Joint Workshop on Standards for the Use of Models that Define the Data and Processes of Information Systems
April, 22-25, Bellevue, Washington, USA

SYSTEM MODELLING

PHYSICAL CONCEPTS, THEORETICAL ELEMENTS, DESCRIPTION LANGUAGE

Serge Savoysky
Docteur ès Sciences

CONTENT

PART 1
EXECUTIVE SUMMARY

Formerly, this contribution dealt essentially with the ISO project n°1.21.63: this project concerns a « standard for the Conceptual Schema Modelling Facilities » (CSMF standard). It appears further that the items of the proposed methodology, outlined thereafter, may concern also some other ISO activities. Thus, this observation justifies the integral presentation of this study to the ISO meeting in Bellevue.

This study, all the time in improvement, was undertaken several years ago, with the technical or financial assistance of:

- *Laboratoire Central des Ponts et Chaussées (Ministère de l'Équipement)*[1],
- *Scétauroute*[2],
- *Club des Utilisateurs Bull Européens*[3].

A) Need of CSMF

System designers and users, in any organisation, have skills normally covering fields other than information technology; these fields constitute their Universe of Discourse (UoD). Consequently, if information technology is of unquestionable importance in any enterprise, it must always be combined with other techniques implemented for the same purpose with equal importance; too radical a separation between these other techniques and information technology, more particularly with regard to software engineering, would be unrealistic. The desired organisation calls for the availability of a method and practical facilities for assistance in the use of information technology in their current activities, in particular for the specification of the functions they assign to the various devices, and specially to the computers, within the systems concerning them.

The engineer having overall responsibility for IT equipment in his enterprise can only have a general attitude with regard to the diversity of techniques and domains involved; however he must have methodical and technical aids enabling him to act as an informed interlocutor of contracting firms, which are no less diverse in their respective specialities, and who he will have to call upon. Consequently, we have chosen the hypothesis of an engineer specialising in a particular engineering domain but not a specialist of the many techniques concerning information technology.

Every engineer required to work on a system or on a component of a system, obviously needs any information existing on this item when performing his work. In addition, information concerning the rest of the system may also be useful to him. Thus, before doing any action on the system, he must consult a volume of information whose size increases with the size and with the complexity of the system

Simulation is the technique usually employed up to the present time to aid engineers in certain types of work. This technique, used for many years, long before the advent of data processing, makes use of means which are either created empirically or deduced by analogy or by the transformation of theoretical models of the items to be simulated. Data processing subsequently favoured the use of numerical simulation, since any program intended for calculation on the values of state parameters of any real item constitutes, in fact, a numerical simulation of this item.

And, the purposes of a theoretical model involve aspects other than numerical simulation, even if data processing has found in this area a privileged field of application: other uses of models are possible. These uses constitute the domain of « Computed Aided X » where X represents different possible activities: design, maintenance, technician teaching, fault diagnostic… From this viewpoint, it should be noted that the notion of model has taken on a new connotation. Formerly, a model, expressed in the form of a program, was a technique intended for data processing. It has quickly become data itself, to be processed by techniques developed for aiding engineers in their activities[4].

These objectives depend on the existence of strict modelling facilities: the CSMFs which provide conceptual definitions, language and software tools (Figure 1).

---

1 The L.C.P.C. is the Main Road and Bridge Research Center of the French Ministry of Equipment.
2 Highway Engineering Office.
3 European Bull Users Association.
4 Savoysky: 1975, xxxiii ; 1981, xxxv.

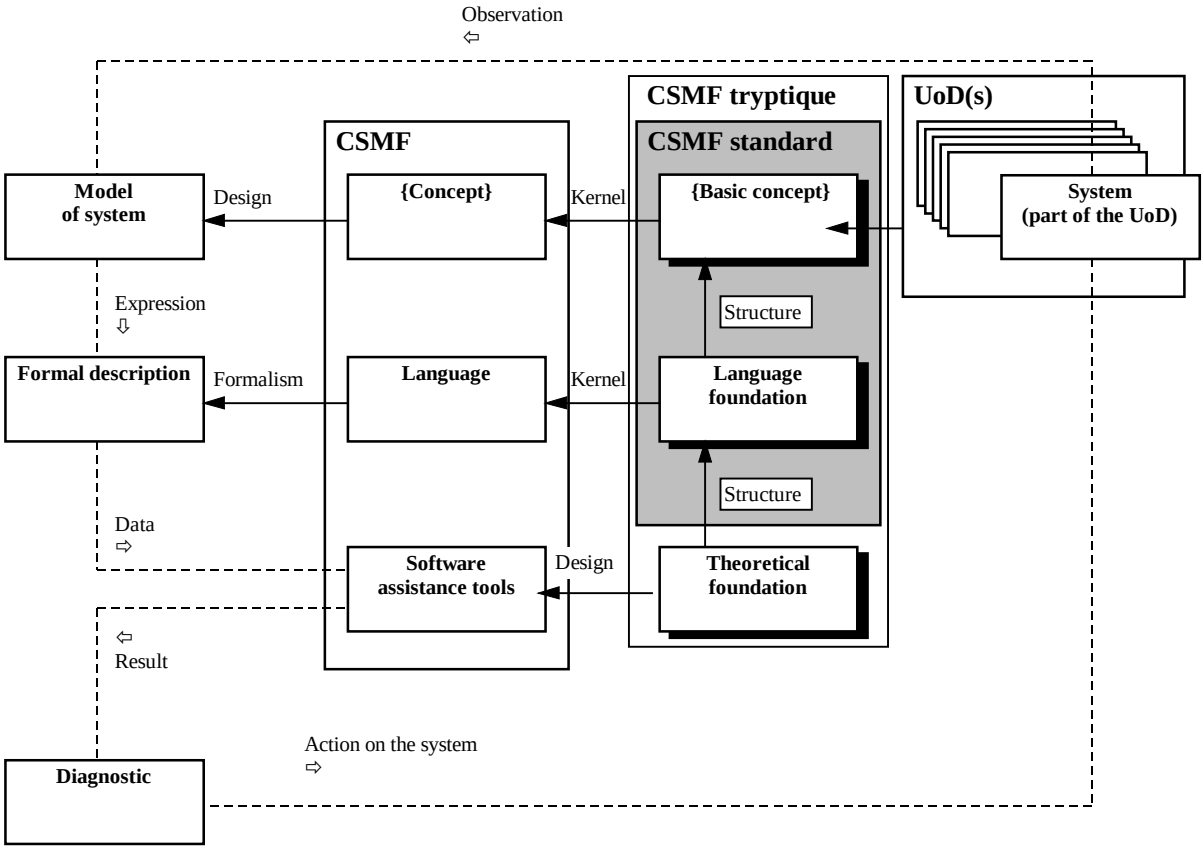


Figure 1: Relation within the CSMF tryptique and its uses

B) Need of standard for CSMFs

The development of relationships between organisations involves a correlative elaboration of links between their systems. There is the origin of the concept of open system.

At any time, a manager may have to guarantee the safety of his system; nowadays, the idea of a totally closed system, i.e. isolated from its environment, is an abstraction. On the contrary, the daily extension of the collection of functions of this system outward, by linking one or more surrounding existing systems, becomes usual.

The accumulation of these exchanges between systems requires the development of common practices for facilitating them.

C) Structure of the study

The next part «Physical concepts » determines, in a an natural language, an unitary and reduced system of concepts and method for producing assemblages of concepts; however, all the items of this approach may be modelled with the further theoretical elements. So, and because mathematics have the main advantage of providing strict models, the following part « Theoretical elements » proposes a structured collection of mathematical elements as a fundamental instrument for this general purpose. Let us note that this approach complies with IRDS[5]. The last part « Description language » determines a short formal etymology method for expressing the precedent concepts; for fixing their axiomatic origins or their relative positions in relation to other concepts; for allowing the comparison between them and other different existing definitions, and finally for introducing a formal definition method of description languages.

The two parts « Physical concepts » and « Description language » show a possible way for the CSMF standard study. The part « Theoretical elements » belongs to mathematics; thus it is universal and has not to be standardised; nevertheless it shows how understandable elementary mathematical concepts may support IT standard without strenuous elaboration. The Figure 2 shows the relationships between the different parts of the study and indicates those of these parts which may belong to a standard document (dashed surfaces).

5 IRDS - part1, pp27/28.

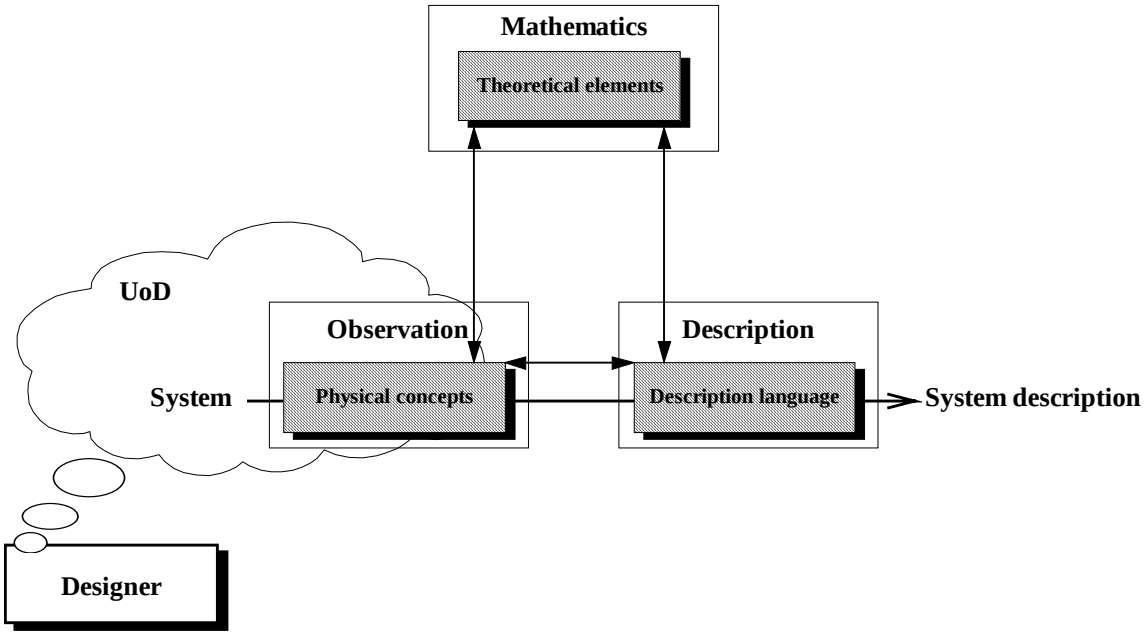


Figure 2: Structure of the study

PART 2
PHYSICAL CONCEPTS (TIN FRAME)

Pourquoy est ce que nostre langage commun, si aysé à tout aultre usage, devient obscur et non intelligible en contract et testament ; et que celuy qui s'exprime si clairement, quoy qu'il die et escrive, ne trouve en cela aulcune maniere de se declarer qui ne tumbe en doubte et contradiction ?…
Montaigne, Les Essais, De l'experience.

1 GENERAL

✉ **Multiple definitions**. Some other similar definitions of the concepts described hereafter are collected in Appendix I. They are stated in different normative documents. If necessary, their existence is pointed out with the symbol heading this note.

It is upon the establishment of some universal simple concepts that lies our only chance of standardisation of system description. The notions developed below, necessarily includes the plurality of techniques used; it is in relation with this broader acceptance of them, that the notion of integration must be considered. They also take in account the plurality of UoD such as: banking, transportation, industrial process control, research, etc.

In that order, the main purpose of this informal chapter is to propose some basic, neutral and general concepts having the theoretical elements proposed in the following part as sound foundations. The principles of their current expression will be examined in the next part. The choice and the definition of basic concepts are justified by common and global needs of system designers dealing with different fields of activities.

2 DEFINITIONS OF PRIMARY COMPONENTS OF SYSTEMS

2.1 General

✉ Multiple definitions: **object** (Page 55)

The concepts are expressed there from an user's viewpoint. Some will at times appear naive to readers. This apparent naivety is however deceptive because the assemblies subsequently constructed using these notions are often complex; a property of information technology resides in the fact that it favours the growth of systems in which the operating logic surreptitiously exceeds human understanding. It is thus important to pay rigorous attention to the definition of basic notions, even the simplest or most evident, thus hopefully avoiding the risk of subsequent ambiguities.

A terminological complication appears. First, the observable components belong to the real world supporting the UoD. Nevertheless, each time we have to study such a component, we must use a representation: the need to describe a component arises just when it comes into consideration for study or for use. A representation may be: a description in a natural language, an image, a model using a sound formalism, etc. Any representation now is also a component and so forth. For that reason the word « component » has a wider acceptation: the same word may be used for speaking of a real entity or for its representation. However, any activity which may concerns us, produces essentially component descriptions, at different stages of completion, and not the real components themselves. Then, in the absence of ambiguity and to avoid abusive language, the

term « component » will be used in place of the corresponding expression « component description ». The same assumption is done for: system, module, process, produce[6], etc.

2.2 System

2.2.1 General definitions

✉ Multiple definitions: **system** (Page 55)

A system is by convention specific to a domain of activity; it is a dependent part of real world.

concept 1: A **system** is a collection of components called devices (concept 2), individually processing and exchanging other components called produces (concept 3), with each other and with the environment of the system[7], in order to achieve the execution of a production or of a service of an industrial nature:

concept 2: A **device** is a component of a system or a part of a system; it represents a set of coherent functionalities (concept 4)[8].

concept 3: A **produce** is a component handled by a device as input or output of its functionalities or exchanged between devices.

This distinction between device and produce is arbitrary: we will further show that a device may be also viewed as a produce and reciprocally. One may object that there is no need for two words to express the same concept and, that there is a vain terminological complication. In spite of these remarks, this terminological distinction will be justified further by the distinction of components having different positions in the system.

2.2.2 General case: physical system

For different purposes, every physical system must be analysed. This analysis is related to the formal system chosen by the analyst, and enabling the result to be expressed in the form of a description[9]. This descriptive aspect, though closely related to the analytical aspect, will sometimes be examined separately in this chapter.

This chapter deals with the general case of real systems integrating elements of information technology, without being limited to systems made up exclusively of such elements. For the designer of a system or its subsequent operator, knowledge of the system and of the physical laws governing it must prevail over that of the elements of information technology used as tools. The notions previously introduced and all that which will be introduced later are usual in certain fields of activity, such as conversion industries. They are generalisable: an automated shop is thus analysable, but so also is the entire company in general and any software in particular. Examples 1 to 12 beneath illustrate the diversity of the types of physical systems, from a top level to a very low level, falling within the scope of this chapter.

---

6 In this version, we prefer to use the word « produce » rather than the word « product » when we are speaking of physical system element. This choice eliminates the polysemy due to the use of the physical acceptation and the mathematical acceptation of « product » in the same paper.

7 Observing the outside, every user module is generally affected by the existence of an environment with which it is destined to interact, but which escapes the control of the system manager. The idea of a totally closed system, i.e. isolated from its environment, is an abstraction: the negation of any exchange of a given system with the rest of the world is a working hypothesis which is always possible but dangerous, especially when its safety is specified. The extension of the system outward by introducing one or more surrounding modules results from this requirement. The specification of these modules is most often equivalent to describe passively the constraints they impose, sometimes outside of standards.

8 Preference is given to the neologism "functionality" over the term "function" whose sense, fixed strictly in mathematics and formal linguistics, is too restrictive for general use.

9 The use of several formal systems is current practice; this is the case, for example, of an algorithm whose expression in a given software exists in several formal systems: high-level language, intermediate machine languages, binary language.

Example 1: A network of motorways, with its infrastructure and its appurtenances, its users and its information system.
Example 2: A multi-purpose vehicle for the measurement of pavement properties.
Example 3: A bridge deck.
Example 4: A bridge deck raising site.
Example 5: An information system.
Example 6: A communications network.
Example 7: A microcomputer and its software.
Example 8: An office automation system.
Example 9: An accounting software.
Example 10: A data acquisition and interpretation system.
Example 11: An " intelligent " actuator (hardware and system with built-in microprocessor for control and communication).
Example 12: A sensor.

The theoretical elements defined in the next part, are intended to support the modelling of such entities.

The listing of elements distinguished by analysis constitutes the **morphology** (concepts 17, 19) of the system; the expression of the rules of individual or mutual behaviour of the elements constitutes the **physiology** (concepts 18, 25) or different aspects of the **behaviour** of the system (concepts 18, 26). A physical system is never isolated: there is at least one process receiving at least one produce from the outside; similarly, there is at least one process sending at least one produce to the outside. This outside, about which there is often scant knowledge, always different from what is known about the physical system, is the result of a generally distinct approach; it is called environment of the system[10].

The notion of **subsystem**, also called **module**, is introduced in this set of basic concepts in order to facilitate the hierarchical classification of components belonging to the same system or to its environment.

Example 13: The part of a system providing communication between all the elements of this system may be set apart as a module.

2.2.3 Special case: information technology system

Some modules may be made up exclusively of elements pertaining to information technology. Any module of this type may be isolated conventionally from the rest of the system; it is then called an information technology system, and the physical system within which it is integrated becomes its environment.

In spite of the importance currently attached to these particular elements, it appears clearly that they constitute neither its morphological totality, nor its physiological finality. It is valuable to the architect engineer, at any time in the life cycle of the physical system, to have methods and means of action taking into account the presence of information technology, but which are not restricted thereto. Two Examples show the relative nature of the notion of information technology system. They show the plurality of relations linking these systems and their respective environments and the resulting diversity of standardisation requirements.

