al. 1994; Ardis et al. 1995; Craigen et al. 1993; Hinchey and Bowen 1995; Crow and Vito 1996], lag be-

This work has been partially supported by the ESPRIT Project EP8593 IDERS: Integrated Development Environment for embedded Real-time Systems with complete process and product visibility.
Address: Dipartimento di Elettronica e Informazione - Politecnico di Milano, Piazza Leonardo da Vinci, 32, I-20133 Milano, Italy, e-mail: [baresi—pezze]@elet.polimi.it

hind informal methods on realistic industrial applications. The success of informal methods is due to several reasons: the accumulated experience, the current training and education level of the experts of the application domains, and the good tool support. Some good tools are becoming available on industrial scale for formal methods too [Owre et al. 1995; Jordan et al. 1991]. However, training personnel for new techniques and accumulating experience with new methods require considerable time and effort, strongly limiting the diffusion of formal methods, and favoring informal approaches at least in the near future.

Informal methods still dominate even in areas where formal methods should provide a competitive advantage, like in safety critical applications, where errors can have disastrous consequences. Real-time extensions to structured analysis (SA/RT [Ward and Mellor 1986; Hatley and Pirbhai 1987; Bruyn et al. 1988]) are based on informally defined specification notations, that present imprecisions and ambiguities. Nevertheless, they are among the most popular methodologies for developing safety critical applications ([Wood and Wood 1989]). SA/RT specification notations look simple and clear. This is mainly due to their appealing graphical representation and to the methodologies that separate the use of different components of the notations in distinct phases of the development. However, the subtle interplay of different components and the many details of each component complicate the notation, resulting in imprecisions and ambiguities that are difficult to reveal and solve. SA/RT specifications are given various interpretations in different organizations and sometimes even within the same organization or development team. The variety of interpretations given to the same SA/RT specifications strongly limits the power of SA/RT methodologies and of the supporting tools ([Interactive Development Environments 1994; Structured Technology Group, Inc. 1995; Mark V Systems 1994]), that provide some form of dynamic semantic analysis, if any at all.

Recently, several studies tried to solve the ambiguities of SA and SA/RT methodologies by ascribing formal semantics to such notations [France and Larrondo-Petrie 1994]. In [Semmens and Allen 1991; Larsen et al. 1991; Wing and Zaremski 1991; France 1992] SA is integrated with formal models. In [Semmens and Allen 1991] SA is formalized by means of Z. In [Larsen et al. 1991] SA/SD is integrated with VDM; in [Wing and Zaremski 1991] SA is integrated with Larch. Translation rules from SA to algebraic state transition systems are supplied in [France 1992]. Formal semantics for Ward's transformation schemas is proposed in [M. D. Fraser and Vaishnavi 1991; Miranda 1989; Fencott et al. 1994; Elmstrøm et al. 1993; Tuya et al. 1995; Petersohn et al. 1994]. Ward's transformation schemas are translated into VDM in [M. D. Fraser and Vaishnavi 1991] and into process algebras in [Fencott et al. 1994]. In [Miranda 1989; Elmstrøm et al. 1993; Tuya et al. 1995] transformation schemas are translated into different kinds of Petri nets. In [Petersohn et al. 1994] transformation schemas are ascribed different semantics by means of the formal methods developed for *synchronous* languages. Petri nets corresponding to Ward's transformation schemas are also built by some commercial tools, e.g., DESIGN/IDEF ([Meta Software Corporation 1992]) or pre-commercial prototypes, e.g., IPTES ([Bologna 1993]). Petri nets are used in [Richter and Maffeo 1993; Shi and Nixon 1996] to ascribe semantics to ESML ([Bruyn et al. 1988]), and in [France and Wu 1995] to ascribe semantics to the Hatley and Pirbhai's requirements

definition notation [Hatley and Pirbhai 1987].

These works differ from one another in many ways: the considered "dialect" of SA notations, the level of detail, the formal model, the algorithmic aspects, and the interpretation of ambiguities. However, they all aim at ascribing specific semantics to a specification notation. Imprecisions and ambiguities are solved, but they are seldom discussed and compared with other possible choices.

The goal of this paper is to discuss imprecisions, ambiguities, and natural extensions to SA/RT. Specifically, the paper focuses on the requirements definition notation described in [Hatley and Pirbhai 1987], hereafter referred to as the *HP notation*. The HP notation has been chosen because it is widely used [Wood and Wood 1989]; it is supported by several commercially available tools ([Interactive Development Environments 1994; Structured Technology Group, Inc. 1995; Mark V Systems 1994]), and it is a good representative of the class of SA/RT notations. We focus mostly on dynamic semantics, since static semantics is already well described in the previously referenced literature, and most dangerous errors are due to misunderstandings on dynamic semantics, as pointed out in [Petersohn et al. 1994]. This paper does not ascribe yet another semantics to the HP notation. Rather, it provides a means for evaluating different interpretations that may be chosen according to specific semantics. The aim of the work described in this paper is to understand the correctness of specific interpretations, to estimate the impact of changing a given interpretation, and to support the definition of tools for dynamic semantics analysis. This paper represents the first step in understanding and formalizing the HP notation by identifying those parts of the language that are problematic.

The investigation documented in this paper has been based on several sources of information: books and papers, commercially available CASE tools, direct interviews with development teams and reviews of documentation of industrial products developed using the HP notation. Specifically, we studied both the "original" book by Hatley and Pirbhai [Hatley and Pirbhai 1987], hereafter the *HP book*, and the paper by France [France and Wu 1995]. Other papers on different extensions to structured analysis have been compared with the HP book and the paper by France to identify other interpretations applicable to the HP notation as well. The HP book has been considered the main source of information. We used both textual descriptions and examples to disambiguate definitions. Imprecisions and ambiguities of the HP book are mainly due to its nature: an informal definition of a methodology for experts of the application domain, and not a formal description of the semantics of the suggested notation. The paper by France has been used mainly as a source of suggestions, being the only available formal definition of the HP notation, to the best of our knowledge. We examined bounds and extensions implemented in Software through Pictures [Interactive Development Environments 1994], hereafter StP, AxiomSys [Structured Technology Group, Inc. 1995], and ObjectMaker [Mark V Systems 1994], three popular commercial tools (also) supporting the HP notation. We interviewed the partners of the IDERS consortium, and we reviewed documentation of industrial projects used to evaluate the IDERS toolset[1]. Finally,

[1]IDERS is a project sponsored by the European Community (ESPRIT EP8593). The IDERS toolset is an extension of StP with execution and animation capabilities. It has been validated on

we took advantage from the experience accumulated in the IPTES[2] project while defining a formal semantics for Ward's transformation schemas [Elmstrøm et al. 1993].

The main contributions of this paper are the identification and solution of imprecisions of the HP notation, the classification and evaluation of different meaningful solutions for ambiguities, and the discussion of natural extensions.

The proposed solutions have been evaluated using the IDERS toolset, that allows the formal definition and experimental validation of the dynamic semantics, by means of graph grammars [Baresi et al. 1995].

The organization of the paper reflects the taxonomy of the problems. Section 2 introduces the terminology. Section 3 sketches the HP notation, identifying and clarifying imprecisions. Section 4 classifies ambiguities and discusses different solutions. Section 5 discusses common extensions to the HP notation. Section 6 provides some criteria to identify classes of design notations for which the proposed interpretations are applicable, referring to our experience on safety critical applications. Section 7 indicates natural follow-ups of the work documented in this paper. Table 3 provides the readers with an orthogonal organization based on the components of the notations, rather than on the classes of problems.

2. TERMINOLOGY

The study of documents written in natural language yields recurring patterns of deficiencies, that can be classified in several ways. In this paper we distinguish three main classes of deficiencies:

Imprecisions:. the elements of the definition do not provide enough information to interpret a feature of the notation.

Ambiguities:. the presence in the definition of elements that make it possible to interpret a feature of the notation in at least two different ("reasonable") ways.

Extensions:. features not belonging to the notation, but required by many industrial users and often supported by CASE tools.