Example 14: An information system (Example 5, Page 13) is a possible module in a motorway network (Example 1, Page 13). It is generally made up of elements of information technology and, in this case, considered to be an information technology system.
Example 15: In a motorway network (Example 1, Page 13) for which many data acquisition operations are to be carried out using appropriate instrumentation, every acquisition and interpretation chain (Example 10 Page 13) is assimilable to an element integrated in the operating equipment (toll, signing, control, etc.) and specific to this equipment. On the other hand, in a bridge deck raising operation (Example 4 Page 13), the chain that acquires and restores in real time to the operator all the information relative to the behaviour of the structure is a sophisticated module considered by the construction project head to be an information technology system.

An information system embedded in any physical system is certainly nowadays one of the masterpieces of this physical system. This kind of module is directly issued from the use of some CSMFs[11].

---

10 The notions of morphology and physiology are commonplace as concerns living systems. Maurice d'Ocagne was apparently the first to explicitly transpose these notions to the field of mechanical calculation while commenting the thesis of Louis Couffignal (*in : Histoire abrégée des sciences mathématiques, Vuibert, Paris, 1952, pp. 389/391*). These notions are clearly present in information technology, especially in the software industry; usually, they are however distinguished other than by their usual names.

11 The existence of such information system is possibly the best reason for arguing in favor of the integration of this study within the TC21 WG3 activity.

2.3 Functionality

2.3.1 Definition

A functionality F is the most elementary form of a device. The use of this neologism is preferred to that of function, because the properties assigned to it are more general than those of the notion of function[12].

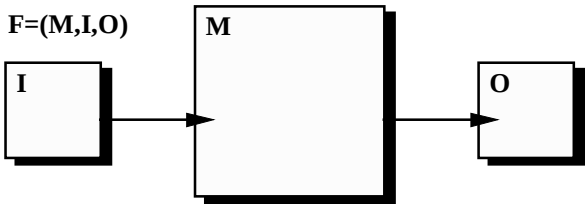


Figure 3: Functionality

concept 4: The classical diagram of Figure 3 represents graphically three constituent elements of a **functionality**[13]:
concept 5: its **body** (M) which is a device,
concept 6: the specification of produces (I) placed at its **inputs**,
concept 7: the specification of produces (O) placed at its **outputs**.

The functionality moreover supports general conditions or constraints imposed upon it: these are the known physical laws applicable to this functionality:

concept 8: those concerning I are **pre-conditions**;
concept 9: those concerning O are **post-conditions**;
concept 10: those concerning both I and O and mutually controlling them are **invariants**[14].

The notion of functionality is recursive: any functionality defined after an analysis stage can, in turn, be analysed in new functionalities. At the start, before any analysis work, the entire physical system constitutes itself a functionality. From the simple viewpoint of the approach we are describing, the notions of system and functionality are thus identical; we shall however distinguish them conventionally, leaving the notion of system at the start of the analysis and placing the notion of functionality at the different stages of the analysis.

2.3.2 Relations between functionalities: exchange of produces, interoperability

⊠ Multiple definitions: **interoperability** (Page 54)

The interactions between devices of the real physical system are possible only if exchanges of produces exist between these devices. The relations between functionalities describe these interactions. Yet, these interactions and consequently the relations that express them depend on various physical laws or conventions applicable to each examined part. The only exception is the

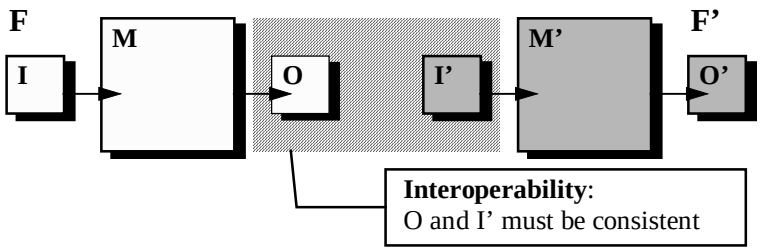


Figure 4: Exchange of produces

12 In fact, this neologism is totally unnecessary: the algebraic notion of functor accounts for the properties attributed to this diagram. This term could thus be used.

13 This well-known diagram is currently used. Mathematical models are less used. Let us however note that the algebraic notions of category and functor are used for modelling E, S and M. The advantage of this approach is that it accounts in a global and standard manner for the morphological and physiological properties of these terms, whatever their respective physical natures.

14 These constraints often remain implicit - or ignored - in system specifications or descriptions. The reason is that many items of equipment are designed and constructed knowing these laws; they consequently protect the uninformed user from their possibly harmful effects with the reserve that this user indeed complies with the validity limits indicated by the supplier. These validity limits are in most cases broad, leading to a false impression of safety, and then negligence, and finally involuntarily abusive usages involving immediate or latent errors. There are many instances in instrumentation, and simply reasoning by induction enables us, for example, to draw up a sizeable inventory of deleterious effects.

relation, omnipresent in all systems, expressing the existence of an exchange to the exclusion of all other properties, for any couple made up of an output O of a given functionality F and an input I' of any functionality F' [15].

The specifications of O and I' are generally different but must obviously be consistent: the template of I' must be compatible with that of O. Two questions must be answered:

- define consistency criteria for the specifications between O and I', which criteria do not exist in the absolute, but depend most often on the couple (F, F'),
- provide the practical means of verifying that this consistency exists in the specifications for any exchange, the verification quickly becoming complicated with the number of exchanges[16]. That is one of the major problems of system validation.

concept 11: The consistency of O in F with I' in F' determines the quality of **interoperability**[17] of the couple (F, F').

Let us note in this respect that interoperability, thus defined, is a quality attached to a couple of functionalities and not intrinsic to a single functionality. Yet, there is a lack of definition: this definition of interoperability lies upon the precedent definition of consistency which is not clearly stated above. Here is an example of dubious informal definition which needs the assistance of mathematics for clarifying it.

Example 16: F is a sensor (Example 12, Page 13). I is a physical magnitude: the frequency at which vehicles cross a given point of a pavement; M a device such as an induction loop embedded in a pavement and its associated circuits; M delivers a voltage O of extremely low amplitude whose derivative changes sign with a frequency proportional to the value of the magnitude I measured. In other words, O is a train of "unformatted" pulses. It should be noted here that O must be regarded as a DC voltage whose evolution depends only on variations in I and can therefore not obey any standard, by definition and construction. F' is a measurement chain including, among other functionalities, a pulse counter reacting only on the edges of variations in I' and imposing by construction of M' a minimum restoring time between two successive edges. The two templates present are thus totally different, but may be consistent: the rising or falling edges of O must be exhaustively identifiable by M' [18].

Example 17: F is the counting device described above (Example 16). O is a structure of data including dates, times and values of counts. F' is an information system (Example 5, Page 13). I' is a structure of digital data made up of fields allowing the recording of a nature of variable, its value, its acquisition attributes. M' is a set of functionalities allowing the acquisition, centralisation, management and finally selection by interrogation of such data structures. O' is a structure identical to I'. Consistency requires, in this case, that the structure O be included in I'.

2.3.3 Relations between functionalities: translation of a functionality into another, portability

✉ Multiple definitions: **portability** (Page 55)

The definition of
F: M(I,O)
consists in defining the properties required for each of these elements as well as the constraints. The body M, its inputs I, its outputs O, then have an expression in a formal system S (Figure 5). This expression may be translated in another formal system S':
F': M'(I',O')
F' is a new functionality whose properties comply with the conditions previously specified:

concept 12: The specifications of the functionality F to be transported determine a template; the **portability** of F implies that the features of every implementation must fall within this template.

---

15 The Graph theory can be used here; it is then important to clearly specify the interpretation adopted for the apexes and the arcs. In the approach of this paragraph, the apexes are inputs or outputs of functionalities. The arcs represent recent exchanges between functionalities but also the functionalities themselves, which establish links between their own inputs and outputs.

16 This consistency check between functionalities (in the specifications) or between components making up these functionalities (in the solutions) calls for formal verification methods. We shall return to this point in the last chapter.

17 Neologism (F: *interoperabilité*).

18 This Example may be considered as a "butterfly" effect in automatic control. It is a point of detail obviously ignored by owners. It entails a latent risk of inadvertently false measurements having, for example, the real-time result of failure or breakage of equipment and always resulting in financial losses incommensurate with the relatively modest cost of the sensor.

Formally, a good solution consists in proposing such transformations T, U, that I'=T(I), M'=U(M) and such a transformation V that, with its inverse[19], O=V^{-1}(O'). As a functionality is a description of a real element of a system and not the element itself, the operation described above can be assimilated with the passage from a first assembly in a formal system to a second assembly, image of the preceding, in another formal system.

This diagram gives an image of **portability**[20]. Let us assume the existence of a first functionality F which must be implemented in different formal systems S', S '', etc. If F' in S', F'' in S'', etc. exist, then F is transportable from S' to S'', etc.

Example 18: F is a program expressed in a high level language S, and F' is the same program translated into the language S' of the computer.

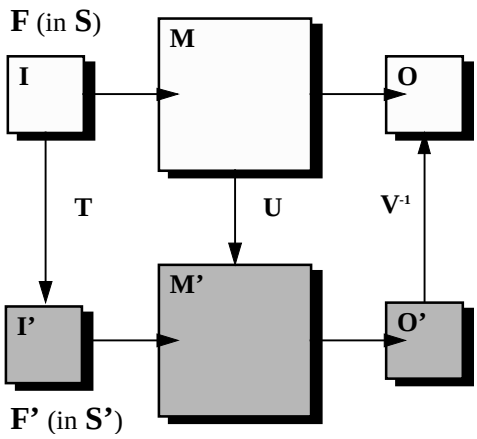


Figure 5: Implementation

2.3.4 Interface

⊠ Multiple definitions: **interface** (Page 54)

The preceding discussions show the importance of the notion of consistency in the definition of system specifications as well as in the synthesis of a solution. The questions of consistency remain when reference is made to standards for specifications, and when elements of standardised solutions are used during construction. The notion of interface is developed in Standardisation to answer these questions.

The interface is, by convention, a set of specifications determining the characteristics that must be met by functionalities in order to proceed with exchanges. Two types of interfaces thus correspond to the two types of relations determined in the first two chapters.

a) Interface of the first type

concept 13: The interface of the first type (Figure 6) is a set of conventions concerning exchanges between two functionalities F and F'. In addition to the definition of the structured set of informations to be linked (O and I'), these conventions must also, if required, include time-dependent specifications as well as qualitative specifications (e.g.: safety) concerning this information.

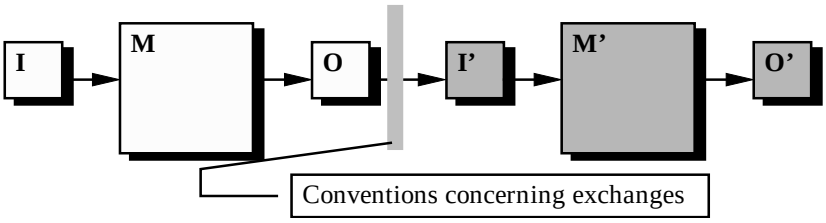


Figure 6: Interface. First type

Let us note that the lack of consistency between O and I' occurs frequently in heterogeneous system. This implies the insertion halfway to M and M' of a mechanism which is well a component which ensures the compliance of O with I'.

---

19 Algebraically, the diagram of Figure 5 "commutes": O=V-1(M'(T(I)) = M(I) and M' = U(M). This strictly theoretical changeover is in practice not feasible; the approximate result O* = V-1(O') must then be compared with the template of O. Algebra makes it possible to rigorously express these conditions and then to construct the formal verification algorithms. This observation, which would appear of no value in the case of simple figures, takes its full importance in the case of functionalities obtained by combinations.

20 This basic diagram may also represent a relative definition of the notions of user and supplier: the user is the individual or organisation which specifies a functionality and, consequently, requires a service; the supplier is the one who proposes and constructs the solution providing the required service. Such a diagram can be used iteratively: according to this remark, there is no absolute definition of user or supplier; in particular, the notion of end user is always a conventional one.

b) Interface of second type

concept 14: The interface of the second type (Figure 7) is a set of conventions governing the transformations from an expression (I,M,O) pertaining to a formalism S into a translation (I',M',O') pertaining to an other formalism S'. The conventions involve produces I, I' and O, O' to be processed by the functionalities with, in addition the functionalities M, M' themselves handling these informations.

Let us note that in this matter, an component serviceable for performing these transformations, and also if necessary their reverses, must exist.

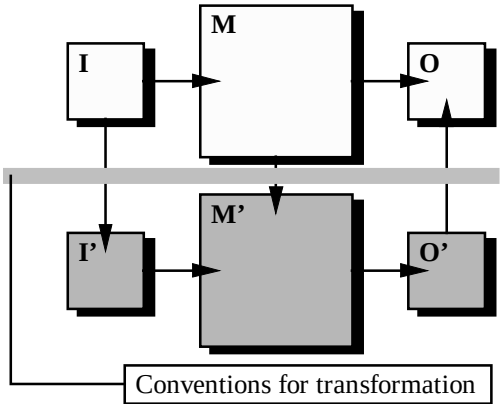


Figure7: Interface of the second type

2.4 Device and produce

A device is a structured collection of functionalities. It has an interest in a system if and only if it is able to interact with other devices. This condition means that the component has, at least, one facial functionality.

concept 15: A functionality F can be seen from another functionality F' if it may send at least one produce towards this functionality or if it may receive at least one produce from it. F is a **facial functionality** if F' may belong to another component.

concept 16: The collection of facial functionalities of a device constitutes its **face** or its **visible part**.

This definition is simple. It assimilates an device to a black box having some points of exchange with its environment. A produce is a component exchanged between devices or performed by a device. A produce may be perceived as a functionality in its simplest form considering the physical absence of input and output specification. A device may be considered as a produce and exchanged between devices or performed by an other device. Inputs and outputs of functionality are produces.

Example 19: A program or data exchanged between two computers.
Example 20: A program processed by a translator.

3 DESIGN OF HETEROGENEOUS COMPONENTS OF A SYSTEM

3.1 General

In treating of any part of a system, we must deal with it in two ways. We must deal with its observable structure made of heterogeneous components and its possible variations in time; we must also deal with its present and future possible uses. That is to say we have to study it morphologically and also physiologicallly.

Two ways for system designing are examined: analysis (top-down) and synthesis (bottom-up).

3.2 Analysis

3.2.1 General definitions

✉ Multiple definitions: **behaviour** (Page 54)

The analysis of a system or of a device makes it possible to distinguish iteratively, gradually, the devices and produces composing it. Let us summarise the basic elements of the approach. A system is a coherent whole of parts assembled to achieve a common goal. The first step in the analyse of a system is the separation of components and the recognition of relationships between them. After, each component may in turn be regarded as a system. The model and its expression must express this general structural aspect. Correspondingly we agree to build our models in two parts: morphology and physiology.

concept 17: the definition of a **components collection** which compose it; these components are defined globally without analysing their respective details, outside of the inventory of the produces they are liable to exchange; we name this collection: morphology (concept 19) ;.

concept 18: the definition of **rules collection** which determines the interactions between these components; these rules constitute the behaviour of the analysed system; we name also this collection: physiology (concept 25);

3.2.2 Morphology

concept 19 The **primary morphology** of a system includes the following elements, which are all components of the system:

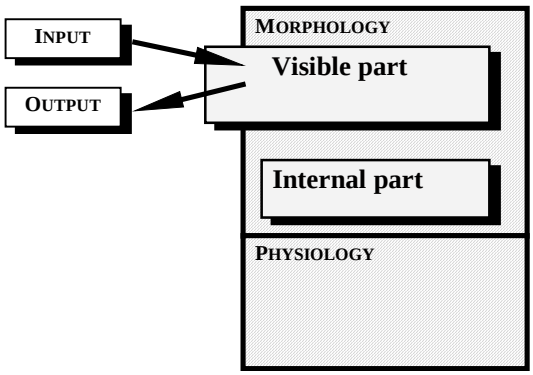


Figure 8: Component

concept 20: **produces**;

concept 21: **devices** used for handling the flows of produces or converting these produces.

Each produce may in turn be regarded as a system having iteratively at its level a primary morphology and so on…

concept 22: the **exhaustive morphology** results from this primary morphology, and in addition from the primary morphology of the different components belonging to it and so on… . In order to simplify the text, a primary morphology will be called shortly morphology.

Example 21: Flows of energy, materials, information, programs, and so on.

Example 22: Construction site equipment, industrial computer interfaces, programmes, and so forth.

Let us note that certain elements may belong to both categories mentioned above. Finally,

concept 23: some elements of a morphology are known from outside the device and constitute its **visible part**,

concept 24: while others constitute **its internal part**.

Example 23: A sensor, an actuator may be viewed as elements of the visible part of a morphology.

3.2.3 Physiology

✉ Multiple definitions: **action** (Page 54)

concept 25: A **primary behaviour** (or **primary physiology**) is a collection of actions

concept 26: the **exhaustive behaviour** (or **exhaustive physiology**) results from this primary physiology, and in addition from the primary physiology of the different components belonging to it and so on…

Each element of a physiology, called an action specification, must specify the conditions for activating each device and must express, among other conditions, the time-related constraints to which it is subjected.

3.2.4 State of a component

✉ Multiple definitions: **state** (Page 55)

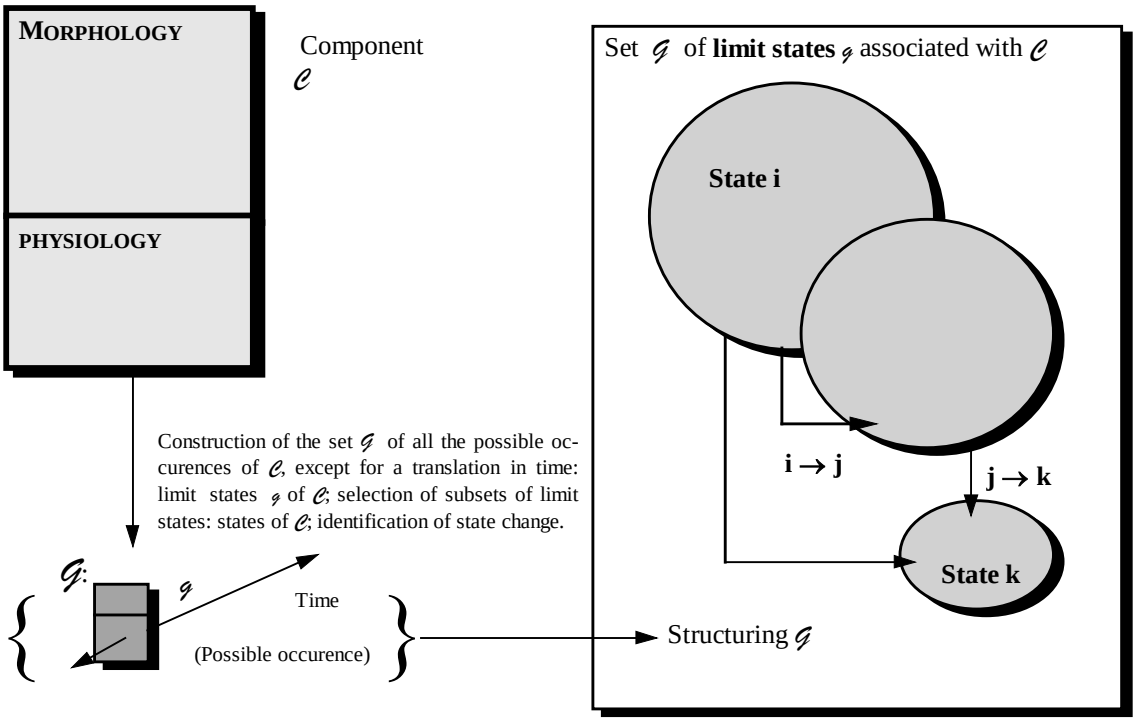


Figure 9: State, change of state

The morphology and the physiology of an component constitute both a general description of all the possible manifestations of this component.

concept 27: A description of a particular manifestation of the component is called an **occurrence**.
concept 28: An occurrence, expressed within a translation with respect to time, is called: **limit sate** of the component.
concept 29: A **sample** is a part, during any time interval, of any limit state of the component.

The set of all possible limit states is associated with the component. This set is structurable (Figure 9). In particular, two main classes of parts may be distinguished:

- the part made of limit states usually pertinent for the UoD,
- the complementary part.

concept 30: Every subset of the class of pertinent limit states is called a **state** of the component. Any set or subsets may be represented by one of its elements.
concept 31: Any subset belonging to the complementary part is an **exception**.

3.2.5 Recursivity of morphology and physiology

The general structure above can be used for the analysis or the description of a system or of a device. In each case, the final result appears as a hierarchical description of the system in which each level of the hierarchy provides a detailed description of the components introduced into the morphology of the preceding level.

Each level of the hierarchy consequently introduces descriptions containing details which are added to those expressed in the preceding levels. There is a limit to this procedure: the specificity of the elements introduced at each level is asserted as the analysis progresses in detail; after a certain level of analyse, three cases appear as possible:

- the description remains possible and pertinent in the proposed global formalism,
- the description remains possible in the proposed formalism, becomes inadequate or insufficient owing to the specificity of the new properties introduced,
- the description is impossible, owing to insufficient knowledge about the element, or is considered to be unnecessary.

In the analysis of a heterogeneous system, one should always be in a position to stop its description in the global formalism and to specify if necessary the existence of a description in another formalism. This is a difficult formal problem for the definition of consistent languages.

3.3 Synthesis

The synthesis is a bottom-up mechanism for designing a system. The synthesis of an component consists:

- in creating a complete morphology (visible and internal parts) by means of previously defined components and if necessary of new components,
- in associating the definition of physiology with these morphologies.

It is then necessary to examine how the combination of these components can function. Several cases may be considered, and we shall mention three essential ones:

(i) The definition of components is sufficient for the complete definition of the association (Figure 10, 1st case).

(ii) The definitions of components are insufficient to ensure a complete definition of their association; the definitions are completed by the definition of a supplementary component which, associated with the initial components, brings us back to the preceding case (Figure 10, 2nd case). In this case, the exhaustive behaviour of the whole results from the physiologies of the embedded components.

(iii) The complement to the definitions is provided by a new specific physiology (Figure 10, 3d case); let us note in this case that the whole behaviour of the new component results not only of this new physiology but also from the physiologies of the embedded components.

(iv) The definition of the new component results from a combination of the former cases.

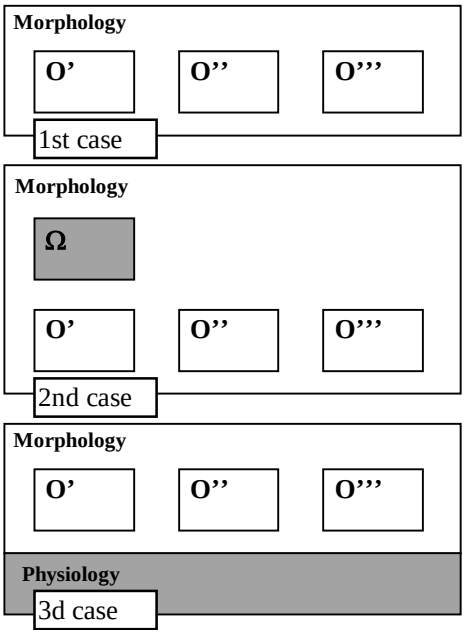


Figure 10: Synthesis of component

Example 24: A collection of asynchronous parallel tasks (1st case).
Example 25: A collection of synchronised parallel tasks; Ω is a clock (2nd case).
Example 26: A collection of scheduled tasks. The physiology rules the schedule (3d case).

3.4 Component typology

✉ Multiple definitions: **class** (Page 54), **type** (Page 55)

concept 32: Component **typology** consists to allow the expression of collectivising relationships between components to be defined later.

Two ways of collectivisation ruling are possible:

- Enumeration and description of each state available for all the components belonging to the set.
- Expression of the laws defining the states and allowing the construction or the recognition of any component having states in compliance with these rules.

concept 33: The collectivisation rules determines a **type**.

We may associate with any type, the set of all possible components complying with the chosen type.

concept 34: Any existent component belonging to a system is an **instantiation** of the type with which this component complies. This component is also an element of the set associated with its type.

concept 35: A **formal component** is a component defined away in state and time and place, that means independently of any actualisation

concept 36: An **actual component** is the specification of a discernible possible occurrence: the use of this component implies the discrimination of a recognisable state, and rigorous locations in time and in space.

concept 37: Sometimes a particular component may be chosen in order to define and further to represent the type: it is a **prototype**. In this case the typology must associate to this component, the collectivisation rules necessary for defining the type.

concept 38: The morphology of a prototype is a **protomorphology**.

concept 39: The physiology of a prototype is a **protophysiology**.

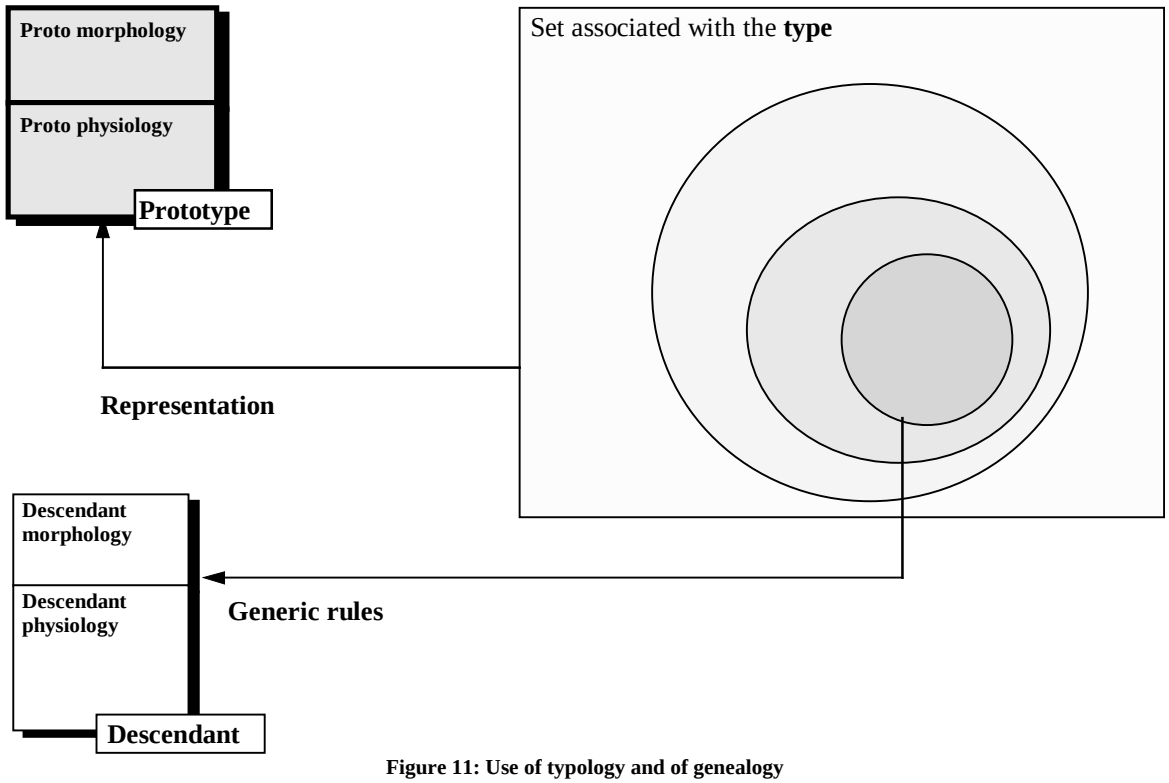


Figure 11: Use of typology and of genealogy

3.5 Component genealogy

✉ Multiple definitions: **inheritance** (Page 54)

concept 40: Component **genealogy** to define a type and to use it as a root or as a parent for further definitions using inheritance rules.

Example 27: An analogue voltage signal is an Example of type of component. The chosen prototype may be the particular signal used for the tuning of apparatus. The sampling is an Example of generic rule. In this case, the descendant type is the set of sampled voltage signals.

Example 28: Let us note that from this approach, the classical subtyping in programming language is a particular and limit aspect of generation; in this case, the generic rules are used only in order to restrict the previous set of states.

concept 41: A parentis an initial or intermediate item of a genealogy used as a basis for applying the inheritance rules

concept 42: The morphology of a parent is a **parent-morphology**

concept 43: The physiology of a parent is a **parent-physiology**

concept 44: A descendant is a final or intermediate item of a genealogy which results of an application of the inheritance rules

concept 45: The morphology of a descendant is a **descendant-morphology**

concept 46: The physiology of a descendant is a **descendant-physiology**

A particular component may be chosen in order to represent the type. In this case, this component is a generic component. These rules of generation may concern either the morphology, either the physiology. The result of application of generic rules is ordinarily a type. However, the set associated with the type may be reduced to a singleton and used implicitly as a single component.

PART 3
THEORETICAL ELEMENTS (GOLD FRAME)

Et ce moyen est le projet que j'ai d'une langue ou écrite nouvelle … ; et non seulement on trouverait là-dedans des voies infaillibles pour arriver à la solution des problèmes qui peuvent se résoudre par la seule force du raisonnement, mais lors même qu'il s'agit d'une question de fait, et qu'il reste encore des expériences à faire qui ne sont pas toujours dans le pouvoir des hommes, ce calcul serait suffisant pour nous conduire, en attendant, sur les connaissances déjà données.
Leibnitz, lettre à Jean Frédéric, 1679.

1 PROLEGOMENA

Physical systems are generally described in terms of operations relating physical items. The logical complexity of actual systems calls for simple descriptions in terms of symbolic expressions modelling these items and their relationships. Mathematics and particularly algebra, deals with this purpose.

This part concerns the use of algebraic notions, and particularly those of category theory, in the modelling of systems and, more particularly, in the modelling of those belonging to the field of application of ISO standardisation work.

a) Modelling and vocabulary

The modelling method proposed concerns the representation of systems made up of entities belonging to the physical world by means of systems made up of abstract entities belonging to mathematics. The changeover from one to the other is based upon the use of hypotheses adopting, for the physical units, properties assigned to the entities of mathematics.

The words designating the entities of the physical world[21] are of current usage; they are however chosen, to the extent possible, preferably from a vocabulary not customary to standardisation; this is seemingly complicated, but the advantages will appear subsequently when we shall apply the results of this work to standardisation work in progress.

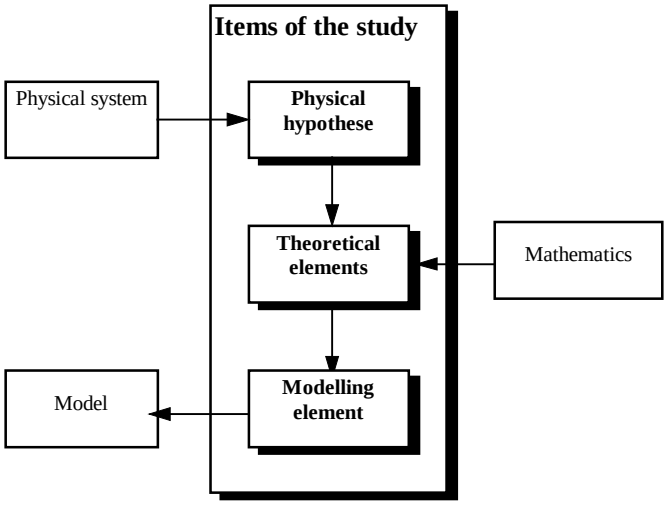


Figure 12: Purpose of the study

Words designating abstract entities are those set by usage in mathematics. We use them in this note without any restriction on their mathematical meaning. This option unavoidably leads to risks of polysemy, as the modelling concerning the domains use these words with different meanings. A fearsome example is that of the "**object**" word wich has different acceptations in different domains: a component of a category in algebra, a tangible thing of the real world, a concept for system designing in data processing, etc. These situations are dealt with respectively as they appear.

Words designating the modelling elements are introduced by the above-mentioned hypotheses or by definitions.

Axioms and theorems appearing in appendices come from texts cited in the references.

---

21 Universe of Discourse (UoD).

b) Modelling and graphic representation

Figures accompany the text. They allow fast and intuitive interpretation of the abstract mathematical notions developed. Although consistent with the text, they are sometimes fundamentally restrictive. Recommended to the reader who wishes to conserve a general idea of the proposed approach, they are not recommended for those wishing to use and possibly go beyond the reasoning proposed. For the informed reader, there are, for these figures, approximately the same restrictions as those existing in the particular case of multilinear algebra and its representations in affine Euclidean spaces.

c) Purpose and limits of theoretical elements

This part proposes modelling elements and, consecutively, a method for model construction. Unlike the usage which is unfortunately widespread, this is not one model, which would thus be universal for the investigated domain: there are in fact potentially many more possible models than entities.

The diversity of the entities of the physical world counter the working out of modelling elements making it possible to depict all their physical properties. The properties of these elements must be sufficiently general to belong to all the entities subject to modelling. Under these conditions, these modelling elements allow the building of models in which is depicted only the existence of these entities and their relations: these are **global models**.

Finally, the finer modelling of each entity requires the introduction of additional hypotheses specific to it. Generally, these hypotheses and the way they are used to build detailed models belong to often interrelated existing theories: information theory, signal theory, mechanical theory, etc. It is important for the elements introduced in this study to be consistent with these theories. The choice of these elements is delicate: if too general, they would limit our possibilities to the mere construction of models low in properties, i.e. naive and of no interest; if too detailed, they would also limit our possibilities to certain technical areas too restrictive to use. Between these two extremes, the reasonable choice is difficult to establish. This note is thus only a prologue to future developments.

This part is independent of any computer-readable language used for system descriptions. Yet, it supports the formal semantics of such language.

d) Brief introduction to main notions

The word "system" is frequently used in this note. Its prior definition would be fitting here. However, as the entire note is devoted to the mathematical modelling of systems, this prerequisite will be limited, in the following paragraph, to a statement of simple facts of current observation.

Physical systems are made up of entities working interactively in order to achieve a common technical or scientific goal. Each of these entities is a **device**. The device perform processing on other entities; each of these other entities is a **produce**. Devices and produces are the **components** of the system. Exchanges of produces between devices are indispensable for the existence of interactions between devices. We shall assume for the moment that an entity can be seen as a device or as a produce.

e) Brief introduction to method

The modelling of a physical system and, step by step, of each entity composing it generally includes two complementary parts:

- The inventory of the entities making up the system and their relations. This is the **morphology** of the system.
- The inventory of rules governing interactions between entities. This is the **physiology** of the system.

Analysis determines these top-down inventories; it is recursive: each morphology designates entities; each of these entities is also a system amenable to study and so on. The stopping of the study on an entity is imposed either by a decision on the part of the analyst or by a lack of knowledge.

The reverse construction consisting in assembling entities and governing their interactions to form from bottom to up a new entity is a **synthesis.** Generally, the modelling of a system is a succession of analyses and syntheses.

An entity whose morphology and physiology are determined is a **procedure**. A procedure is a model since it is made up of two descriptions: a morphology and a physiology. It is a global and predictive model of a physical system, of a device or a produce[22] concerning **all** their possible occurrences.