In this paper, the class of imprecisions comprises also incomplete definitions, or silences as defined in [Meyer 1985]; i.e., the existence of features in the notation not covered by any element of the definition. Incomplete definitions represent the special case of absence of information on how to interpret a feature of the notation. The class of ambiguities comprises also inconsistencies, or contradictions as defined in [Meyer 1985]; i.e., the presence in the definition of elements that define a feature of the notation in at least two incompatible ways. Inconsistencies represent the special case of incompatibility between the different definitions of the same construct.

Other classes of deficiencies discussed in [Meyer 1985], i.e., noises, overspecifications, forward references, wishful thinkings are not addressed here. Noises,

---

industrial applications by experts of Alenia (an Italian company that produces critical systems, such as defense and air traffic control systems), and it is currently used by Ansaldo Trasporti (An Italian company that produces transportation systems) within the UseGat project (ESPRIT EP20347). The experience of Alenia's engineers with the reply processor and channel management of a mode-S radar system is reported in [Bove et al. 1996].

[2]IPTES is a project sponsored by the European Community (ESPRIT no. EP5570 and EP7811)

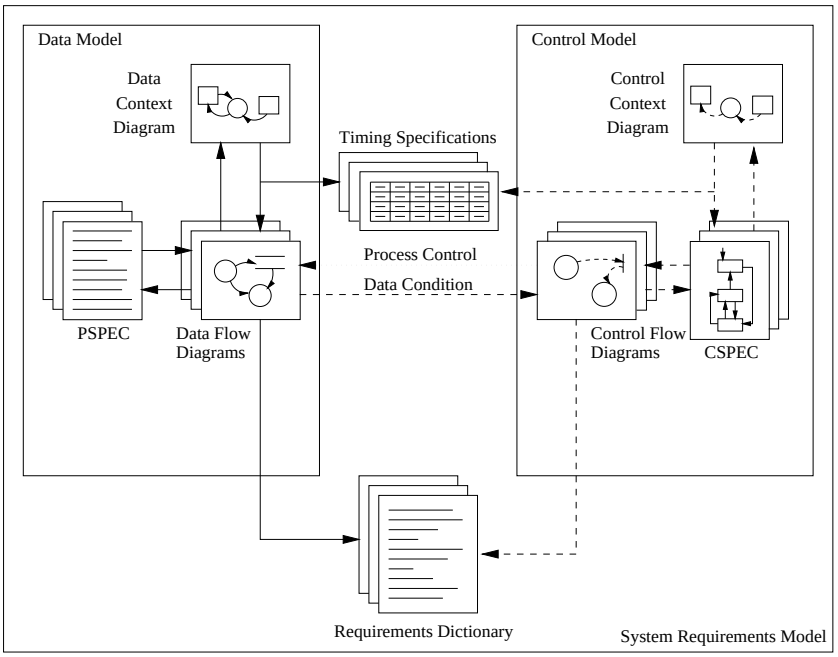


Fig. 1. The System Requirements Model

overspecifications, and forward references can be annoying and bad in requirements documents and shall be avoided; but not in books, where some redundancy can be used intentionally to facilitate learning. Since this paper is mostly based on the *HP book*, we will not address such deficiencies. Wishful thinkings refer to a context of implementations of specifications, but are less important in our context: the definition of a specification notation.

3. IMPRECISIONS OF THE HP NOTATION

This section provides a sketchy introduction to the HP notation for the reader not familiar with [Hatley and Pirbhai 1987], and highlights its imprecisions.

Figure 1 shows the structure of a *System Requirements Model*, i.e., a full specification. Arcs indicate the relations between the main components. Solid lines represent data flow, dashed lines represent control flow, and dotted lines represent process control. A complete specification comprises a *Data Model*, a *Control Model*, a *Timing Specifications* module, and a *Requirements Dictionary*.

3.1 Data Model

The Data Model defines the functional behavior of the system. It specifies the data flows between the system and the environment, with which it interacts, and the data transformations performed by the system.

A data model comprises a *Data Context Diagram*, several *Data Flow Diagrams* and *Process Specifications*.

3.1.1 *Data Context Diagram (DCD).* The DCD shows the data flows between the system, identified as a single function, hereafter *process*, and the environment, with which it interacts, described as external entities, hereafter called *terminators*. The DCD represents the highest and most abstract view of the system, as far as data transformation is concerned.

3.1.2 *Data Flow Diagrams (DFDs).* DFDs define a hierarchy of views. The hierarchy is rooted in the DCD, and refines the data model. Leaf processes represent primitive functionalities. Intermediate processes introduce views at different abstraction levels. Each DFD comprises *processes*, *data stores*, and *data flows*. Processes define instantaneous data transformations. Data stores are information repositories. They can be used to store data, that otherwise would be lost, or to provide constant values (read-only stores). Values in data stores can be accessed indivisibly. Selective accesses are allowed only if statically defined. Readings do not alter the contents of stores. Data are lost only when overwritten.

Connections between processes or between processes and data stores are established by means of data flows. Data flows can be either *continuous* or *transient* in the time domain. Continuous flows retain their value until overwritten or switched off. Transient flows carry a value only instantaneously. Processes can send information to the control model (described below) by means of particular control flows called *data conditions*.

Imprecisions

The connections of flows to processes are not completely specified in the HP book. The number and type of flows entering a process are open, and the enabling conditions are described vaguely. Such questions can be solved by looking at the examples presented in the HP book. The solutions proposed hereafter are accepted by any reasonable interpretation investigated by the authors.

Number of In/Out Flows. Leaf processes can have zero or more inputs, but must have at least one output, being a data flow, a data condition, or a data store. This rule is explicitly imposed in StP, and can be motivated as follows. A process without outputs cannot communicate the results of the modeled transformation; thus it would be useless. On the contrary, a process with no inputs can be used, for instance, to generate a random number, or to initialize a store. An example is given by process `Clear Payment` of Figure 2, that is an excerpt from the "Vending Machine" example presented in [Hatley and Pirbhai 1987] (Chapter 28, pages 342-354). In [France and Wu 1995], the problem is ignored; AxiomSys and ObjectMaker do not perform any control.

Number of Time-Transient Input Data Flows. Time-transient data flows "trigger" the firing of their target processes, since they carry values only instantaneously. To keep the enabling condition simple and clear, some interpretations of the HP notation restrict the number of time-transient inputs to be at most one. However, this must be considered as a particular interpretation of the notation, since two or more time-transient data flows entering the same process are used in several cases, see, as examples, the "Foreign Exchange Autoteller" of Figure 5 (page 14), and the

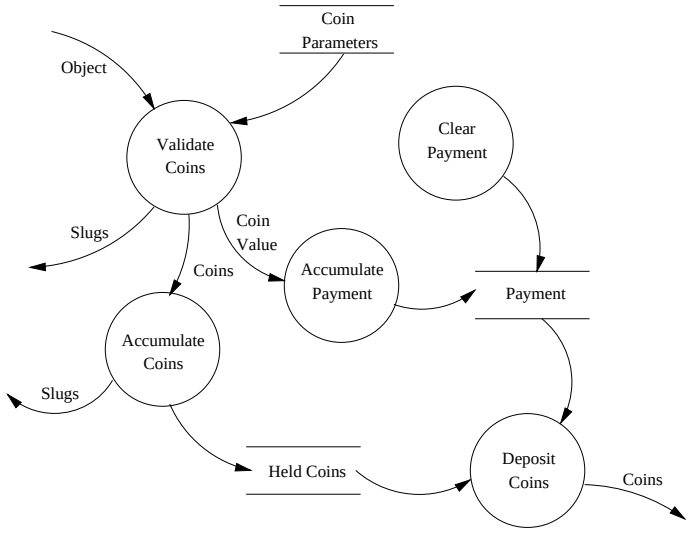


Fig. 2. An excerpt of the "Vending Machine" example

"Evaluator" of Figure 6 (page 14). Thus, any combination of both time-transient and time-continuous data flows entering the same process shall be allowed (related problems are addressed in Section 4.2).

Enabling Condition. The conditions to start executing a process are vaguely described in the HP book. They are addressed by two different sentences:

> *Whenever data sufficient for a given process to perform its task appears at its inputs, then it will perform that task, and do so instantaneously.*
>
> [Hatley and Pirbhai 1987], Section 4.8, page 56.

> *The moment an input stimulus occurs, the corresponding output is instantly present.*
>
> [Hatley and Pirbhai 1987], Chapter 7, page 98.

The word *stimulus* is not defined further. The obvious interpretation as synonymous to *flow* is too restrictive here; it shall be interpreted as a set of input flows carrying values, as shown in examples of the HP book. For instance, in the example reported in Figure 2, process `Validate Coins` starts executing when both `Object` and `Coin Parameters` are available, as indicated by the PSPEC reported in Figure 3 (page 12). Further problems concerning enabling conditions, that can be given different interpretations are discussed in Section 4.

◁ ○ ○▷

3.1.3 *Process Specifications (PSPECs).* PSPECs define the functional transformations performed by processes. Only leaf processes of the DFDs hierarchy are associated with PSPECs. The behavior of intermediate processes is described by means of their descendants. The HP book does not constrain PSPECs. PSPECs

can be anything; they can be expressed either graphically or textually, either formally or informally. The only rule given in the HP book is keyword "issue" to indicate that an output flow is time-transient ([Hatley and Pirbhai 1987], Section 13.3, page 162). The lack of any rule for PSPECs is the source of many ambiguities that are discussed in Section 4.1.

3.2 Control Model

The Control Model describes the control aspects of the system. It interacts with the data model through control flows and by enabling/disabling processes. The control model comprises a *Control Context Diagram*, *Control Flow Diagrams*, and *Control Specifications*.

3.2.1 *Control Context Diagram (CCD).* The CCD describes control interactions with the environment. It consists of control flows connecting the terminators and the process, modeling the whole system, introduced in the DCD.

3.2.2 *Control Flow Diagrams (CFDs).* CFDs augment the DFD hierarchy with control information, modeled as control flows, control stores, and CSPEC bars. Although Hatley and Pirbhai separate completely CFD and DFD hierarchies, practitioners and related tools smooth the distinction. StP and ObjectMaker merge the two hierarchies. AxiomSys leaves the choice of merging the two hierarchies up to users.

Control stores are information repositories like data stores. Data stores are used by the data model, while data in control stores are used by the control model.

CSPEC bars indicate connections between processes and CSPECs. CSPECs, that define the control patterns, are described in the next paragraph.

Control flows are time-transient. They can be seen as sparks to CSPECs that react by activating/deactivating processes and/or producing new values on other control flows. Control flows cannot activate/deactivate processes explicitly. A process is activated/deactivated by means of *process controls*. Process controls are not drawn explicitly, but remain hidden within CSPEC bars. Control flows can exit any process, but they can enter only CSPEC bars, terminators, and intermediate processes. Control flows entering a process shall eventually enter a CSPEC bar belonging to the sub-hierarchy rooted in the process.

CSPECs control the data model hierarchically: a CSPEC activates a process and all its descendants. Thus, a process is active only when all of its ancestors are active too.

3.2.3 *Control Specifications (CSPECs).* CSPECs define how CSPEC bars react to control signals and data conditions in terms of both other control signals and process controls. CSPECs are Mealy or Moore automata. In Mealy automata, transitions can be associated with events and/or actions. Events are predicates that must be satisfied to let the transitions happen. Actions produce new signals and/or process controls. In Moore automata, actions are functions of the current state only. In some cases, all or part of the information provided by the automaton is scattered over a set of tables: state transition tables, process activation tables, event logic tables, and action logic tables.

Imprecisions

Process Activation/Deactivation. The definition of process activation/deactivation could lead to doubts. Processes are not switched on and off as one would expect intuitively. This is explicitly defined by a reference to Ward and Mellor's work [Ward and Mellor 1986]:

> *They are not activated by "switching them on" and deactivated by "switching the off" – the convention adopted by Ward and Mellor, which implies that the process itself has memory.*
>
> [Hatley and Pirbhai 1987], Section 6.2, page 86.

The state of a process depends only on the last transition executed in the corresponding CSPEC:

> *An important convention used in this model is that the actions, although they are associated with the transitions, which are transient in nature, are assumed to continue in effect until the next transition occurs.*
>
> [Hatley and Pirbhai 1987], Section 6.2, page 86.

This assumption could lead to misunderstandings when considering process activation tables:

> *Instead of using just the usual binary output values of 0 and 1, some rows in the table contain 0, 1, 2, 3, and so on. The 0 has the usual meaning: False or Off. The sequence 1, 2, 3 indicates that when the input criteria for that row become True, the processes are activated in that sequence.*
>
> [Hatley and Pirbhai 1987], Section 6.3, page 90.

The activation of the processes listed in the table is computed dynamically from the column corresponding to the last occurred event. The table contains one column for each event that triggers the CSPEC, i.e., the set of control flows that enter the CSPEC bar. Each cell of the table contains a natural number. Positive numbers indicate activation sequences, while 0 simply means that the corresponding process is inactive. When a new event occurs, activation is newly computed, regardless of the status prior to the execution. Processes that do not correspond to rows in the table are always active ([Hatley and Pirbhai 1987], Section 2.2, page 16).

◁ ○ ○▷

3.3 Timing Specifications

Users can specify only "external" timing requirements, i.e., timing properties at the interfaces with the environment with which the system interacts. The HP methodology postpones the definition of "internal" timings to the design phase, that follows requirements analysis:

> *Since users are only concerned with external timing, this is our only concern in the requirements model.*

[Hatley and Pirbhai 1987], Section 7.3, page 102.

According to the methodology, users can define only:

—Repetition rates: the intervals between two subsequent occurrences of the same external signal.

—Input to output responses: the maximum interval that can elapse between the arrival of an external event and the answer of the system.

However, PSPECs can cope with both absolute and relative time in at least two cases:

> *One is the case in which an external primitive signal is being generated, and its rate or time relationship with another signal is an important requirement; the other occurs when repetition rate of a process is an essential part of that process.*
>
> [Hatley and Pirbhai 1987], Section 15.1, page 190.

Neither France [France and Wu 1995] nor the tools we examined (StP, AxiomSys, and Objectmaker) take into account timing specifications. Problems related to internal timing specifications are discussed in Sections 4.2 and 5.3.

Imprecisions

Process Duration. The two exceptions listed so far could suggest that the general assumption of instantaneous execution is not attended. This is not true, if we distinguish between enabling a process to proceed and executing it. The possibility to refer to internal timing implies that processes can be non instantaneously enabled when their inputs are available. But, when enabled, processes execute instantaneously. There is no way to associate a duration with process executions. Consider, for instance, a process that can execute once per second. The enabling condition is defined by using a clock: the process can execute instantaneously after every second.

We can only argue that in this way there is an unnatural mixture of data processing and control within PSPECs, but the semantics is clear.

◁ ○ ○▷

3.4 Requirements Dictionary

The requirements dictionary contains the definitions of all the data types used by all the components. Every store and every flow must be defined. Types can be either primitive or composed with the usual constructs: composition, iteration, selection. The concrete syntax varies according to the different sources of information that we considered. In their book, Hatley and Pirbhai adopt a simple evolution of the notation proposed by De Marco [Marco 1978]. StP provides users with a graphical editor, where constructs are rendered by proper shapes. AxiomSys constrains textual type definitions with keywords `begin description - end description`. France explicitly suggests some textual languages [Goguen and Winkler 1988; Guttag et al. 1985]; however, in his work, type definitions are not relevant.

The requirements dictionary is standard and does not introduce ambiguities. It is mentioned here for completeness only.

4. AMBIGUITIES OF THE HP NOTATION

The HP notation presents several ambiguities that affect various features of the notation: PSPECs, "internal timing conditions", multiple input flows, time-transient flows, process activation/deactivation, CSPECs, and merging points.