22 The systems, machines and produces are entities of the physical world, the process is the general model thereof.

The concrete or experimental study of a physical system is a succession of observations of this system or of certain entities of which it is composed during operation. We shall call **process** the description of an operating case of a procedure. It is a special model of a physical system or device or produce[23] concerning only **one** of its possible occurrences.

f) Summary of approach

Chapter 2: **Primitive elements**; introduces the theoretical elements necessary in mathematics relative to a process and then a procedure.

Chapter 3: **Recursive elements**; constructs on the preceding elements the mathematical models of the simple components; it then gives the basic rules for construction by successive analyses or syntheses the mathematical models of complex components.

2 PRIMITIVE ELEMENTS

2.1 Method and constructive principles

These elements have to do with whatever is immediately perceptible in a system: observable facts or those that intuition or accumulated experience suggest as observable.

The process describes a real or imagined phenomenon but only in time and space: it involves the functioning of the system or of an entity composing this system; the collection of all possible processes constitutes the fundamental space associated with the entity concerned.

General laws exist for this collection of processes; translated mathematically, they allow the structuring of the fundamental space and its transformation into a new space called a substrate. This type of space is the basis for all the modelling elements developed thereafter.

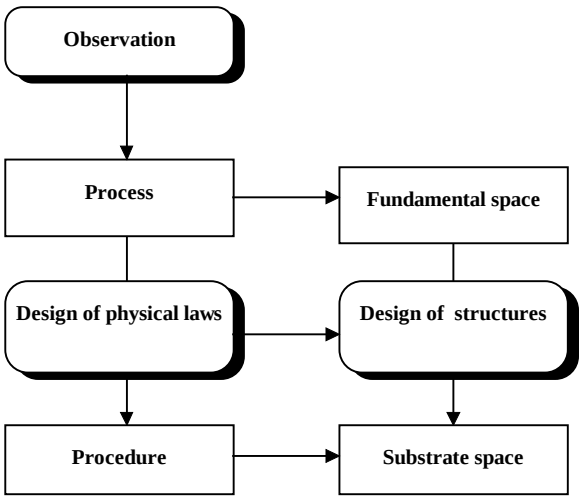


Figure 13: Production of primitive elements

2.2 Fundamental space

2.2.1 General

As we are concerned with observable phenomena, intuition immediately suggests that any process should be expressed by a function allowing space as a variable. Modelling experience however warns us about the known drawbacks of this representation method: there are techniques, and in particular signal processing, for which the functional representation does not readily account for certain physical properties such as, for example, discontinuities[24].

The proposed approach thus avoids having to set *a priori* a precise and definitive mathematical form for the process model. We study the fundamental space without considering any particular form for this

23 Any operating recurrence of a system, a machine or a produce is a phenomenon of the physical world; the process is its particular model.

24 The theory of distributions was devised to obviate this kind of difficulty.

process. The properties of the fundamental space result directly or by deduction from hypotheses allowed for the represented entities. These properties then determine structures for the fundamental space.

2.2.2 Properties

Let us consider a physical system S (or a physical entity) with which we associate the following theoretical elements:

a) Intrinsic properties

hyp. 1: There is a G = {g}; each g represents a **process** of S; G[25]is the **fundamental space.**
hyp. 2: There is a particular element $g_0 \in G$ representing the **absence of process**.

- Comments

The data item for the assembly G is silent as regards the mathematical nature of its elements; there is thus no collectivising property of the processes g allowing the construction of G, except for the one of existing. The existence of G is thus allowed intuitively;

The word "functioning" is used in this text in preference to "behaviour" in order to avoid possible confusion, the latter term moreover having many definitions and already being widely used. To be exhaustive, we must thus include, paradoxically, in the meaning of "functioning" the case of total or partial inactivity of S represented by g_0.

There are few intrinsic properties. This situation is normal in the absence of knowledge regarding the processes: knowledge regarding processes ordinarily results from their observation and hence from the existence of physical entities allowing these observations. The following hypotheses concern these new entities.

b) Extrinsic properties

The following modelling elements concern physical entities indispensable for observation:

- observation spaces,
- time which is a part of any observation space,
- binary signals[26] used for observation control.

- <u>Observation spaces</u>

hyp. 3: There is a countable collection of n sets A_i: A_0, A_1, …, A_n where A_i = {ai}; A_i is a **space of values**[27] and ai[28] is a **value**;
hyp. 4: Any observation set A_i must have a "minimal" structure; we shall assume that this minimal structure is a relationship of a total order: ≤; Consequently, there is an element ai_0 such that $ai_0 \leq ai$ for any ai. ai_0 is **minimum**.

def. 1: The scalar product $A = \{a\} = \Pi A_i\ (i>0)$ where a = (a1, …, ai, …, an) is a **reference**.

- <u>Time</u>

hyp. 5: $A_0 = T$ represents **real time**; T is isomorphic to the set of real numbers.
hyp. 6: A closed set $[t_1, t_2]$ is associated with each g: this is the **domain** of g.

---

25 The notion of behaviour will be defined later.
26 Still in anticipation of what follows, let us distinguish:
- bivalents signals having any two states,
- bivalent logical signals having two states: “true ”, “ false ”,
- binary signals having two conventional states: 0,1.

27 The index i marks the rank of each observation space A_i in the collection of observation spaces; used in the notation ai, it indicates that ai belongs to the space A_i and does not mark any position in this set; the index notation is avoided for this reason.
28 Note that, in the notation “ai”, the letter i indicates that the element belongs to the set A_i and not the place of this element in this set.

- Binary signals

hyp. 7: There is a $\Delta = \{\delta[\tau_1, \tau_2]\}$; any $\delta[\tau_1, \tau_2]$ is a model of the state of the **rectangular signal** associated with the closed set $[\tau_1, \tau_2] \subset T$: this is the Heaviside step, ascending at instant τ_1, combined with the reverse step, descending at instant τ_2. T is the domain of $\delta[\tau_1, \tau_2]$.

hyp. 8: There is a $\Delta\varepsilon = \{\delta[\tau]\}$; any $\delta[\tau]$ is a model for the high state of the **pulse signal** associated with the closed set $[\tau \pm \varepsilon / \varepsilon \rightarrow 0] \subset T$; this is Dirac's pulse centred on τ. T is the domain of $\delta[\tau]$.

Anticipating the rest of the elements, hypotheses 7 and 8 above have to do with special processes: binary elementary signals. These are signals having only two possible states, designated conventionally as high and low. Each occurrence of one of these signals is thus the appearance of one or the other of these states. Only the occurrences of high states are used for the moment.

- Comments

Hypotheses 7 and 8 refer to distribution theory. Under these conditions, any rectangular signal $\delta[\tau_1, \tau_2]$ can be derived for any τ of its domain T. In particular, the derivative for the lower bound τ_1 is the occurrence of the pulse $\delta[\tau_1]$ at instant τ_1:

$$\frac{\partial}{\partial \tau} \delta[\tau_1, \tau_2] = \delta[\tau_1], \text{ for } \tau = \tau_1$$

For the upper bound, the derivative is $-\delta[\tau_2]$; at any other instant, the derivative is 0.
It will be reminded that $\delta[\tau_1, \tau_2]$ and $\delta[\tau]$ are not functions and are nevertheless indefinitely differentiable[29].

2.2.3 Algebraic structures

The following hypotheses have to do with the physical properties of the observed system; these hypotheses are represented by algebraic structures.

hyp. 9: There is an application $\bullet\square:\square\Delta_\varepsilon \times G \rightarrow G$. This application is the **sampling** at the instant τ. The image of any g is a **sample**.

hyp. 10: There is an application $\otimes\square:\square\Delta_\varepsilon \times G \rightarrow A$. This application is the **quantification** at the instant τ. The result is the **quantified value** of the sample.

def. 2: For any g, the process $g' = \delta[\tau] \bullet g$ and $g' \in G$ is the **instantaneous sample** of the process g at the instant τ

For any instant τ not belonging to the domain of g, the sample of g at that instant is the sample of g_0:

hyp. 11: for any $\tau \square \notin [t_1, t_2]$, $(\delta[\tau] \bullet g) = (\delta[\tau] \bullet g_0)$

def. 3: for any g, $\delta[\tau] \otimes g \in A$ is the **instantaneous value** of g at the instant τ.

For any instant τ, the instantaneous value of a process is equal to the quantified value of its sample at the same instant:

hyp. 12: $\delta[\tau] \otimes g = \delta[\tau] \otimes (\delta[\tau] \bullet g)$

Example 29: g being any process, $\delta[\tau] \bullet g$ is what a fast shutter enables us to observe; $\delta[\tau] \otimes g$ is what is recorded on a film.

- Comments

If g is a function of the time taking on its values in A, signal theory is applicable directly.

hyp. 13: The **observation** is an application $\bullet\square:\square\Delta_\varepsilon \times G \rightarrow G$; for any $\delta[\tau_1, \tau_2] \in \Delta$ and all $g \in G$ of domain $[t_1, t_2]$, there is $g'' \in G$ such that:

---

29 E. Roubine. *Intrroduction à la théorie de la communication* (Introduction to communication theory). *Tome I. Signaux non aléatoires* (non random signals). *Chap.III. Emploi des distribution* (use of distributions), pp.23/52.

if $\tau \in [\tau_1, \tau_2] \cap [t_1, t_2]$, then $\delta[\tau] \bullet g'' = \delta[\tau] \bullet g$
if $\tau \notin [\tau_1, \tau_2] \cap [t_1, t_2]$, then $\delta[\tau] \bullet g'' = \delta[\tau] \bullet g_0$

The domain of g'' is $[\tau_1, \tau_2] \cap [t_1, t_2]$. This formalism means physically that the rectangular pulse $\delta[\tau_1, \tau_2]$ limits the observation of g to the interval $[\tau_1, \tau_2]$. To simplify the language, we shall also say that either the rectangular pulse breaks up the process g or the process g modulates the rectangular pulse. Note that this formalism is independent of the mathematical nature of g; it is applicable in particular when g is an analytical function in the form of a simple product of a function multiplied by a distribution.

The simultaneous application of hypotheses 12 and 13 leads the following theorem (Appendix, page 61, dem. 2):

th. 1: if $\tau \in [\tau_1, \tau_2] \cap [t_1, t_2]$, then $\delta[\tau] \otimes g'' = \delta[\tau] \otimes g$
else
if $\tau \notin [\tau_1, \tau_2] \cap [t_1, t_2]$, then $\delta[\tau] \otimes g'' = \delta[\tau] \otimes g_0 = a_0$

- Comments

Note should be made of the difference between the two notions previously defined: the simple sample is a process, an element of G, and the instantaneous value is the image of this sample in the observation space A.

Example 30: g represents an electric voltage, $\delta[\tau] \bullet g$ is what is restored by a sampler; $\delta[\tau] \otimes g$ is what is restored by a converter.
Example 31: g represents the changes in the balance of a bank account, $\delta[\tau] \bullet g$ is the balance at a given date and time; $\delta[\tau] \otimes g$ is the message delivered by the visual display of an automatic cash register at the same date and time.

2.2.4 Structures in classes

a) Structure based upon process recursiveness

We are concerned with phenomena which are reproducible in time. The hypothesis used in this paragraph is that any phenomenon in the functioning of a system is reproducible.

Let g and g' be two elements of G; g and g' are equivalent if they represent the same phenomenon, to within a translation in time. Let ρ be the equivalence relation based upon this property; the quotient set, the elements of which are the equivalence classes is noted: $\Gamma = G/\rho$.

hyp. 14: Whatever g, there is, for any value of the real number θ, a translated g' such that:
$(\forall t \in T)$, $\delta[t-\theta] \otimes g' = \delta[t] \otimes g$,

b) Structure based upon the existence of an initial instant

We have associated a closed set $[t_1,t_2] \subset T$ as a domain with any existing element $g \in G$. This notion of domain must be explained. In fact, for the moment, the choice of the domain appears to be arbitrary: whatever $[t_1,t_2]$, it is possible to find $[t'_1,t'_2] \neq [t_1,t_2]$ where $[t_1,t_2] \subset [t'_1,t'_2]$, and then substitute $[t'_1,t'_2]$ for $[t_1,t_2]$ without calling into question the stated hypotheses. $[t'_1,t'_2]$ is thus also a domain. To clear this indetermination, we express the intuitive fact that after t_1 "there is something happening" and that, after t_2 "nothing is happening anymore".

Let us consider the subset T' of the instants t' of $[t_1,t_2]$ such that for any t', $\delta[t'] \bullet g = \delta[t'] \bullet g_0$. Each t thus defined is enclosed in at least one open set whose intersection with $[t_1,t_2]$ does not contain any other element of T'. Intuitively, the instants such as t' are "isolated" within the domain of the process:

hyp. 15: There is at least one open set $O_{t'}$ containing t' such that for any $t \in O_{t'} \cap [t_1,t_2]$ and $t \neq t'$,
$\delta[t] \bullet g \neq g_0$

- Comments

t_1 and t_2, the left and right borders of the domain can be elements of T'.

Example 32: If the process is a variation in electric voltage u(), the hypothesis means that u(t1)=0 but that immediately after and for a non-zero duration (t)≠0. The hypothesis brings mathematical rigour to physical fuzziness. Any computer specialist having used industrial instrumentation knows that, beyond a certain accuracy threshold, it is difficult to locate the

initial instant of the occurrence of an electric signal. This imprecision was one of the main causes for the abandonment of the “single-slope” analogue converter in benefit of the "dual-slope" converter eliminating this drawback.

Let g and g’ be two elements of G having respectively $[\tau_1,\tau_2]$ et $[\tau'_1,\tau'_2]$ as domains; g and g’ are equivalents if and only if their respective domains have the same initial instant which is their common lower bound: $\tau_1 = \tau'_1 = \tau$; Let us designate as ρ’ the equivalence relation based upon this property; the quotient set is noted as: $\Gamma' = F/\rho'$.

c) Properties common to the two structures

Hereafter we shall use the notation $\Gamma' = \{\gamma'_t\}$; intuitively γ'_t is the class of all g having the same « starting time » t ; γ'_0 is in particular the element of Γ’ associated with the instant 0; let us note $\gamma'_0 = \{\mathcal{g}\}$ where $\mathcal{g}$ represents any g having 0 as starting time. By construction also, there is a bijection between any class γ'_t and Γ; let us choose class γ'_0; any $\mathcal{g}$ represents a class of Γ. From these properties, there is obvious theorem.

th. 2: Any t defines an operator Θ_t in Γ’ (Annexe, Page 61, dem. **1**): $\Theta_t(\gamma'_\tau) = \gamma'_{t-\tau}$.

2.3 Substrate space

2.3.1 Purpose

A **process** is a modelling element; it defines a structured collection of processes defined in time except for a translation; the study of the general properties of this element is based upon the definition of the **substrate space**.

2.3.2 Definitions

def. 4: Any class, element of Γ’, constitutes a **substrate element**.
def. 5: Conventionally, we choose the class associated with the instant 0. This class is the **substrate space**.

In the rest of the document we are using the notations:

$\{\mathcal{g}\} = \mathcal{G} = \gamma'_0 \in \Gamma'$.

Class $\mathcal{G}=\gamma'_0$ is included in G, and the hypothesis relative to the observation (hyp. 13) is applicable; it implies the existence of an application o: $\Delta\times\mathcal{G} \rightarrow \mathcal{G}$.

For any $\delta[\tau_1, \tau_2] \in \Delta$ and any $\mathcal{g} \in \mathcal{G}$ having a domain $[0, t_2]$, there is g’∈G such that:

if $\tau\in[\tau_1, \tau_2] \cap [0, t_2]$ then $\delta[\tau-\tau_1]\otimes g' = \delta[\tau] \otimes(\delta[\tau_1, \tau_2] \bullet \mathcal{g}) = \delta[\tau]\otimes\mathcal{g}$
if $\tau\notin[\tau_1, \tau_2]\cap [0, t_2]$ then $\delta[\tau-\tau_1] \otimes g' = \delta[\tau] \otimes(\delta[\tau_1, \tau_2] \bullet \mathcal{g}) = a_0$

The left bound of the domain of g’ is τ_1; then g’ is an element of the class γ'_{τ_1}; τ_1 defines an operator $\Theta\tau_1$ in Γ’ such that: $\Theta\tau_1(\gamma'_{\tau_1}) = \gamma'_0 = \mathcal{G}$. Consequently, $\mathcal{g}$’ has an image g’ in $\mathcal{G}$.

th. 3: If $\mathcal{g}$ is a process, then $\mathcal{g}$’ its observation,

- obtained during an interval $[\tau_1, \tau_2]$ and,
- translated to the origin, is also an element of $\mathcal{G}$:

$\mathcal{g}'=\delta[\tau_1, \tau_2]$ o $\mathcal{g}\in\mathcal{G}$

2.3.3 Properties of substrate space

The following hypotheses and definitions are given for the substrate space $\mathcal{G}$ conventionally associated with the instant 0. They are immediately applicable to any translated substrate space of the preceding and associated with any other instant t. Let us consider: P($\mathcal{G}$), the set of subsets of $\mathcal{G}$.

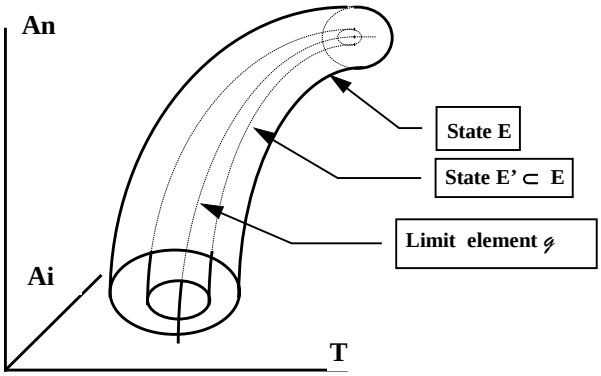


Figure 14: Interpretation of notion of state

a) Family of states

def. 6: A **substrate state** g is a part $X \in \mathcal{P}(\mathcal{G})$ containing g.

def. 7: $\mathcal{P}_g(\mathcal{G})$ is the **family** of the **substrate states** associated with g: for any $X \in \mathcal{P}_g(\mathcal{G})$, $g \in X$,

The following two hypotheses enable us to assign a topological structure to the substrate space.