4.1 PSPECs

In the HP book, PSPECs are specified informally. StP [Interactive Development Environments 1994] considers PSPECs as comments; no checks are provided. AxiomSys [Structured Technology Group, Inc. 1995] requires the direction of flows used in PSPECs to be specified. The direction of the flows is checked with respect to their use.

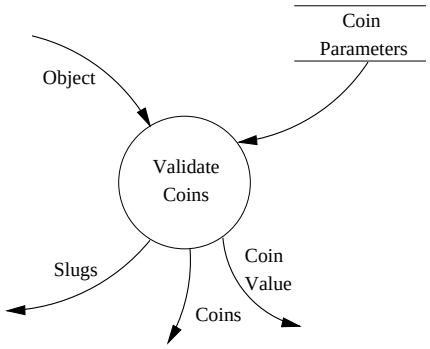
The lack of constraints for PSPEC definitions favors imprecisions and ambiguities. Even in the simple examples presented in the HP book, PSPECs are ambiguous and not always consistent with the rules. For example, the informal PSPEC of process `Validate Coins` of the "Vending Machine" example shown in Figure 3 ([Hatley and Pirbhai 1987], Chapter 28, pages 342-354) does not specify in which cases the execution of the process produces a value on output `Coin Value`, nor does it specify the value that has to be produced. The possible interpretation that process `Validate Coins` does not produce any value on output `Coin Value`, contradicts the rule that requires each output to correspond to at least one combination of input values:

> *Every input to the process is used within some expression in the specification, which shows how every output from the process is generated.*
>
> [Hatley and Pirbhai 1987], Section 4.7, page 53.

Practitioners often constrain PSPECs to be defined in a precise language, usually some kind of pseudo-code. Simply adopting a precise language does not guarantee the satisfaction of all the requirements. Open problems concern the scope of PSPECs and their relation with input/output flows. We show that PSPECs do not describe only a purely functional behavior, as expected, but often include control and timing information. Relations with input/output flows can be ambiguous, since programming languages do not usually define when and how input/output values shall be matched, or what to do with unmatched values.

4.1.1 *Functionality, Control and Timing.* In the HP book, PSPECs are described as purely functional data transformations, that instantaneously produce data on output flows when sufficient input data are provided (see quote on page 7). Activation conditions and timing requirements are ruled out, since control must be defined in the control model and internal timing requirements cannot be included in the requirements analysis (see quote on page 9).

However, in the examples reported in the HP book and in the examined industrial applications, PSPECs define also control and timing conditions. For instance, the PSPEC of process `Clock Trip Time` of the "Automobile Management System" example ([Hatley and Pirbhai 1987], Chapter 26, pages 295-325), reported in Figure 4

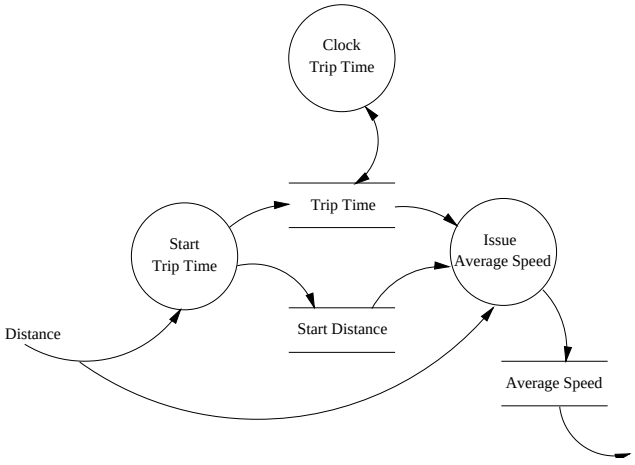



```
Validate Coins:

Examine OBJECT to see if it matches any set of COIN PARAMETERS
    if so:
        accept OBJECT as COIN
    otherwise
        return OBJECT as SLUG
```

Fig. 3. Process `Validate Coins` of the "Vending Machine" example

defines also the activation condition (*"is on"*), and the internal timing requirements (*every second*).




```
Clock Trip Time:

Every second process is on, add 1 second to TRIP TIME.
```

Fig. 4. The "Automobile Management System" example (DFD 4.1)

Thus, examples and case-studies suggest that PSPECs can describe not only functional data transformations, but also activation conditions, and internal timing

requirements. Processes are executed when data are available on their input data flows, activation conditions are satisfied, and internal timing requirements are met. A language for defining PSPECs shall thus include constructs to precisely define not only functional transformations, but also conditions on activation (e.g., *process is on*) and internal timing specifications (e.g., *every second*).

4.1.2 *Input Data Flows.* In the HP book, input consumption during process execution seems to be defined precisely (see quotes on pages 11 and 7). However, several examples raise unexpected problems.

Input Coverage. Let us consider the "Foreign Exchange Autoteller" example of Figure 5. The PSPEC of process `Autoteller` meets the requirement: every input is used within some sub-expression. If process `Autoteller` is fed with either US Dollars or German Marks, its behavior is clearly defined. However, if both US Dollars and German Marks are provided simultaneously, the PSPEC does not indicate clearly the expected behavior. Different solutions could be acceptable:

—*Run-time error:* the execution could lead to a run time error, being the input combination not considered explicitly. This is an odd solution for the class of systems addressed by the methodology: in safety critical applications, run-time errors shall carefully be avoided.

—*Random interleaving:* one of the two inputs could be considered first. However, thanks to instantaneous process executions, the second input would be taken into account later. This solution leads to hidden nondeterminism. Although nondeterminism can be useful at the requirements level, hidden nondeterminism is often avoided in safety critical systems.

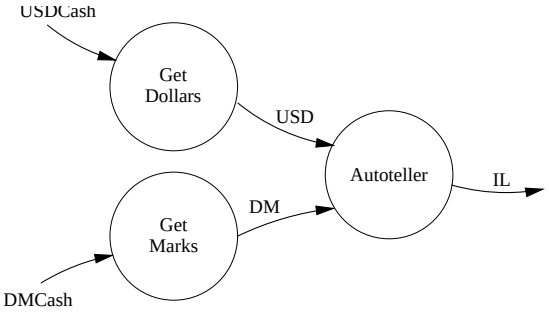
—*Static description of all cases:* all possible cases must statically be defined.

—*Deterministic choice:* the choice of which input to process first could be given by the order of statements. However in this way, decisions are taken implicitly.

The specific case can be solved by splitting process `Autoteller` into sub-processes dealing with US Dollars and German Marks, respectively. In general, the example suggests that the use of pseudo-code, e.g., C- or Pascal-like functions, does not solve the problem. A precise specification language for PSPECs shall clearly indicate how to process all combinations of inputs, either explicitly or with some precisely defined default mechanism.

Nondeterminism. In the former paragraph, we discussed how the simultaneous presence of the two inputs to process `Autoteller` can lead to nondeterministic execution sequences. The example of Figure 6 shows that PSPECs satisfying the requirements can unexpectedly produce not only nondeterministic execution sequences, but also nondeterministic behaviors, i.e., different and unpredictable results. In the PSPEC of process `Evaluate` of Figure 6, the two considered combination of inputs partially overlap. Hence, the presence of all the three inputs could lead to the production of either `X + Y` or `X - Z`. Since inputs are time-transient, one of the two executions disables the alternatives.

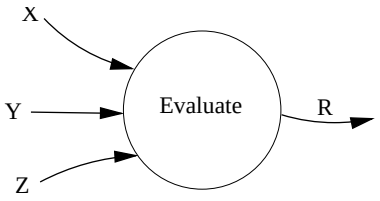
To avoid situations as the one exemplified in Figure 6, the specification language for PSPECs must either prevent partial overlapping of inputs in the different cases or clearly specify the choice.




```
Get Dollars:                    Autoteller:

issue USD = USDCash             if (USD) then
                                   IL = US * 1600
Get Marks:                      end if
                                if (DM) then
issue DM = DMCash                  IL = DM * 1100
                                end if
```

Fig. 5. The "Foreign Exchange Autoteller" example




```
Evaluate:

case
    XY : R = X + Y
    XZ : R = X - Z
end case
```

Fig. 6. A trivial arithmetic evaluator

4.1.3 *Output Data Flows.* Although not explicitly defined, it is clear from examples and practical uses that PSPECs can produce data only on selected outputs. For example, the PSPEC of process `Validate Coins` of Figure 3 (page 12) does never produce data on both flows `Slugs` and `Coins`. The effect on output flows not involved in the execution is not defined.

Since time-continuous flows keep their value, we can assume that after executing a process, output flows not involved in the execution retain their old value. However, we can also assume that process executions clear output flows not explicitly overwritten. Both interpretations are reasonable and are considered in different application domains, but the choice cannot remain unspecified, since it can result in different behaviors.

Notice that in the Ward and Mellor's notation, output flows cannot be overwritten, thus different interpretations may arise ([Petersohn et al. 1994]). Such interpretations do not apply to the HP notation, which explicitly states that flows can be overwritten ([Hatley and Pirbhai 1987], Section 13.3 page 162).

4.2 Internal Timing Conditions

The need to include internal timing requirements in PSPECs has been discussed in paragraph 4.1.1. Although explicitly denied (see quote on page 9), it is exceptionally admitted also in the HP book:

> *... the convention that all PSPECs and CSPECs have access to absolute and relative time. This convention should be used with great discretion.*
>
> [Hatley and Pirbhai 1987], Section 15.1, page 190.

In the examples of the HP book and in practical industrial uses, internal timing requirements are usually given in terms of repetition rates (see Figure 4, page 12). Repetition rates are defined by either indicating fixed values (*every n seconds*, see Figure 4), or *random* values ([Hatley and Pirbhai 1987], Section 15.1, page 191), or execution dependent values: *on demand* ([Hatley and Pirbhai 1987], Section 15.1, page 191), *each time activated* ([Hatley and Pirbhai 1987], Section 26.3, page 306).

Neither the HP book nor the examined industrial applications clearly define whether repetition rate refers to an absolute or a relative time scale. An absolute time scale can be hardly identifiable since time is ruled out from (most of) requirements specifications and subsystems are specified asynchronously. Relative time scales raise the problem of determining the time origin.

The time origin can refer to either the "first" or the "last" enabling instant of the process, i.e., the first (the last) instant at which a "valid combination of values" appeared on the input flows. In both cases, the time origin is "local" to processes: different processes can refer to different time origins. In the case of unique time origin (absolute time scale), lack of synchronization among neighbors processes could lead to unexpected loss of data. In the case of variable time origin (relative time scale), problems are given by "fast changing" flows. Time-transient input flows that change faster than the time rate of the process make such a rate meaningless.

4.3 Multiple Input Flows

The HP book allows any combination of time-continuous and time-transient flows, entering or exiting a process (see Section 3, page 6). Data stores behave like time-

continuous flows, and will not be further referenced in this subsection. Outputs are produced by the instantaneous execution of enabled processes. Time-transient outputs keep their value only instantaneously, while time-continuous outputs retain their value until turned off or overwritten ([Hatley and Pirbhai 1987], Section 13.3, page 162). Process enabling is determined by the availability of values on the input flows, as discussed in Subsection 4.1.2 (page 13). The definition of enabling conditions is straightforward for processes with exactly one time-transient input flow and any number of, including zero, time-continuous input flows. On the contrary, different interpretations are possible when processes have several or no time-transient input flows combined with any number of time-continuous flows.

4.3.1 *Multiple Time-Transient Input Flows.* Although some application domains require at most one time-transient input flow for each process, multiple time-transient flows entering the same process are admitted in the HP book. In this case, it must be defined whether they have to be considered synchronously or asynchronously. In the first case (synchronous), all the time-transient inputs of a process must carry a value simultaneously to "enable" the process, since time-transient flows carry data only instantaneously ([Hatley and Pirbhai 1987], Section 13.3, page 162). In the second case (asynchronous), process executions can be triggered by data on a subset of its time-transient input flows. The presence of time-continuous flows does not affect the above interpretations, since they continuously hold values.

4.3.2 *No Time-Transient Input Flows.* The HP book does not impose the use of time-transient input flows to trigger process executions. Thus, both processes with only time-continuous inputs, and processes with no inputs are accepted.

Since values are persistent on time-continuous flows, processes with only time-continuous inputs can be executed either continuously or only when at least an input value changes. Although the problem can look like a typical problem of performance while executing a model, it actually impacts on repetition rates and time-transient flows of neighbor processes. Continuously executing processes continuously produce new values on output flows, while process executions triggered by changes on input flows produce only discrete value sequences.

Processes with no inputs can either be executed continuously or require to be controlled, i.e., activated/deactivated by CSPECs. In the second case, these processes would be executed only when activated.

4.4 Time-Transient Flows

Time-transient flows are defined by keyword "issue" in PSPECs ([Hatley and Pirbhai 1987], Section 13.3, page 162-163). No further control is required. A flow is transient if it assumes a value only temporarily.

Due to the assumption that data transformations (processes) are instantaneous, when a process is continuously executing, output flows are continuously refreshed. The refresh frequency would be infinite and then the sampling interval null. In this case, time-transient flows would act as time-continuous ones. Such a problem does not occur if the process is assigned with internal timing conditions (e.g., execution rate), or it is suitably controlled by a CSPEC.

The identification of time-transient flows by keyword "issue" in PSPECs has an unexpected effect. Since keyword "issue" can be used only within PSPECs, time-

transient flows can exit only processes, but not terminators. Thus, transience in time depends on the scope of the specification. Narrowing the size of the considered system may turn time-transient flows into time-continuous ones. For example, turning process `Autoteller` of Figure 5 (page 14) into a stand alone system would force the input flows `USD` and `DM` to become time-continuous.

4.5 Process Activation/Deactivation

Processes are controlled by the control model, that indicates when they are active or not. Active processes are defined as follows:

> *The convention chosen is that when a process is deactivated, it does nothing, and its outputs are null (just as they are when it has insufficient inputs to do its job). Deactivating a process is just like temporary deleting it, and all its outputs, from the system.*
>
> [Hatley and Pirbhai 1987], Section 5.7, page 73.

> *when a process is deactivated, all its descendants are also deactivated ... it seems reasonable that when a process is turned off, everything inside it is turned off also.*
>
> [Hatley and Pirbhai 1987], Section 5.7, page 74.

The HP book does not define the effect of deactivation on the input flows of a deactivated process and on data stores and CSPEC bars, that belong to its descendants.

4.5.1 *Input Flows.* On the surface, time-transient input flows do not cause problems, since their values are available only instantaneously. However, a deactivated process can have different acceptable effects on its time-continuous input flows:

—Time-continuous input flows are treated as if they were output flows: they are "virtually deleted" when the target process is deactivated. In this case they loose their values. Such interpretation complicates the execution, requiring a "partial" deactivation of processes that produce values on them.

—Time-continuous input flows are ignored by the deactivated process, retaining their values during the deactivation of their target process.

—Both time-continuous input and output flows retain their value. Although this solution violates the explicit requirements of the HP book, it is adopted in some application areas.

4.5.2 *Data Stores and CSPEC Bars.* According to the HP book, descendants of a deactivated process are deactivated too (see quote on page 17). Such requirements lead to different interpretations for data stores and CSPEC bars. Stores could either loose their contents or preserve it until next re-activation. If the contents of stores are lost, their re-activation can result either in an undefined or in a default value. A similar problem applies to CSPECs. When CSPECs are deactivated, they can either clear their state or keep it for the next activation. The two interpretations correspond to the default and history modes of Statecharts [Harel et al. 1990].

4.6 CSPECs

CSPECs are deterministic finite state automata (Mealy or Moore machines), i.e., transitions exiting the same state are associated with different events. Thus the occurrence of an event univocally defines the new active state. However, the simultaneous occurrence of events, that match transitions exiting from the same state of a CSPEC, can cause nondeterministic behavior. This case can be given different interpretations:

—One of the events is chosen randomly and the others are discarded.