- Comments

This topological structure is not unique: other structures depending on the properties of the elements may be introduced later. Depending on the particular physical properties of the represented entity, they are not examined in this paragraph.

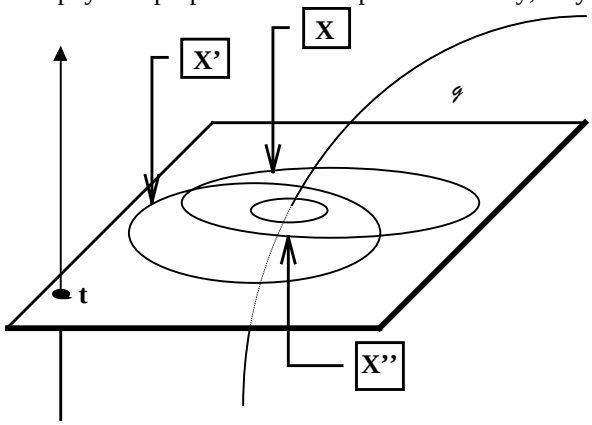


Figure 15: State families - convergence

b) Convergence

hyp. 16: For any X and $X' \in \mathcal{P}_g(\mathcal{G})$, there is an X'' such that:

$X'' \subset X \cap X'$ and $X'' \in \mathcal{P}_g(\mathcal{G})$.

The intersection of two substrate states, elements of a given family, contain another substrate state belonging to this family.

- Comments

This hypothesis again states with rigour an idealised physical fact.

Example 33: Let us suppose that the same process g is observed with several carefully adjusted and calibrated observation systems all giving an image of g in A. For example, the preceding electric voltage u() measured with several voltmeters. Each image is a bundle in A×T. The above hypothesis means that all these bundles have a non-empty intersection; this intersection can contain another bundle possibly produced by another observation system of better precision than the preceding ones; the perfect image of g, indeterminable with full certainty, is somewhere within this latter bundle.

c) Neighbourhood

hyp. 17: For any $X \in \mathcal{P}_g(\mathcal{G})$ and for any $h \in X$, there is $Y \subset X$ with $g \in Y$ and $Y \in \mathcal{P}_h(\mathcal{G})$

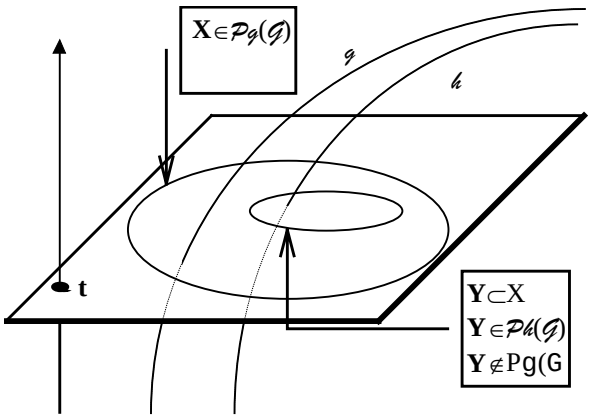


Figure 16: Family of states - continuity

The collection of all the families defines a topology for $\mathcal{G}$.

d) Primitive element definition limits

The belonging of $\{g\}$ to $\mathcal{P}g(\mathcal{G})$ will be discussed later; g is a **limit element**; $\{g\}$ is a **limit state**.. This is a hypothesis which, depending on whether it is accepted or rejected, modifies the topological properties of the substrate space. Intuitively, the hypothesis will be admitted for the construction of models of real phenomena perceived through observation systems which yield only imperfect images. Rejection will be used for the construction of models of phenomena which are well controlled in their occurrences, such as executions of numerical models; in this case, the observation system could communicate an exact trace of the occurrence.

3 RECURSIVE ELEMENTS

3.1 Method and constructional principles

The theoretical elements examined earlier are used in this chapter for the construction of system models. Two approaches are possible.

A) Analysis

The analysis of an entity shows new entities. We first of all examine how theoretical elements are used to express the models of these entities which are:

- produces,
- devices.

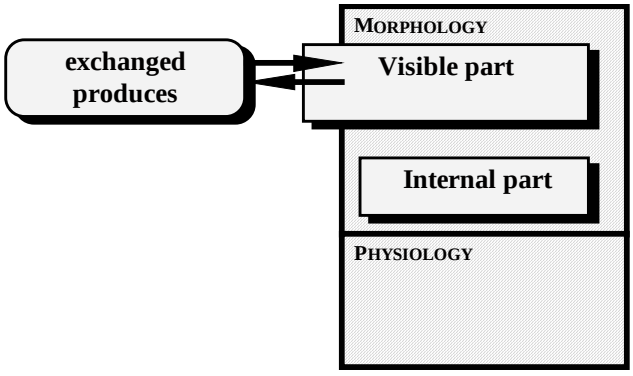


Figure 17: Analysis of an entity

The method of expressing the assembly of these entities is assimilated with synthesis.

B) Synthesis

Given a collection of entities, synthesis consists in assembling these entities, possibly modified, and explaining their interactions. We examine how theoretical elements are used to form the morphology and then the physiology of the new entity thus constructed.

These approaches are covered in detail in the following paragraphs.

3.2 Elements for morphological design

3.2.1 General

There are two possibilities for observing the functioning of a physical entity:

- There is an observation system allowing a correspondence between any instant and an instantaneous value in the observation space. The entity is then considered to be a produce
- There is an observation system enabling the preceding procedure for produces placed at the input of the physical entity as well as for produces coming from its outputs. The entity is then considered to be a device.

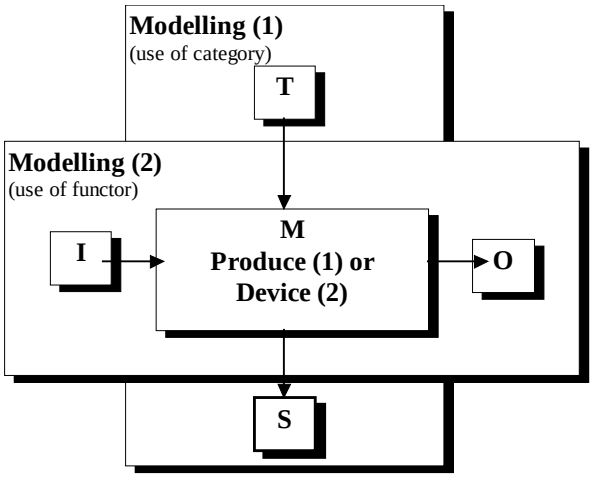


Figure 18: Two possibilities for observing and two objectives of modelling

A procedure is a model describing the general functioning rules of a produce or a device; a procedure instantiation is a process. The notion of component having different possible states is almost universal. It appears at different moments of an analysis: a variable, a task, a program, the system itself are all elements having this common physical property, independent of any other property. These two possibilities for observing imply two ways for modelling. The theory of categories brings the required formal means to the representation of these basic properties

3.2.2 Produce model

a) Definitions

Given a family of processes, only some open sets X (see Fig. 14, Page 30) are of interest for the study of this family. Consequently, we are assuming here that there is at least one condition Φ available for defining a particular subset: $S = \Phi(\mathcal{P}(\mathcal{G})) \subset \mathcal{P}(\mathcal{G})$.

def. 8: Each element of $S = \Phi(\mathcal{P}(\mathcal{G})) \subset \mathcal{P}(\mathcal{G})$ is a **procedure state**, chosen according to the condition:
Φ: $S = \{ X / X \in \mathcal{P}(\mathcal{G}) \text{ and } \Phi(X) = \text{true} \}$

def. 9: A **limit state** is an open set in which each element differs very little from a particular element g called the **limit element** or is reduced to this element.

Let us note that it is experimentally impossible to distinguish a limit element. Any means of observation gives only knowledge of a limit state, the scattering of this limit state depending on the accuracy of the observation means used. However, at this level of analysis, it is impossible, as we have already noted, to affirm or invalidate the belonging of g to S.

The theory of categories is used with the remark that there are relations between the states assigned to a procedure. Let us now consider a set of such states associated with a procedure. We assume that:

hyp. 18: For any pair (X_α, X_β) of states there is a set $M_{\alpha\beta}$ of relations from X_α to X_β; if $_\alpha = _\beta$, this set is reduced to the singleton formed by the identical relation; each relation is called a **morphism**; For any (X'_α, X'_β), $M_{\alpha\beta} = M'_{\alpha\beta}$ implies that $X'_\alpha = X_\alpha$ and $X'_\beta = X_\beta$. In this study, any $M_{\alpha\beta}$ is, either **empty**, or a singleton having the element $m_{\alpha\beta}$.

hyp. 19: For any triplet $(X_\alpha, X_\beta, X_\chi)$ such as $M_{\alpha\beta} \neq \varnothing$ and $M_{\beta\chi} \neq \varnothing$ and $M_{\alpha\chi} \neq \varnothing$, there is a relation:
$m_{\alpha\beta} \times m_{\beta\chi} \rightarrow m_{\alpha\chi}$

hyp. 20: The relation defined above is associative:
$(m_{\alpha\beta} \times m_{\beta\chi}) \times m_{\chi\delta} = m_{\alpha\chi} \times m_{\chi\delta} = m_{\alpha\beta} \times (m_{\beta\chi} \times m_{\chi\delta}) = m_{\alpha\beta} \times m_{\beta\delta}$

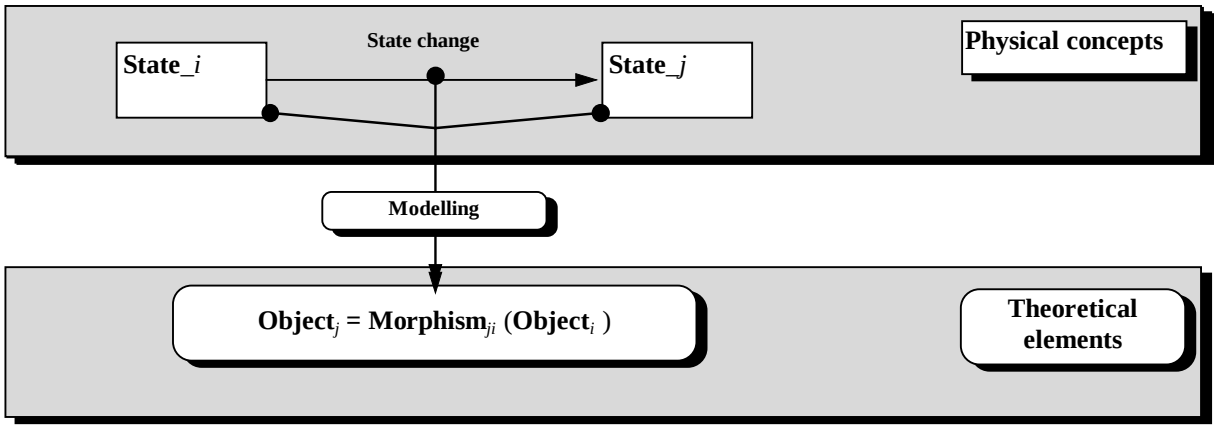


Figure 19: Approximative illustration of the relationships between physical concepts and theoretical elements (morphology)

These hypotheses concerning the properties of procedures are such that the notion of category is utilizable as models for the states of procedures. The states are the objects of the category and the state changes are morphisms (Figure 19); ordinarily, the sets of morphisms in this case of application have at most an element.

b) Discussion on the continuity of states

th. 4: The substrate space has the power of the continuous (Appendix, Page 61, dem. 3.).

The set of states has a power. This power depends on the definition of S. The set of states which is a set of open subsets of the substrate space, may be either continuous or countable.

c) Combination of produces

Let be a functionality F having several inputs I_i, i = 1, …, n. If each I_i is a category, then the model of the compound produce $I_1 \times \ldots I_n$ is a product of categories (see: appendix).

We will use this property in the following section.

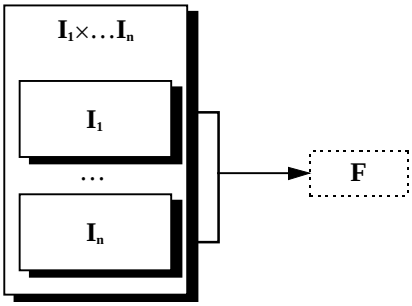


Figure 20: Compound produce

3.2.3 Device model

We shall examine first of all a simple functionality giving a transformation of a produce at the input into an output produce. We represent the input (respectively the output) by an input category (respectively an output category). We then assume that the functionality applies the objects of the input category (respectively the morphisms) in the objects of the output category (respectively the morphisms).

hyp. 21: We assume that the application carried out by the functionality complies with the axioms defining a **fonctor**.

A) Simple one-state device

def. 10: The **simple one-state device** embeds a functionality defining a correspondence between the inputs and the outputs.

The model of each input or of each output is a product of categories. The model of the functionality is a fonctor; therewith the model of the simple one-state device is a fonctor.

b) Compound one-state devices

We examine two simple compositions of simple one-state devices. The result is a device which embeds:

- either a serial assemblage of one state-devices

- either a parallel assemblage of one-state devices.

A collection of functionalities F_1, …, F_n is a serial collection if the input I_i of any F_i ($n \geq i > 1$) is the output O_{i-1} of F_{i-1}. The result is a functionality such that: $F: I_1 \rightarrow O_n$.

def. 11: The **serial compound one-state device** is a device which embeds a serial collection of functionalities.

The model of this device is the fonctor $F: I_1 \rightarrow O_n$.

A collection of functionalities F_1, …, F_n is a parallel collection if any pair (F_i, F_j) of this collection exists without link[30]; The functionality $F: (I_1 \times \ldots I_n) \rightarrow (O_1 \times \ldots O_n)$ represents this collection.

def. 12: The **parallel compound one-state device** is a device which embeds a parallel collection of functionalities.

The model of this device is the fonctor $F: (I_1 \times \ldots I_n) \rightarrow (O_1 \times \ldots O_n)$.

c) Multi-states device

A functionality may be a product having multiple states. This assumption is very usual in data processing technologies. In this case, each state of the functioality may be viewed as a simple functionality. We examine now the features of a device embeding such a functionality F. F may be viewed as a fonctor or as a caregory if we need to take in account its different states. Let be:

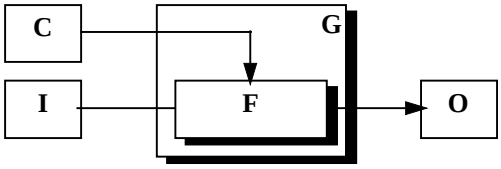


Figure 21: multiple states functionality

- $F=(\{F_i\}, \{f_{ji}\})$, the category, model of the multi-states functionality; each Fi is a fonctor and represents a simple-state functionality.
- $C=(\{C_i\}, \{c_{ji}\})$, a category, model of the input data used in order to control the states of F
- $I=(\{I_a\}, \{m_{ba}\})$, a category, model of the input of the device.
- $O'=(\{O'_{(ia)}\}, \{n'_{(jb)(ia)}\})$, a category, model of the output of the device.

We assume that:

hyp. 22: it exists an one to one application between $\{C_i\}$ and $\{F_i\}$,
hyp. 23: it exists an one to one application between $\{c_{ji}\}$ and $\{f_{ji}\}$,
hyp. 24: O' is a subcategory of a category O which has the following properties;
hyp. 25: For any i and a, $(C_i, I_a) \rightarrow O_{ia} = F_i(I_a)$;
hyp. 26: For any (j,i) and (b,a),

$(c_{ji}, m_{ba}) \rightarrow n_{(ji)(ba)}$

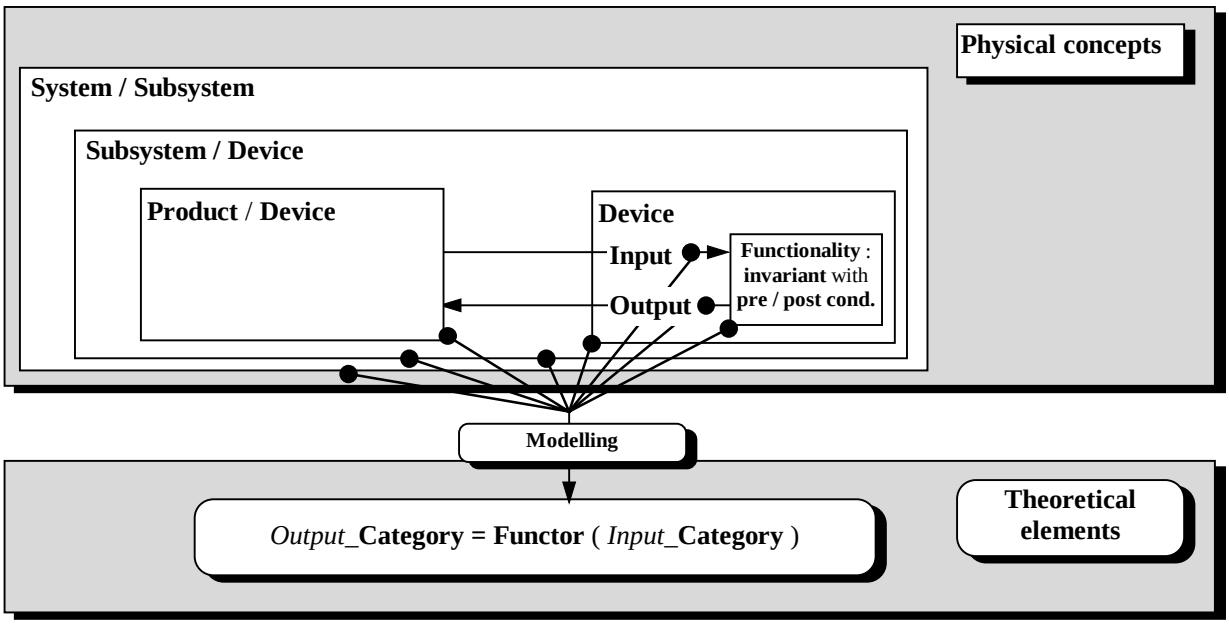
and

$O_{jb} = n_{(ji)(ba)} O_{ia}$
$= (c_{ji}, m_{ba})(F_i(I_a))$
$= (c_{ji}F_i)(m_{ba}I_a)$.