—One event is chosen first and the others are served afterwards. This solution is equivalent to the former one when the chosen event moves to a state that does not accept any of the remaining events. It produces different effects, if some of the pending events can be served in the new state.

—A run-time error is raised.

4.7 Merging Points

The HP book does not constrain merging and splitting points. Splitting points do not raise interpretation problems in all examined cases. Although apparently "secure", the absence of constraints on merging flows can introduce inconsistencies. Problems arise when defining the result of merging "conflicting" flows: time-continuous with time-transient flows, flows with "overlapping" type definitions, and time-transient flows with different sampling rates.

4.7.1 *Merging Time-Continuous and Time-Transient Flows.* The HP book does not constrain merging flows to be of the same type, nor does it define the effects of heterogeneous merging. Many practical applications of the HP methodology simply restrict the notation by only allowing merging of flows of the same type. However, most supporting tools, including all the ones we examined, do not forbid merging of heterogeneous flows, but they do not define the dynamic semantics.

The merging of flows of different types can be regarded as an error. Alternatively it can produce either a time-transient or a time-continuous result. In the case of time transient result, the resulting value is available just when the transient input is available; it is undefined otherwise. In the case of time continuous result, the output value is always available. Figure 7, taken from the "Automobile Management System" example in the HP book (Section 26.3, page 310) illustrates the case. Flow `Display` is the merging of two time-continuous flows (`Fuel Consumption` and `Average Speed`) and a time-transient flow (`Maintenance Needed`). In this case, all interpretations are acceptable.

4.7.2 *Overlapping Type Definitions.* The HP book does not constrain the type definition associated with merging flows. Merging flows with overlapping type definitions are allowed in all examined tools and (seldom) used in practical applications.

Intersecting type definitions of merging flows can be given different solutions:

—It is considered to be an error. This solution is similar to the one adopted by object-oriented languages, which do not allow the same property to be inherited by two distinct parents.

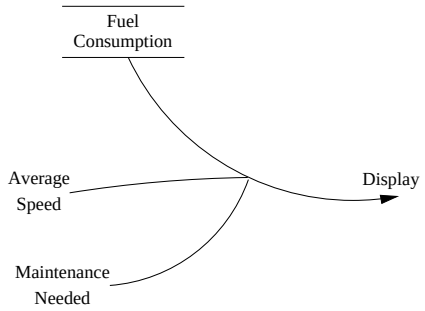


Fig. 7. Merging of heterogeneous flows

—One of the overlapping type definitions is chosen randomly. This solution introduces hidden nondeterminism, which conflicts with the application domain (safety critical software), where nondeterminism is avoided and hidden nondeterminism is often considered an error.

—Conflicts on type definitions of merging flows can explicitly be solved by annotating the merging point. This solution represents a good trade-off between the previous ones, but augments the notation, thereby requiring explicit conflict resolutions whenever raised.

4.7.3 *Multiple Time-Transient Flows.* The HP book allows several time-transient flows to merge, but does not define when the resulting value is produced. The problem is exemplified in Figure 8. Flows `X`, `Y`, and `Z` enter the same merging point that produces a value on the output flow `Position`. Since the input flows are produced independently, they can carry values at different instants. Values on the output flow can be given two different semantics:

—They can be produced only when all the three values are present at the same time. In this case in Figure 8, output `Position` would be defined only at time t_0 and t_6 (case a). This interpretation can cause "silent" outputs if the sampling rates of the inputs never intersect.

—They can be produced when at least one input is available, missing values being undefined. In this case, the sample rate of the output is given by the union of the sample rates of the inputs. In the example of Figure 8, output `Position` would be defined at every instant (case b). This solution can produce meaningless outputs for particular subsets of inputs.

The problem is not solved by replacing the merging point with a process. Nor is it solved by assuming that some values wait for the arrival of the missing ones. The first solution allows us to program the way outputs have to be produced in an explicit way, but three time-transient flows entering a process cause the problem described in Section 4.3.1.

The second solution, proposed in [Petersohn et al. 1994], applies to Ward and Mellor models; not to the HP notation. It would violate the semantics of time-transient flows: values are only present temporarily and no processing takes place at merging points ([Hatley and Pirbhai 1987], Section 4.5, page 50).

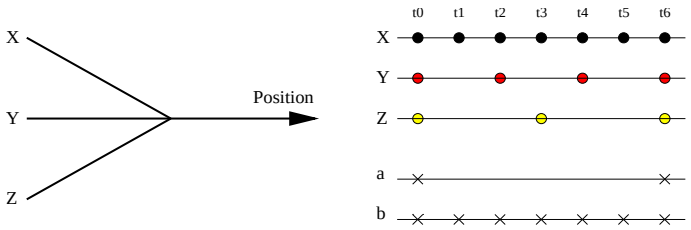


Fig. 8. Merging of multiple time-transient flows.

5. EXTENSIONS TO THE HP NOTATION

The HP notation is often extended by practitioners and tools. Many extensions reflect particular choices and vary from tool to tool, and from application environment to application environment. However, some extensions are common to several instantiations of the HP methodology and are or can easily be supported by the considered tools. In this section, we overview the most common extensions to the HP notation, that are so widespread that they can be considered almost part of the HP notation from their frequent use. We also indicate some extensions that, although less used and supported, have been welcome in most examined industrial sites.

5.1 Structured Data Stores

Data stores record values that can be accessed either as a whole or by means of statically defined selectors. End-users cannot associate lists, queues, or any other dynamic data structure to data stores. Dynamic objects can be simulated by accessing the information as a whole and by programming the desired access policy within the objects that actually access it. In many cases, data stores are augmented with dynamic data structures. Commonly used dynamic data structures are *list* and *set*.

5.2 Structured Flows

In the HP notation, flows carry a single value at a time. Values on time-continuous flows are lost when overwritten. Values on time-transient flows disappear immediately. Such constraint can cause synchronization problems between processes connected by using data flows: production of values on data flows must be synchronized with their consumption. Asynchronous communication can be modeled by means of data stores. Alternatively, practitioners and tools define specific data flows. On these flows, values are not overwritten, but queued.

5.3 Timing

As discussed in the previous sections, internal timing requirements are controversial: although explicitly forbidden, they are exceptionally allowed (see quote on page 15). However, even when permitted, internal timing requirements are limited to delays and activation rates, but execution is always instantaneous. The rationale provided by the HP book is that internal timing requirements are not part of the requirements, but of the detailed design specifications.

However, in several industrial case studies we did, some internal timings are part

of the requirements. For example, the mode-S radar system described in [Bove et al. 1996] imposes maximum delays in processing data to be able to scan all the aircrafts during a revolution period of the antenna. Ignoring internal timings during the early requirements specification can compromise the whole design of the system.

Many practitioners extend the HP notation by allowing specification of duration of process execution. The most common models allow either fixed duration or minimum and maximum duration to be associated with processes. This extension has been used to define the requirements of the radar system in [Bove et al. 1996]. The ability to associate a duration with processes lets designers cope with timing requirements, not only at the boundaries of the system, but also within the system. Thus designers could start reasoning on the delays of each component at this level, instead of postponing the reasoning to the design phase.

This extension changes the semantics of process execution. Specifically, it modifies process activation/deactivation. The deactivation of executing processes can be interpreted in different ways:

—The involved process is aborted, breaking the current execution. The effect of a deactivation is immediate. Outputs are never produced. Inputs consumed but not completely used are either lost or restored.

—The process is deactivated only after the termination of its execution. Thus deactivation is delayed for the time required to reach a "stable" state.

—The current execution is suspended and the process is deactivated. The execution is resumed when the process is re-activated. A process is suspended and not killed, thus consumed inputs are deleted, while outputs are only postponed.