All these hypotheses allow the existence of a fonctor $G: C \times I \rightarrow O'$. This fonctor is the model of the device embeding the multi-state functionality. Globaly, this device is a one-state functionality having the product $C \times I$ as input and O' as output.

Let us note that the last hypothesis (26) is not a universal property of the product of categories; this hypothesis is restrictive for modelling any component of a system. At this point we can imagine, as we like, some multi-state component losing the properties of fonctor. Then, the result is not a device; all the components which embed recursively this item are also losing the properties of a fonctor and are not devices. Whe have not been studying at this time the effects of such an arrangement; these effects transform certainly the conditions of transportability and of interoperability and complicate their specifications. For that reason, the existence of softwares available for checking in a system description the different possible lacks of fonctor properties.

30 In practice, the check of this condition is often uneasy.



Figuree 22: Approximative illustration of the relationships between physical concepts and theoretical elements (morphology)

D) Special types of devices: The receiver, transmitter, connector

The model of the receiver, transmitter or connector is a procedure. This procedure is also a fonctor; it specifies the criteria for the recognition of categories or generation of categories at the border of a procedure. These are fonctors to the extent that generally they impose transformations on the categories mentioned in the section on the borders of the procedure.

Example 34: A receiver receives a pulse train but, in each pulse, recognises only the rising edge. A transmitter transforms a pulse train generated inside the procedure into a continuous signal whose frequency is that of the occurrences of the distinct pulses.

3.2.4 Representation of fundamental properties

The category serves as a model for the produces; the fonctor serves as a model for the devices.

A) Existence

A fonctor which establishes a relation between a product of categories and the category (false, true).

B) Temporal domain

For any category C and for any category limited to a single object $\theta \in T$, an operator defines a transformation: $C \rightarrow C' \times \theta$

Example 35: A sampler defines a category C' having the same properties as C over the interval θ.

An operator is fundamentally the base of a rule of meta production indispensable in any system description language:

```
operator: morphology →□morphology
```

In a description, the operator allows a simple definition of a produce based on a type given by modification of the domain.

C) Linkage

A fonctor intended to identify the links between I/Os of different procedures.

Example 36: The "identical" fonctor used to specify that the inputs of one procedure are the outputs of another procedure and are identical.

3.3 Elements for physiological design

3.3.1 General

The analysis of a system leads to the identification of its components. These components - devices and produces - form the morphology of the system; this morphology evolves as the analysis progresses. The notion of category serves as a model for the produces. Analysis determines the recognition of devices and produces meeting the general characteristics of Figure 17. Modelling should make it possible, for each produce, to specify its different possible states. It must allow, for each device, an account:

- of the produces exchanged and of the morphological specifications imposed by the device on these produces, possibly specifications relative to processing performed on these produces,
- of its location in time.

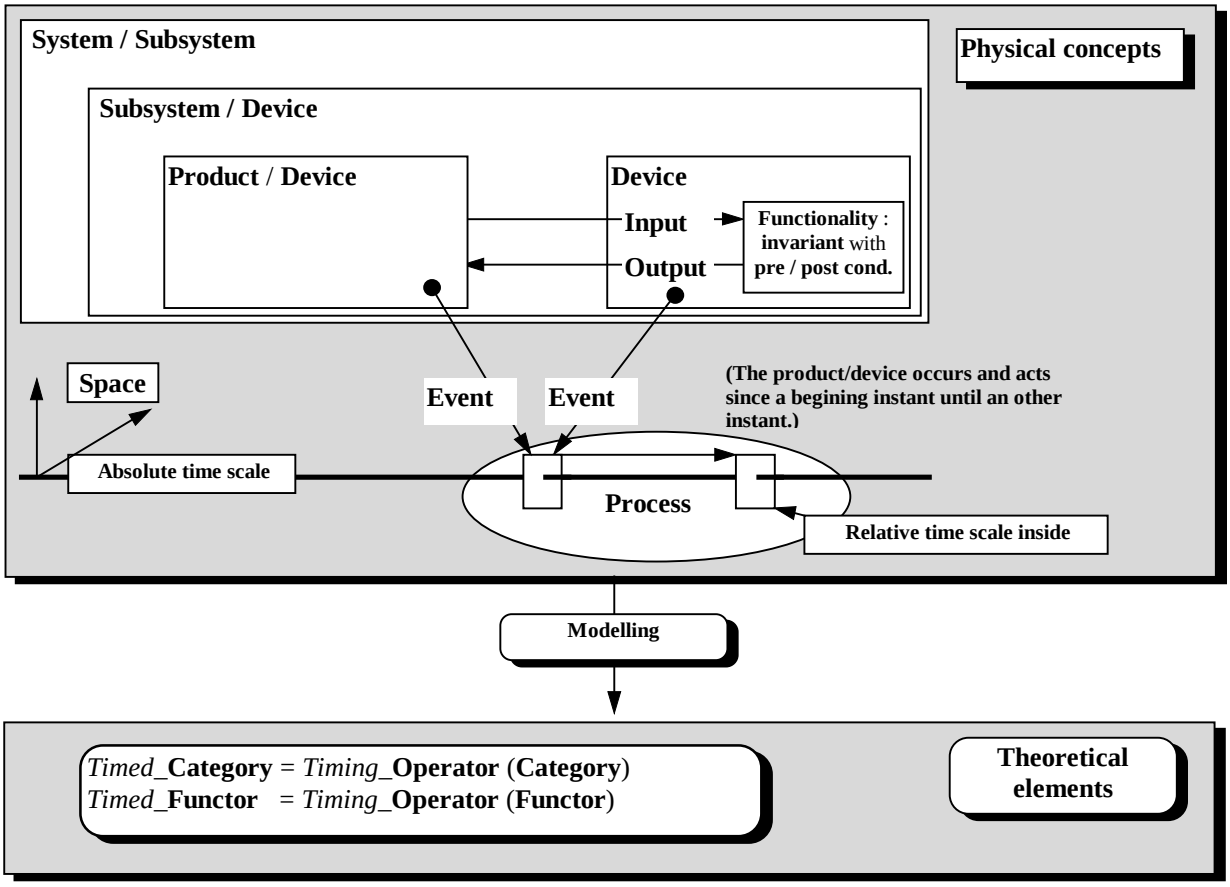


Figure 23: Approximative illustration of the relationships between physical concepts and theoretical elements (physiology)

3.3.2 Command or Timing-operator

A command is a fundamental element of the meta-production rule:

`command: morphology×time→physiology`

This meta-production rule has two physical acceptations described by the following mathematical models. The behaviour of a system is the definition of a temporal location of each active part of the system. Any fonctor used in the definition of a behaviour transforms a given category C (see the Section « Produce model », Page 32) into a other category C' taking in account its location in time.This kind of fonctor is a **command** or a **timing-operator**. For any category C, two type of command are available.

A) On/off command

Let be $\mathcal{P}(T)$ the set of the open subsets of T: $\mathcal{P}(T) = \{]\tau_i,\tau_j[\}$ and Θ the n-uple such that:
$\Theta = (]\tau_{i1},\tau_{j1}[, \ldots,]\tau_{in},\tau_{jn}[) \in \mathcal{P}^n(T)$
with the condition: for any α and β, $\alpha \neq \beta$, $1\leq\alpha\leq n$, $1\leq\beta\leq n$, $]\tau_{i\alpha},\tau_{j\alpha}[\cap]\tau_{i\beta},\tau_{j\beta}[= \varnothing$.

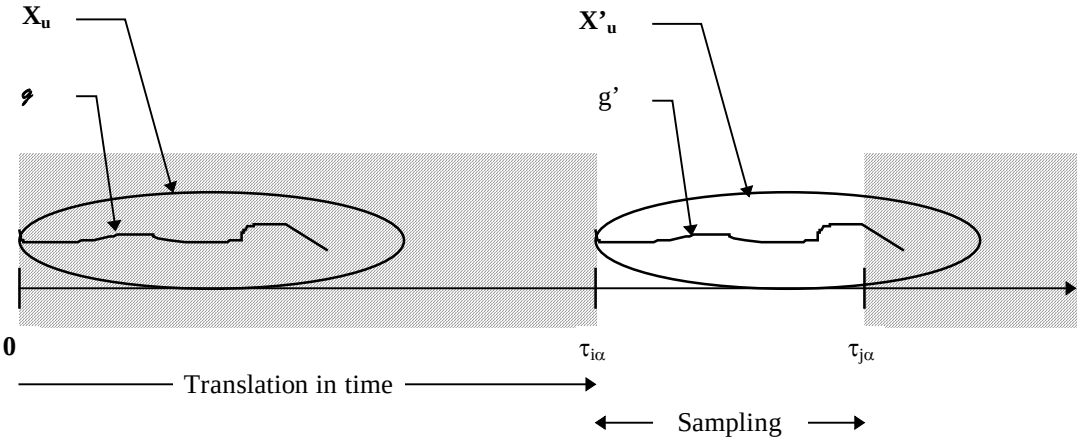


Figure 24 : ON/OFF command

Θ is a category; it models a sequence of disjointed windows along the time-scale; it has n objects: the intervals $]\tau_{i\alpha},\tau_{j\alpha}[$ $(1\leq\alpha\leq n)$; these objects are ordered and this relationship defines a morphism for each pair of objects.

For any category C which models a static object, and for any category Θ which is the support of the command, an On/Off command defines the produce: $C' = C\times\Theta$. Each object of this category is a pair $X'_u = (X_u,]\tau_{i\alpha},\tau_{j\alpha}[)$; this pair means that :

- each element g' of X'_u is the translation in time, from 0 to $\tau_{i\alpha}$ of an object g of X_u,
- the left bound of the domain of each g, element of X_u is $\tau_{i\alpha}$, and that each g is sampled after this translation in time.

B) Trigger or pulse-command

Let be $\mathcal{P}'(T)$ the set of the open subsets of T: $\mathcal{P}(T) = \{]\tau_i\pm\varepsilon[\}$ and Θ' the n-uple such that:
$\Theta' = (]\tau_1\pm\varepsilon[, \ldots,]\tau_n\pm\varepsilon[) \in \mathcal{P}^n(T)$ with the condition: for any i and j, $i \neq j$, $1\leq i\leq n$, $1\leq j\leq n$, $\tau_i \neq \tau_j$.

Θ' is a category; it has n objects: the intervals $]\tau_i\pm\varepsilon[$ $(1\leq i\leq n)$; these objects are ordered and this relationship defines a morphism for each pair of objects.

The trigger transforms C into $C' = C\times\Theta'$. Each object of this category is a pair $X'_u = (X_u,]\tau_i\pm\varepsilon[)$; this pair means that:

- each element g' of X'_u is the translation in time, from 0 to τ_i of an untimed element g of X_u,

- the left bound of the domain of each g', element of X_u is τ_i, and g' is equal to g after this translation without sampling.

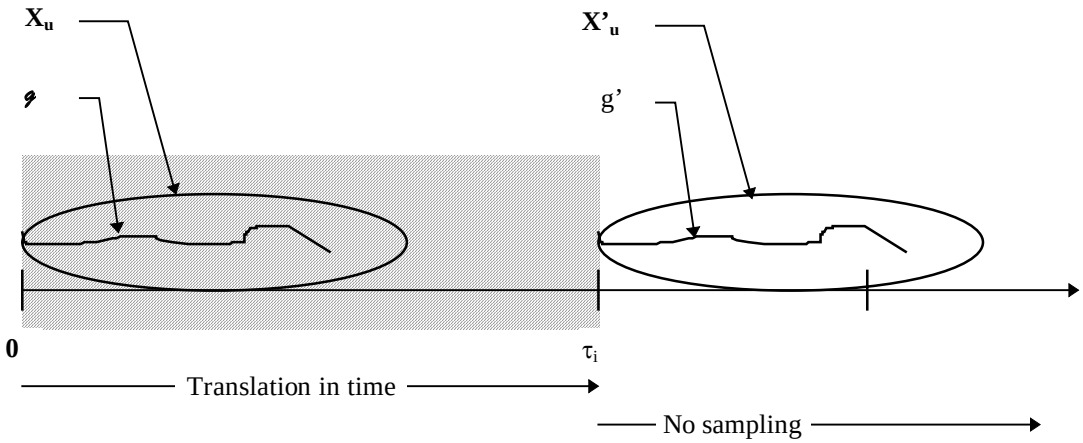


Figure 25 : Trigger command

3.3.3 Temporal condition

Let us note that any command can be combined with a condition using T.

Example 37: Activate a procedure at an instant t, generate a pulse at an instant t.

4 INTEROPERABILITY

The models of entities are assemblies of the previously described theoretical elements. The properties of these assemblies result from the hypotheses and definitions proposed earlier. We shall be examining more particularly in this chapter the ability of the assembled entities to function interactively: they indicate the physical consistency of the system.

The mathematical model enables this property to be studied by offering the means of distinguishing its different aspects. We shall thus be examining in particular the questions of interoperability and portability, linking them with the examination of the consistency of the mathematical expressions of the entities concerned.

As we stated earlier, interoperability is the ability of a procedure **F'** to accept any produce sent by another procedure **F**. Let **O** be the category sent by **F,** and **I'** the category that can be received by **F'**. The consistency between **O** and **I'** exhibits different aspects. We shall attempt now to classify, to define some consistency criteria between procedures acting interactively.

4.1 Morphological consistency

This notion is intrinsic to the pair (**F**, **F'**). Its modelling uses the mathematical concept of universal element (Appendix, Page 60). Let **I'°** be a subcategory of **I';** we propose:

def. 13: **Normal consistency**. **F** and **F'** are normally consistent for **O** and **I'** if there is an abstract fonctor **Γ**: **O** → **I'**° having at least one universal element . **Γ** defines a **standard interface**.

def. 14: **Adapted consistency** **F** and **F'** are adapted for **O** and **I'** if there is a fonctor **Γ***: **I*** → **O***° having at least one universal element such that (**F**, **Γ***) and (**Γ***,**F'**) are normally consistent. **Γ*** defines an **adapted interface.**

4.2 Physiological consistency

This notion is concerned fundamentally with real time. It is intrinsic to the pair (**F**, **F'**) but introduces time. Let us consider a closed set $[\tau_1, \tau_2]$; the set of open subsets included in $[\tau_1, \tau_2]$ forms a category. Since real time is physically invariant in our applications, every fonctor **Γ** transforms this category in itself and conserves its morphisms. There is a universal element in this case which is $[\tau_1, \tau_2]$. For any τ, let us consider the instantaneous open set $[\tau \pm \varepsilon]$ and f: $[\tau_1, \tau_2] \rightarrow [\tau \pm \varepsilon]$; for any $[\tau' \pm \varepsilon]$ there is a unique f such that: f ($[\tau_1, \tau_2]$) = $[\tau' \pm \varepsilon]$. If **F** and **F'** are naturally consistent, f is such that $[\tau \pm \varepsilon] = [\tau' \pm \varepsilon]$; if **F** and **F'** are not naturally consistent, f depends on the artificial interface.

It is now possible that **F** and **F'** are active in the different intervals: $[\tau_1, \tau_2]$, $[\tau'_1, \tau'_2]$.

def. 15: Physiological interoperability exists if:
- **F** and **F'** are consistent,
- $[\tau' \pm \varepsilon] \subset [\tau'_1, \tau'_2]$.

5 PORTABILITY

The notion of portability can be expressed in the theory of categories. In the following proposition we use the mathematical notion of natural transformation (Appendix, Page 59). An even superficial examination of portability shows the complexity of this notion, owing to the existence of what we conventionally call the service layers. The following proposition is thus only a theoretical element applicable to a physical system element; this theoretical element is thus available for the construction of a complete model specific to each case examined; it is not a universal model.

Let us consider two platforms "m" and "n"; we shall assume in our proposal that the binary element configurations used to store information have the same number of elements in m and in n and that m and n have the same number of such configurations[31]. Let us consider an application (computer) **S** and its implementation $\mathbf{S_m}$ in the platform m (resp. $\mathbf{S_n}$ in n). $\mathbf{S_m}$ and $\mathbf{S_n}$ are fonctors. The common input of these implementations is a category **C** situated at the level of the user (for example, a record on a user's data disk). We thus assume here that $\mathbf{S_m}$ and $\mathbf{S_n}$ include the components allowing the implementation of C in the different platforms. The outputs $\mathbf{S_m}(\mathbf{C})$ and $\mathbf{S_n}(\mathbf{C})$ are respectively in accordance with the specifications of platforms which support the applications. If C' is the category formed by the ordered set of all the possible combinations of valorised binary element configurations of m or n, $\mathbf{S_m}(\mathbf{C})$ and $\mathbf{S_n}(\mathbf{C})$ are subcategories generally different from **C'**.