6. INSTANTIATING THE HP NOTATION

The analysis presented in the previous sections indicates that the HP notation is a family of notations. Particular instantiations are given by choosing one of the solutions proposed in Section 4 for the identified ambiguities, and adding some of the extensions presented in Section 5.

Some choices are extremely specific, and can vary among applications of the same domain. Some other choices determine the degree of nondeterminism of the instantiation, and thus the application domain they can be used for. Deterministic instantiation are well suited for safety critical applications, where predictability is a critical property. Nondeterministic solutions are better suited for soft real-time applications, where flexibility of the notation is more important than predictability. Thus, we can identify two sub-families: *hard HP notations*, which are completely deterministic, and *soft HP notations*, which are intrinsically nondeterministic. Guidelines to instantiate the HP notation are supplied in Tables 1 and 2, which list the ambiguities identified in Section 4 and summarize the proposed solutions. The last column of the tables indicates for which family of notations the solution is suitable for:

Section	Problem	Solutions	Family
PSPECs (Section 4.1)	PSPECs describe:	Functional data transformations only.	H
		Functional data transformations, activation conditions, and internal timing requirements.	h
	If there are too many input flows carrying values:	A run-time error is raised.	h
		Random interleavings are allowed.	s
		Static description of all cases is required.	H
		A deterministic choice is imposed.	h
	Nondeterminism in using input values can be avoided by:	Preventing partial overlapping.	H
		Clearly specifying the choice.	h
	Output flows not involved in an execution:	Retain their old values.	-
		Are cleared.	-
Internal Timing Conditions (Section 4.2)	Repetition rates can refer to:	Absolute time scale.	H
		Relative time scale with unique time origin.	h
		Relative time scale with variable time origin.	s
Multiple Input Flows (Section 4.3)	Processes, with more than one time-transient input flow, can be enabled when:	All time-transient input flows carry a value simultaneously.	H
		There is a value on, at least, one time-transient input flow.	h
	Processes with only time-continuous input flows can:	Be always executing.	h
		Execute only when there are changes on input values.	H
	Processes with no inputs:	Can be always executing.	h
		Must be explicitly activated/deactivated.	H

Table 1. Summary of ambiguities of the HP notation and possible solutions. Part A: PSPECs, Timing, and Flows.

Section	Problem	Solutions	Family
Time-transient Flows (Section 4.4)	If a process is continuously executing, its exiting time-transient flows:	Have infinite refresh frequency.	-
		Act as time-continuous flows.	-
	The identification of time-transient flows by keyword "issue":	Forbids time-transient flows exiting from terminators.	-
		Lets transience in time depend on the scope of the specification.	-
Process Activation/Deactivation (Section 4.5)	The effect of deactivating a process on its incident flows can be that:	Both input and output flows are "virtually" deleted, loosing their values.	-
		Input flows remain alive, while output flows are deactivated.	-
		Both input and output flows remain active.	-
	If a process is deactivated, its child stores and CSPECs can:	Preserve their contents.	-
		Loose their state and be re-activated with a default state.	H
		Loose their state and be re-activated with an undefined state.	h
CSPECs (Section 4.6)	When simultaneous events awake more than a single transition:	One event is chosen randomly and the others are discarded.	s
		One event is chosen first, and the others are served afterwards.	s
		A run-time error is raised.	h
Merging Points (Section 4.7)	The result of merging time-transient and time-continuous flows can:	Be regarded as an error.	h
		Result in a time-transient flow.	H
		Result in a time-continuous flow.	H
	Overlapping type definitions of merging flows can be:	Treated as errors.	H
		Solved randomly.	s
		Solved by annotating the merging point.	h
	The flow resulting from merging two or more time-transient flows can carry a value when:	All the incoming flows carry a value simultaneously.	H
		At least one input value is available.	h

Table 2. Summary of ambiguities of the HP notation. Part B: Flows, Process activation, CPSECs, and Merging Points.

-	The solution does not affect the determinism of the notation. Any alternative is acceptable for both hard and soft HP notations.
H	The solution enforces strong predictability of the notation. It has to be used for hard HP notations.
h	The solution enforces determinism, but in some cases, it can be difficult to check statically. It can be used for hard HP notations, but is less strong than a solution labeled with **H**.
s	The solution causes nondeterminism. Unsuitable for hard HP notations, it belongs to soft HP notations.

The extensions described in Section 5 can be added freely to both hard and soft HP notations.

7. CONCLUSIONS

The advantages of using structured analysis (SA) are limited by the informality of the notations, that can lead to ambiguous specifications.

This work discusses imprecisions, ambiguities, and possible extensions of the requirements definition notation proposed by Hatley and Pirbhai. The paper identifies the limits of the notation and describes different plausible solutions of the identified problems. It does not choose a specific solution, but rather discusses the different proposals.

The first result of this paper is the description of imprecisions and ambiguities with the HP notation. Most of the imprecisions and ambiguities are not easily identifiable by simply reading documentation and some of them may remain unidentified even by experts. Sometimes it may be difficult to distinguish precise requirements of the notation from arbitrary choices. The second result of the paper is an overview of different interpretations for the ambiguities. The paper attempts to provide a complete picture by discussing all the solutions that are either provided in books and papers, implicitly assumed by supporting tools, or that can be derived from sample industrial applications. Table 3 summarizes the imprecisions, ambiguities, and extensions we have discussed.

This paper identifies a family of formal semantics for the HP notation. There exist many precise interpretations that can be formalized and used in different application domains. The family of precise interpretations for the HP notation can be obtained by combining the different interpretations of the ambiguities described in this paper. The impossibility of indicating a unique precise interpretation explains the limited success of specific formalizations of structured analysis and explains the vagueness of the notations supported by commercial CASE tools. This paper suggests a new approach towards the formalization of structured analysis, which leads to the formalization of a family of interpretations to be adapted to the specific application domains, rather than the definition of a specific semantics.

ACKNOWLEDGMENTS

The authors would like to thank Carlo Ghezzi for his valuable suggestions, Hanne Christensen for the useful discussions and comments, the IDERS consortium for

Class	Description	Section	Page	Terminators	Processes	TC data flows	TT data flows	Control flows	Stores	PSPECs	CSPECs
Imprecisions	Input/Output flows	3.1.2	6		•	•	•	•	•		
	TT input data flows	3.1.2	6		•		•				
	Enabling condition	3.1.2	7		•						
	Process activation/deactivation	3.2.3	9		•						•
	Process duration	3.3	10		•					•	
Ambiguities	Functionalities control and timing of PSPECs	4.1.1	11							•	
	Input flows in excess	4.1.2	13			•	•		•	•	
	Nondeterminism in using input values	4.1.2	13			•	•		•	•	
	Unspecified values on output data flows	4.1.3	15			•			•	•	
	Internal timing conditions	4.2	15							•	
	Enabling of processes with more than one TT input flow	4.3.1	16		•		•				
	Execution of processes with only TC flows or stores as inputs	4.3.2	16		•	•			•		
	Processes with no inputs	4.3.2	16		•						•
	TT output flows of continuosly executing processes	4.4	16		•		•	•			
	Keyword “issue” and TT flows	4.4	16	•			•	•			
	Flows entering/leaving a deactivated process	4.5.1	17		•	•					
	Child elements of a deactivated process	4.5.2	17		•				•		•
	Simultaneous events entering a CSPEC	4.6	18					•			•
	Merging TT and TC flows	4.7.1	18			•	•				
	Overlapping type definitions of merging flows	4.7.2	18			•	•	•			
	Merging of multiple TT flows	4.7.3	19				•	•			
Extensions	Structured data stores	5.1	20						•		
	Structured flows	5.2	20			•					
	Timing (Process duration)	5.3	20		•						

Table 3. Summary of problems discussed in this paper (TC: time-continuous, TT: time-transient).

the cooperation supplied, and the "Cab Crew" for their daily help.

The work documented in this paper has been possible thanks to Structured Technology Group, that provided us with an evaluation copy of AxiomSys, Mark V Systems, that provided us with a copy of ObjectMaker, and the IDERS consortium, that provided us with a copy of StP.

We would like to thank the editor and the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Ardis, M. A., Chaves, J. A., Jagadeesan, L. J., Mataga, P., Puchol, C., Staskauskas, M. G., and Olnhausen, J. V. 1995. A Framework for Evaluating Specification Methods for Reactive Systems. In *Proceedings of the 17th IEEE International Conference on Software Engineering* (April 1995), pp. 159–168. IEEE-CS Press.

Baresi, L., Orso, A., and Pezzè, M. 1995. Customizable Notations for Kernel Formalisms. In *Proceedings of the 1st IEEE International Conference on Engineering of Complex Computer Systems* (Nov. 1995), pp. 43–46. IEEE-CS Press.

Bologna, S. 1993. *Incremental Prototyping Technology for Embedded Real-Time Systems*. Number 2/3 (5) in The Journal of Real-Time Systems, Special Issue. Kluwer Academic Publishers.

Bove, R., Dipoppa, G., and Perrotta, A. 1996. The RPCM System Core Specified and Animated Using IDERS. Technical Report IDERS-ALENIA-42-V2.2 (April), Alenia.

Bruyn, W., Jensen, R., Keskar, D., and Ward, P. T. 1988. ESML: An extended systems modeling language based on the data flow diagram. *ACM SIGSOFT Software Engineering Notes 13*, 1 (Jan.), 58–67.

Clarke, E. M. and Wing, J. M. 1996. Formal Methods: State of the Art and Future Directions. Technical Report CMU-CS-96-178 (Sept.), Carnagie Mellon University – School of Computer Science.

Craigen, D., Gerhart, S. L., and Ralston, T. J. 1993. An International Survey of Industrial Applications of Formal Methods. In *Z User Workshop, London 1992*, Workshops in Computing (1993), pp. 1–5. Springer-Verlag.

Crow, J. and Vito, B. L. D. 1996. Formalizing Space Shuttle Software Requirements. In *First Workshop on Formal Methods in Software Practice (FMSP '96)* (Jan. 1996), pp. 40–48. ACM.

Elmstrøm, R., Lintulampi, R., and Pezzè, M. 1993. Giving Semantics to SA/RT by Means of High-Level Timed Petri Nets. *Journal of Real-Time Systems 5*, 2/3 (May), 249–271.

Fencott, P. C., Galloway, A. J., Lockyer, M. A., O'Brien, S. J., and Pearson, S. 1994. Formalising the Semantics of Ward/Mellor SA/RT Essential Models using a Process Algebra. In *FME'94: Industrial Benefit of Formal Methods*, Volume 873 of *Lecture Notes in Computer Science* (1994), pp. 681–702. Formal Methods Europe: Springer-Verlag.

France, R. B. 1992. Semantically Extended Data Flow Diagrams: A Formal Specification Tool. *IEEE Transactions on Software Engineering 18*, 4, 329–346.

France, R. B. and Larrondo-Petrie, M. M. 1994. From Structured Analysis to Formal Specifications: State of the Theory. In *Proceedings of the ACM Computer Science Conference* (March 1994), pp. 249–256. ACM.

France, R. B. and Wu, J. 1995. Rigorous Dynamic Analysis of SA/RT Requirements Models. Technical Report TR-CSE-95-14, Department of Computer Science & Engineering - Florida Atlantic University.

Gerhart, S., Craigen, D., and Ralston, T. 1994. Experience with Formal Methods in Critical Systems. *IEEE Software 11*, 1 (Jan.), 21–28.

Goguen, J. A. and Winkler, T. 1988. Introducing OBJ3. Technical Report SRI-CSL-88-9 (Aug.), Computer Science Laboratory, SRI International.

Guttag, J. V., Horning, J. J., and Wing, J. M. 1985. Larch Family of Specification Languages. *IEEE Software 2*, 5 (Sept.), 24–36.

Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., and Trakhtenbrot, M. 1990. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering 16*, 4 (April), 403–414.

Hatley, D. J. and Pirbhai, I. A. 1987. *Strategies for Real-Time System Specification*. Dorset House, New York.

Hinchey, M. G. and Bowen, J. P. Eds. 1995. *Applications of Formal Methods*. Prentice Hall International Series in Computer Science.

Interactive Development Environments. 1994. *Structure Environment: Using the StP/SE Editors*. Interactive Development Environments. Release 5.

Jordan, D., McDermid, J. A., and Toyn, I. 1991. CADiZ – Computer Aided Design in Z. In *Z User Workshop, Oxford 1990*, Workshops in Computing (1991), pp. 93–104. Springer-Verlag.

Larsen, P. G., van Katwijk, J., Plat, N., Pronk, K., and Toetenel, H. 1991. SVDM: An Integrated Combination of SA and VDM. In *Methods Integration Conference* (Sept. 1991). `ftp://ftp.ifad.dk/pub/papers/vdm.ps.gz`.

M. D. Fraser, K. K. and Vaishnavi, V. K. 1991. Informal and Formal Requirements Specification Languages: Bridging the Gap. *IEEE Transactions on Software Engineering 17*, 5 (May), 454–466.

Marco, T. D. 1978. *Structured Analysis and System Specification*. Prentice-Hall.

Mark V Systems. 1994. *ObjectMaker User's Guide*. Mark V Systems. Version 3.

Meta Software Corporation. 1992. *Design/IDEF: User's Manual*. Meta Software Corporation.

Meyer, B. 1985. On Formalism in Specifications. *IEEE Software 2*, 1 (Jan.), 6–26.

Miranda, E. L. 1989. Specifying Control Transformations Through Petri Nets. *ACM SIGSOFT Software Engineering Notes 14*, 2 (April), 45–48.

Owre, S., Rushby, J., Shankar, N., and Henke, F. V. 1995. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering 21*, 2 (Feb.), 107–125.

Petersohn, C., Huizing, C., Peleska, J., and de Roever, W. P. 1994. Formal Semantics for Ward & Mellor's Transformation Schemas. In *6th Refinement Workshop*, Workshops in Computing (1994), pp. 14–41. BCS-FACS: Springer Verlag.

Richter, G. and Maffeo, B. 1993. Toward a Rigorous Interpretation of ESML - Extended Systems Modeling Language. *IEEE Transactions on Software Engineering 19*, 2 (Feb.), 165–180.

Semmens, L. T. and Allen, P. M. 1991. Using Yourdon and Z: An Approach to Formal Specification. In *Z User Workshop, Oxford 1990*, Workshops in Computing (Dec. 1991), pp. 228–253. Springer-Verlag.

Shi, L. and Nixon, P. 1996. An Improved Translation of SA/RT Specifications Model to High-Level Timed Petri Nets. In *FME'96: Industrial Benefit and Advances in Formal Methods*, Volume 1051 of *Lecture Notes in Computer Science* (March 1996), pp. 518–537. Formal Methods Europe: Springer-Verlag.

Structured Technology Group, Inc. 1995. *AxiomSys/AxiomDsn: Design CASE Tool, Tutorial*. Structured Technology Group, Inc.

Tuya, J., Sanchez, L., and Corrales, J. A. 1995. Using a Symbolic Model Checker to Verify Safety Properties in SA/RT Models. In *Proceedings of the 5th European Software Engineering Conference*, Number 989 in Lecture Notes in Computer Science (Sept. 1995), pp. 59–75. Springer-Verlag.

Ward, P. T. and Mellor, S. J. 1986. *Structured Development for Real-Time Systems*, Volume 1-3. Yourdon Press, New York.

Wing, J. and Zaremski, A. 1991. Unintrusive Ways to Integrate Formal Specifications in Practice. In *VDM 91: Formal Software Development Methods*, Volume 551 of *Lecture*

Notes in Computer Science (Oct. 1991), pp. 545–569. Springer-Verlag.

Wood, D. and Wood, W. 1989. Comparative Evaluations of Four Specification Methods for Real-Time Systems. Technical Report CMU/SEI-89-TR-36 ADA219187, Software Engineering Institute (Carnegie Mellon University).