Using the notion de natural transformation, we express the portability of S from m to n by the existence of a relation between the installed applications $\mathbf{S_m}$ and $\mathbf{S_n}$, leading to a relation between the outputs $\mathbf{S_m}(\mathbf{C})$ and $\mathbf{S_n}(\mathbf{C})$ for any input **C**.

31 This proposition is a simplified example and not an exhaustive study of the subject.

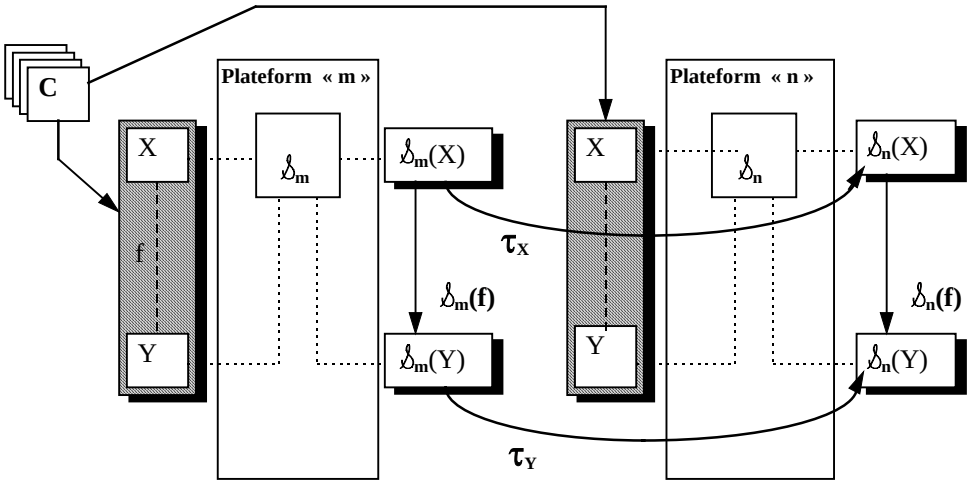
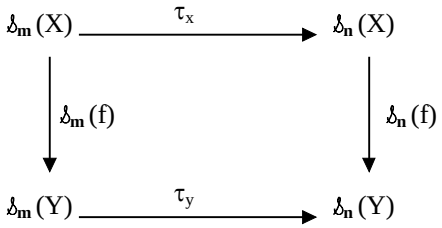


Fig. 26: Portability

$\tau: \mathbf{S_m} \rightarrow \mathbf{S_n}$.



def. 16: The above diagram defines an oriented elementary portability from S_m to S_n (go only).

th. 5: In order for the elementary portability to be reversible between S_m and S_n, it is necessary and sufficient for all the morphisms τ to be invertibles (go and return).

6 CONCLUSION

This part concerned the use of mathematics for system modelling: we have given an overview of some simple physical evidences as we perceive them, and we have consequently proposed their mathematical representations as we imagine them.

In any other usual scientific or technical domain of applied mathematics, anyone, who has been studying about mathematical modelling elements, will have been fascinated by the power of these tools, for discerning and enlightening the properties of the examined physical objects; furthermore these tools avail the growth of design techniques for engineering which gain gradually a consensus from the users and finally lead common practices for technicians who are not mathematicians. An example is given by the finite elements method which was a subject for mathematician at the end of the years fifty and is nowadays an usual tool in all engineering offices and lays the foundation of many civil engineering standards.

The various processes whereby these common practices emerge and evolve are complex. We recognise entirely their end results without explanation. The conclusion is that we must keep in mind the existence of many theoretical researchs undertaken by teams in numerous laboratories and we suggest finally to take them into account in our standardisation activities.We have established the possibility of associating a model with a physical system as well as with each entity recognised in its composition. This model is an assembly of morphological and physiological assertions; such an assembly may comprise several thousand assertions.

The value of mathematics, namely that it produces models which are condensed in terms of expression but rich in terms of meaning, thus appears to be absent from this approach. Reasoning with a model

having such a number of relations is humanly impossible: for example, the simple search for contradictions which could mean potential functioning errors is excluded.

Such formal verifications by reasoning become possible with the assistance of the computer. Classical notations are unfortunately inappropriate. The search for a formal, computer-readable language whose elements and assemblies have the same semantics as the usual mathematical language is thus required. This search can be based upon existing description languages.

It forms the subject of the next part.

PART 4
DESCRIPTION LANGUAGE (IRON FRAME)

Toute interdisciplinarité réelle passe par l'édification d'un langage commun qui puisse exprimer les divers moyens théoriques en usage dans les disciplines les plus variées.
René Thom[32]*. Logos Phoenix, Modèles mathématiques de la morphogénèse, Bourgoi, Paris, 1980, p.295.*

1 GENERAL

During the eighteenth century the existence of such an universal scientific language was soon supposed by Leibnitz. Unfortunately, Leibnitz papers tell us almost nothing about that. It is one of the most famous enigma of mathematics history. We can say nowadays that the binary formalism discovered or re-discovered by Leibnitz is perhaps the basis of this hypothetical language. The conclusion to which we are coming since the beginning of computers era is that in the present state of the technology, binary language is obviously at the lowest level, the only universal language with which any problem of calculation must be expressed and solved; nevertheless, in practice, it is easily readable and understandable only by computer and never by human mind.

While we have this language in view, and while we must leave for the present the study of the conditions under which it may be designed, we should just consider for a moment what we mean by the concept of global description.

At the present day no known high level language, that means readable by computer and undersantable by human mind, possesses without restriction the property of universality. The theory of category underlies the precedent definitions. The theory of category is an algebra; thus it is a logical theory and the axioms, definitions and theorems of fundamental logic are available and therefore must be applied.

A description language is one of the most important items of a CSMF. Consequently, the CSMF standard must provide for these languages:

- A general primary formalism allowing an unitary basic structure.
- General specifications for their capabilities.
- Criteria for checking the compliance of the features of a particular language with the preceding specifications.

The purpose of this chapter is to study the problems concerning the description languages. The first paragraph proposes a simple primary formalism for expressing, at the lowest level, the concepts of CSMF standard and their possible assemblages. The second paragraph examines how that language may be used as a meta-language for producing description languages. In compliance with the primary formalism, the third paragraph gives in natural language some general specifications for the description languages.

32 Fields Medal, 1958.

2 PRIMARY FORMALISM (STONE FRAME)

2.1 Objectives

The purpose of this paragraph is the definition of a primary formalism without ambiguity. This formalism must be viewed firstly as a language a the lowest level ; consequently, it must have only a reduced set of notions and a reduced set of rules. Then, this primitive formalism will be used uniquely for expressing strictly the physical concepts defined informally in the preceding chapter and the relationships between these concepts. It must be consistent with the mathematical elements defined in the next chapter.

This low level formalism cannot be used for the description of systems. Hereafter, the notions and rules of this low level formalism will be used, for that purpose, as a frame for defining description languages. Then, it may be viewed also as a general and formal specification of description languages.

2.2 General features

This formalism is a direct application of set theory. It uses the habitual notations of this theory:

- braces enclose a repeated item; the item may appear zero or more times; each occurrence of the repeated item is distinct from the other occurrences.
- $\cap$, $\cup$, $\times$ represent respectively intersection, reunion and scalar product; Π represents a scalar product having an undetermined finite list of terms.
- $\varnothing$ is the empty set.
- The chain,
  « `<expression_1>: <expression_2>` $\rightarrow$ `<expression- 3>` »
  means:
  `<expression_1>` represents an application from
  `<expression_2>`
  into
  `<expression_3>`.
- In addition in any definition, the symbol « `::=` » means « `is` », and the symbol « `--` » precedes a informal comment enclosed inside the formal text.

From a mathematical point of view, {item} represents the classical set of element « item » which is a list of sentences in the spirit of this chapter.

2.3 Primary notions

The terms expressed below are the primitive terms of the formalism. They express both:

- either a concept;
- either a set of sentences associated with the concept, whose elements complies with the concept.

For each word, the existence of the expressed concept is an axiom of the primary formalism. Likewise the existence of the concept, which is expressed by the word « point », is an axiom of geometry. Besides, each word, from (1) to (8), represents any possible element of a set, and likewise the word « point » represents the elements of a geometrical space.

```
(1) morphological_item::=
```

```
(2) morphological_typology _template|
(3) morphological_genealogy _ancestor|
(4) elementary_object

(5) physiological_item::=
(6) physiological_typology _template|
(7) physiological_genealogy _ancestor|
(8) action_specification
```

2.4 Compound notions

```
(9) morphology::= {morphological_item}|
    morphology∩morphology |
    morphology∪morphology |
    morphology×morphology
    -- Figure 28, Page 46

(10)physiology::= {physiological_item}|
    physiology∩physiology|
    physiology∪physiology|
    physiology×physiology
    -- Figure 29, Page 46

(11)visible_morphology::= morphology
(12)internal_morphology::= morphology

(13)normal_physiology::= physiology
(14)exceptional_physiology::=
    physiology
(15)object::= morphological_item |
    visible_morphology ∪internal_morphology ∪normal_physiology
    ∪exceptional_physiology
```

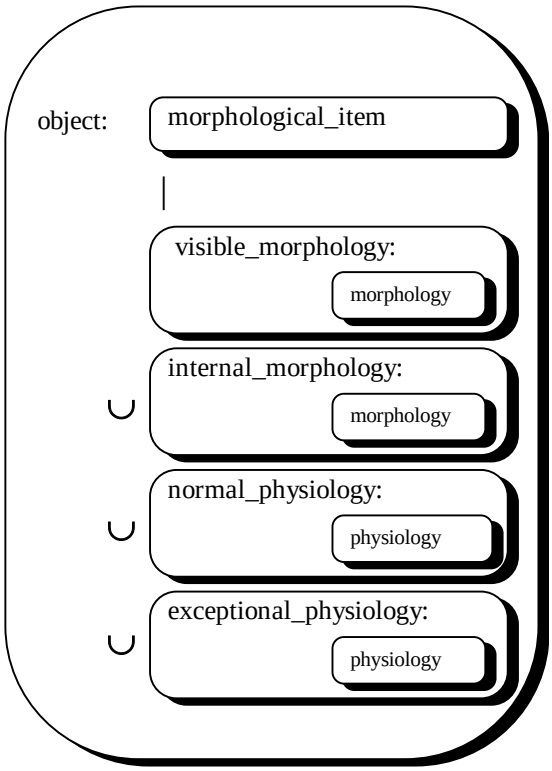


Figure 27: Object

2.5 Primary rules

```
(16)time::= object
(17)morphological_production_rule::= morphology | operator: morphology→morphology
(18)physiological_production_rule::= command: morphology×time→physiology
```

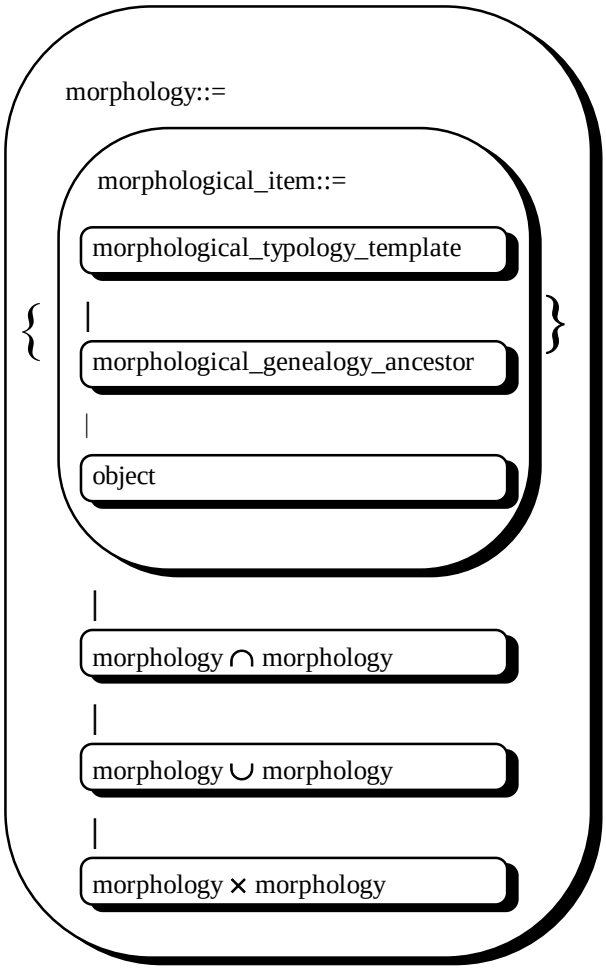


Figure 28: Map of morphological concepts

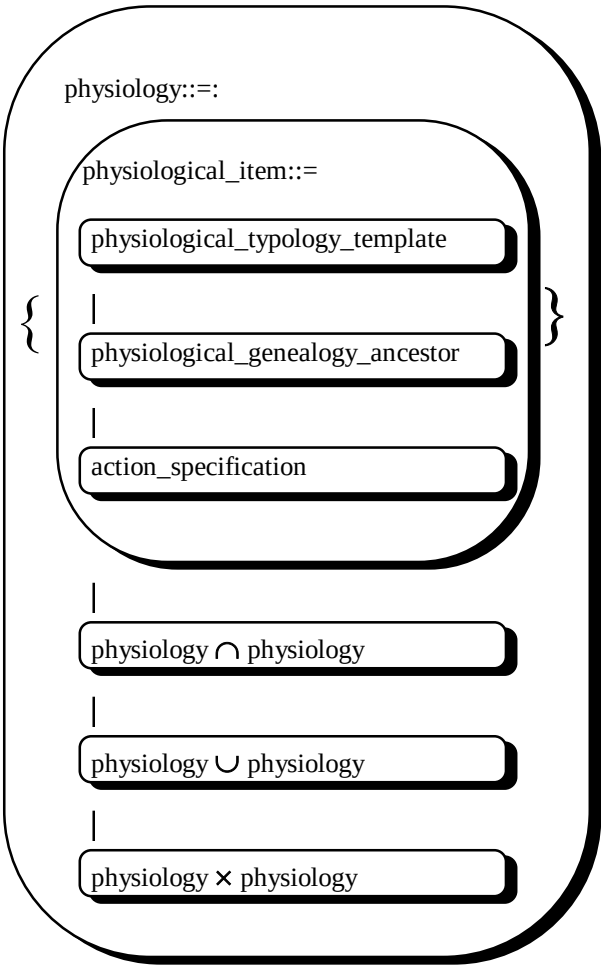


Figure 29: Map of physiological concepts

3 SPECIFICATION OF DESCRIPTION LANGUAGES

A description language is a formal tool which can actually express systems and must conform with a specification.

To express something in this field of activity means to ship description of physical system between human cognition and computer artificial mind. A computer needs an immuable rigor at any level of information, within the expression of any system, in order to well scan this expression. Yet, the human cognition involves, some other supplementary features for describing its knowledge of a system or well acquiring it; for example the possibility to emphasise away in time or place some global aspects of a description without erosion for the others, particularly those abundant of the deepest level of detail.

Consequently, there is an outline of a set of specifications for defining system description language. All these specifications are consistent with the frame given by the primitive formalism: this mathemùatical origin is a garantee for the rigor. Some others introduce the features wich are necessary for the ease of human mind, without change for the prevalent rigor.

Spec. 1 High level descriptive language with semantic and syntax based upon a sound abstract formalism: the descriptions using this language may be processed for formal validation.

Spec. 2 The language must allow global descriptions of objects belonging to different subject areas (multidisciplinarity). The concept of global description is defined in the formalism.

Comment on Spec. 2 — The CSMF standard must specify languages for expressing objects whatever their respective origins. An object may be: simple data, signal (analogue voltage transmitted through a cable, configuration of binary elements transmitted through a BUS, …), a program, a hardware device, a machine tool, etc.

Spec. 3 — The language must allow the description of collection of objects

Spec. 4 — The language must allow the description of a set of objects.

Comment on Spec. 4 — This item has two aspects:

- The set may be an existing collection.
- The set may be directly defined by collectivising properties. Then it is a template for defining any particular object having these properties.

Spec. 5 — The language must allow the description of a generic family of objects.

Spec. 6 — For any given object belonging to a particular subject area, the language, if necessary must embed, a more detailed description using specific language.

Spec. 7 — For any given object, in assistance of a top-down analysis, the language must allow the expression of the collection of the lower level objects which compose this given object (analysis of an object morphology).

Comment on Spec. 7 — The analysis of an object implies to examine how the existing lower level objects can act. Three cases are possible:

- This question does not matter immediately.
- The definitions of the lower level objects are sufficient to ensure a complete definition of the behaviour of the synthesised object.
- These definitions are insufficient. Then a set of behavioural rules must be stated in order to complete the description of the synthesised object. This implies the spec. n° 10

Spec. 8 — For any given collection of objects, in assistance of a bottom-up analysis, the language must allow the description of any upper-level object having this collection as morphology (synthesis of an object morphology).

Comment on Spec. 8 — The synthesis of an object is obtained whenever other objects are gathered in order to form its morphology. It is then necessary to examine how the combination of these objects can act. Two cases are possible:

- The definitions of the lower level object behaviours are sufficient to ensure a complete definition of the behaviour of the synthesised object.
- These definitions are insufficient. Then a set of behavioural rules must be stated in order to complete the description of the synthesised object. This implies the spec. n° 10

Spec. 9 — For any system, the respect of the specification 7 and of the specification 8 implies the availability of the language for the description of objects belonging to different levels of details.

Comment on Spec. 9 — The concepts of macro-description and micro-description must be introduced there.

- Example of macro description: the description of cooperating workshops in an enterprise..
- Example of micro description: the description of a hardware device.

Spec. 10 — For any given collection of interacting objects, the language must allow the expression of the conditional, or unconditional temporal rules, of their individual actions and of their interactions (physiology of the collection).

Comment on Spec. 10 — If the given collection constitutes the morphology of an object, then these rules complete the definition of its behaviour.

Spec. 11 — For any given object, the language must allow the definition of a time interval supporting the description of the object.

Comment on Spec. 11 — This time interval is somewhere upon the real time scale. It is the purpose of the behavioural rules to indicate the moment (« lower bound of the interval ») from which the object acts (use of trigger, on/off actuator, …).

Spec. 12 — For any given object, the language must allow the definition of a space location belonging to the description of the object.

Spec. 13 — For any given object, the language must allow the topological[33] description of its set of states within the time interval supporting the definition of the object.

Comment on Spec. 13 — The set of the states of an object may be continuous or discrete.

---

33 In this specification the word « topological » deals with the mathematical structure; it does not comply with the definition of IRDS which is too restrictive (bibliogr.: IRDS - part1, p.94).

Spec. 14 For any given object, the language must allow the description of the changes of states and of the association of changes of states.

Comment on Spec. 14 The state of an object is an object itself. Then a collection of objects completed with a set of rules of changes for each pair of objects constitutes a new object.

Spec. 15 The language must allow the description of objects processing other objects. A given object may be, both, an object processing other objects, or an object being processed by an other object.

Comment on Spec. 15 Particularly, two important Examples must be emphasised:

- The transformation of an object having a continuous time support into an other object having a discrete set of time supports (sampling).
- The transformation of an object having a continuous set of states in an other object having a discrete set of states (« quantifying »).

Spec. 16 For any given processing object the language must allow the specification of the exchanges between this object and its environment.

4 PRODUCTION OF LANGUAGES

4.1 General

The efficient designer of system description must be attentive of many properties of the described system, but his first concern should be to frame his description elements so that they convey the exact meaning intended. For that purpose, any CSMF must contain a formal description language. This kind of language is usually provided by the used CSMF. We have now to establish a mechanism for validating such a language in respect to the primary formalism.

4.2 Rules of meta-production

Many different description languages are available in IT. This paragraph proposes a mechanism for attempting to link any language definition with the primary formalism of the CSMF standard. For that purpose, the notions and the rules of the primary formalism are used as meta-notions and meta-rules. Then some rules, named « rules of meta-production », must be defined especially for any description language L, for linking it with the primary formalism. These meta-rules transform the meta-notion in notions belonging to L. Similarly they transform the meta-rules in rules belonging to L.

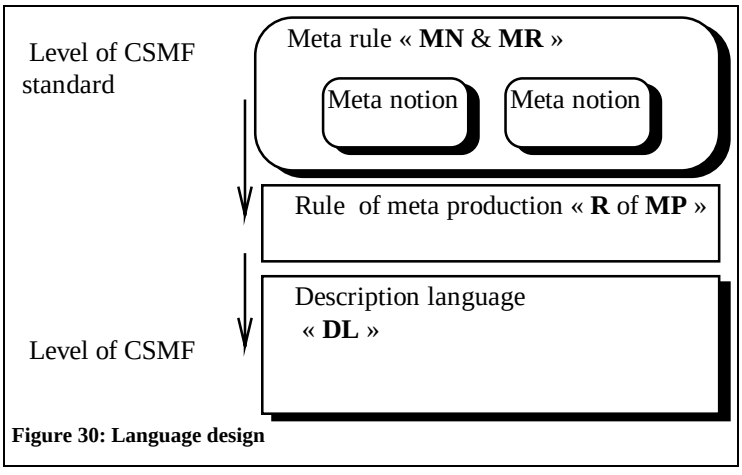


Figure 30: Language design

4.3 Conformity criterion

The possibility of defining a set of such rules of meta-production for each given language L is unprovable. In addition, if the primary formalism is an element of the CSMF standard, this possibility is a criterion for proving the compliance of the examined language with the CSMF standard.

An example of description language is given in the Appendix IV, page 63. This language was defined more than ten years before this study. We will attempt now in this appendix to link its definition with the primary formalism defined in this part.

PART 5

CONCLUSION

The introductory quotation from Leibnitz placed before the theoretical elements (3rd part) shows how old this idea is: create a universal language for the statement of any problem and as a medium for solution by reasoning. In his correspondence, Leibnitz often affirmed the existence of this language but did hand down anything to us. With respectful irony, René Thom qualifies this vision today as an « old Leibnitzian dream ». He however suggests, seriously, that the only language having these virtues could quite simply be mathematics.

The study was carried out based upon this idea. In terms of results, it proposes a triptych: a pragmatic method for analysing and modelling information systems based on a catalogue of physical concepts; then, theoretical elements supporting their mathematical modelling; finally, a language for expressing the models built by means of this method and this means.

Physical concepts

In this approach, the method remains the primary element. We shall rapidly conclude on this subject because the classicism of its elaboration and its results call for few comments. Through a set of physical concepts, the method sets forth a vision of information systems based upon long experience with such systems in scientific and industrial circles. This experience was acquired by personal work and reinforced by the contribution of colleagues working along the same lines. It is however restricted in the end to a field of application with boundaries that are still somewhat fuzzy. Such is the fate of all scientific investigation methods: only long usage will make it possible to solidly assess their value and to determine their limits gradually, as work progresses and years go by.

Theoretical elements

Our hope now is that the models thus devised will become useful, if not indispensable data for reasoning on the properties of the physical systems investigated. The complexity and the size of the models resulting from the method previously described require a precise and rigorous expression and practically call for the use of powerful information processing facilities.

The chosen means of expression is mathematical modelling. It is a sufficiently powerful tool to meet the requirements of the method and follow the experiences of pragmatic models. Mathematics brings or takes nothing from the assumed fidelity of any model built by means of physical concepts, with regard to the physical being that it represents. The virtues of mathematical modelling lie essentially in the precision and rigor of its definitions, its constructions, and its notations: communication and reasoning, essential parts of any subsequent work, are now based on models void of ambiguity. These models form a family whose general properties have been presented extensively in the part of the study devoted to them.

Information processing facilities are processing systems omnipresent today but have a basic characteristic that must be pointed out here: they are systems with a behaviour that is discrete with respect to time and processing only discrete data sets.

What, now, is the possible use of mathematical models? The following proposals concern exclusively any mathematical model of the type proposed in this study and its use as data for computer processing facilities.

For this purpose, let us return for a moment to a classical engineering problem. We all know that the bugbear of a designer or of a system manager is the risk of failures. The identification and nature of possible failures, their causes and their effects are thus of major concern. Simulation techniques provide certain presumptions in this respect during the validation of the systems but never any absolute certainty. Some investigators have consequently turned to methods of formal expertise executed on mathematical models. This is done currently in other areas: a mathematical model, if it is reliable, expresses the qualities of the physical world, its advantages, its drawbacks and, in our case, potential system malfunctioning. Let us briefly examine the nature of the work that can be executed by means of models we know how to build.

Two activities must be distinguished in this examination: that of the mathematician and that of the engineer.

Let us begin with the mathematician's activity and let us consider anyone of the mathematical models concerned.

Let us first eliminate the possibility of any gratuitous disrespect for mathematics in the construction of the model. What is involved would be purely errors of writing. The element of the model having such a fault is most currently void of any meaning; compilers exist ordinarily to detect such errors. There is still the exceptional case in which this disrespect could lead, by extraordinary chance, to a mathematically correct expression which is whimsical in its relations with the modelled physical object. The model would then be like the many other models resulting from imperfect observations of reality, both mathematically correct and physically incorrect. The validation of the models is in fact the function designed to locate and reduce such defects; this classical function forms part of engineering.

After having rid the mathematical model of any mistakes on the part of the writer, we can now assume that there is formal perfection; we have a set of assertions, of complex structure, concerning the physical properties of the investigated system. And, experience warns us that these assertions may contain other contradictions, despite all the preceding precautions. This is even a quite frequent and obviously troublesome case. The formal rectitude of the model now being established, it is the physical rectitude of the physical system that must checked. Guided by these contradictions, unjustly scorned, one generally then discovers latent and insidious risks of inconsistent behaviour, failures and even disasters. It is thus important to reveal and study any contradictions of a mathematical model because they reveal latent ills of the physical system. What procedure shall we use?

We have distinguished two parts in a pragmatic model, conserved in the mathematical model:

- the primary morphology which determines the direct descendants of the modelled physical object;
- the primary physiology which determines the rules of activity of these descendants.

Each descendent is also a physical object capable of being modelled in the same manner, and so on. The reader is referred to the part devoted to physical concepts for details on these definitions.

Primary morphology and physiology are both sets of assertions.

Primary morphology that results from a first analysis of the physical object is a finite sequence of assertions, each meaning intuitively: « x, of the X type, exists ». Each assertion is a true proposition by construction; their sequence represents their conjunction meaning intuitively:

« (a of the A type exists) and (b of the B type exists) and… (z of the Z type exists) ».

The primary morphology is thus globally a true proposition. Consequently, we do not find the contradictions, if they exist, at this modelling level. This assurance is unfortunately of limited value because a primary morphology is only a sort of first inventory determining an initial collection of inert objects whose fine descriptions and animations still remain to be created.

A physical object has different behaviour possibilities that have been modelled using the state concept. To each object corresponds a collection of possible states. The purpose of primary physiology is to define, for each object, the associated primary morphology, the temporal and non-temporal rules of occurrence of these states. Each rule is an action. The legitimacy of these rules does not belong to the world of mathematics; it is an engineering matter. On the other hand, these multiple rules may be gratuitously contradictory in the determination of the state of an object, the different states mutually excluding themselves, and legitimately in this mathematical modelling technique. These contradictions are generally concealed by the overlapping of assertions resulting from the recursiveness of the modelling technique.

Let us review intuitively here, by simplifying to the utmost, the two basic meanings of an action:

« if « condition C » then at t « state(a)=Ai » otherwise at t' « state(a)=Aj » » (pulse mode)

« if « condition C » then « from t1 to t2 « state(a)=Ai » » otherwise « from t'1 to t'2 « state(a)=Aj » » » (on/off mode).

A model is not a theory in which the different assertions are axioms or theorems deduced from axioms according to the rules of logic. Any relations between assertions express formally a physical reality, which is what distinguishes the model from a demonstration. In particular, the different actions are independent if the physical behaviours represented are independent. On the other hand, they are related by relations translating the relations between the physical components, if necessary. As in any human construction, a system is fallible, and the modelled system may exhibit conceptual errors translated into the model by inconsistencies

The search for contradiction is theoretically simple: for each state « Ai » of an object « a » and for all the actions governing this state directly or indirectly, its characteristic domain is constructed in the space made up of parameters that are involved in its determination. This characteristic domain is the domain « Di » of which « state (a) =Ai » as a function of these parameters. In order for there to be no contradiction between the assertions determining any « Ai » and « Aj » couple with respect to time, it is necessary and sufficient for the

intersections of the characteristic domains « Di » and « Dj » of these states by the same « plane » to be disjointed whatever t.

Practical research is far more difficult. Determining the intersection of two domains varying with respect to time is possible only if the domains are themselves determined. As processing facilities are digital machines, these domains need only be calculable. And, time and any object « a » are objects that may have complex structures: their respective sets of states may be chosen independently as continuous or discrete depending on modelling requirements. Their calculability at any instant is therefore impossible in the general case. There is in fact a countable, finite and often even highly populated set of calculable cases, but it always remains a set having the power of the continuum of noncalculable cases. The question of knowing whether the countable and finite set of calculable cases is sufficient to provide a satisfactory answer is a question difficult to analyse numerically. It is a sort of discretization of the system into finite elements whose validity, when the solution is found, is an engineering problem[34].

The severity of this diagnosis is compensated by the fact that certain families of physical systems have models whose properties restrict the problem: for example, so-called discrete sequential systems for which the time and the set of values are discretized. In this case, the assertions are always calculable in principle, and it is then possible to hope to find partial solutions, some of which are in fact found. The accumulation of such solutions and the fact that knowledge on the subject is growing with the years induces the researcher to continue along these lines but should not obscure the theoretical possibility of finding a general solution with numerical processing facilities.

This approach brings us to engineering because it depends on the existence of system families for which its results can be validated.

The examination of the engineer's viewpoint is limited to the examination of two frequent exemplary cases.

The first example is the verification of the compatibility of input and output specifications in the exchange of products between the elements of a system. In a large system, this is typically the first source of failures, which may very reasonably be confined by a systematic examination in the mathematical model of all the morphological conditions of exchanges. This is where it is possible to find the grain of sand forgotten and which will later and quite unexpectedly jam the entire system! By « morphological condition » is meant any specification involved in the definition of a product when it is transmitted and then received. Such an inspection quickly becomes humanly impossible, whereas it can be validly performed by software, not only during the design of the system but also after all the maintenance and upgrading operations which affect its morphology.

The second example is the mutual waiting for two physiologies at a meeting point: an action « a » in one physiology « A » waits for an action « b » in another physiology « B » to be terminated and vice versa. Petri network specialists will recognise this as an example of a partial subgraph which is easy to correct, when one knows where it is hidden. This is a frequent case in complex evolving systems and, as in the preceding case, having a model that evolves with the system makes it possible to locate many points of such blockage.

Language

Then there is the question of language. Language is designed for the writing of texts easily legible by individuals and machines, while also faithfully expressing the mathematical models.

When the theoretical elements are acknowledged, the definition of the language must comply with the following specification: its vocabulary and its syntax must present faithfully and exhaustively the elements of the theory. This is a stringent rule but practical solutions are possible. In fact, while remaining within the strict framework of this specification, the definition of a language, i.e. of a formal system of notations, has no limits other than those of the imagination.

However, the remark just made with respect to standardisation work has the utility of defining a standard language whose legibility would not necessarily be the main quality. Its purpose would be to serve as an intermediary for any translation of a model expressed in a language based on theoretical elements into an expression in another language different in its appearance, but also complying with the same theoretical elements. This provision favours the creation of languages specific to certain scientific or technical disciplines, the existence of which becomes inevitable owing to the multidisciplinary nature of the systems examined.

Finally, it is noted that the number of problems raised following this study exceeds that of the few questions for which we propose a solution. Having completed this stage, let us put down our instruments. We can take our closing words from either of two great French mathematicians, as the reader chooses.

---

34 Let us note that Shannon's theorem provides an answer to this question for certain objects: signals.

The first, Bézout, has a style which shows his era. The intense jubilation of having completed something is difficult to hide behind the extreme modesty of pronouncements befitting a decent man of the 18th century.

> *Nous nous estimerons heureux si considérant le point où nous avons pris les choses, et celui où nous les amenons, on trouve que nous avons acuité une partie du tribut que tout homme doit à la société dans l'état où il se trouve placé.*[35]
> (We shall regard ourselves as fortunate if, considering the point at which we took things over, and the point to which we shall take them, it is found that we have completed part of the tribute that any individual owes to society.)

The second, which is contemporary because it has to do with Laurent Schwartz, expresses equal satisfaction, judging from the last, clear and limpid line of one of his many valuable works:

> *Ouf* ![36]
> (Phew!)

35 M. Bézout. *Théorie générale des équations algèbriques. Préface.1779.*
36 Laurent Schwartz. *Cours d'analyse. Ecole Polytechnique*.

PART 6
APPENDIXES

I LIST OF DEFINITIONS

Action
SC21 7054

Something which happens.
…
Notes -1: when used without qualification, action means « action occurrence ».
…

Action
IRDS - part1

An event whose agent (in this case, an actor) is volitional. Action is a relation among the following: a spatio-temporal location (i.e. a location in space and time); a volitional agent; a phenomenon; an instrument; and an act. One or more elementary actions that, as a unit, change a collection of sentences into another collection of sentences in the information base or conceptual schema and/or make known a collection of sentences present in the information base or conceptual schema. (Page 90).

Activity
SC21 7054

A single-headed directed acyclic graph of actions where occurrence of each action in the graph is made possible by the occurrence of all immediately preceding actions.

Behaviour
SC21 7054

A specification of a set of activities.
…

Behaviour
SC21 9563
(LOTOS)

… is defined by the LOTOS behaviour expression associated with the process definition that constitutes the object template. …
…

Behaviour
SC21 9563
(SDL)

The behaviour of a *process*/*service* is the set of all transitions of that *process*/*service*. …
…

Behaviour
SC21 9563
(Z, p.19)

The behaviour of an object in a given state is the set of all possible activities that may occur from that state. ….
…

Behaviour
SC21 9563
(Estelle, p.25)

The behaviour of an object is determined by the set of all *transitions* of that object. …
…

Class
CSMF SOU-07

A class is an abstraction of objects having common properties. Such an object is called an instance of the class. When an object is an instance of a class, we say the object belongs to the class

Environment
EWOS/ETG 012
ISO TR 9007

(of information system). That part of the real world containing the users which exchange messages with the information system.

Inheritance

(p.961)

Inheritance is a mechanism for sharing code and behaviour. It allows to reuse the behaviour of a class in the definition of new classes. Subclasses of a class inherit the operation of their parent class and may add new operation and new instance variables.

Interface
EWOS/ETG 012
ISO 2382-1

A shared boundary between two functional units, defined by functional characteristics, common physical interconnection characteristics, signal characterisrics, or other characteristics, as appropriate.

Interface
(p.328)

… To work in a system, every product must comply with precise rules governing its intended relationships with other products, with user software, and even wuth user interactions. Such relationships are called *interfaces*. …

Interface
SC21 7054

An abstraction of the behaviour of an object obtained by considering only a specified subset of the observable actions of that object.
…

Interoperability
EWOS/ETG 012
IEEE 729

The ability of two ore more systems to exchange information and to mutually use the information that has been exchanged. (IEEE 729).

Interoperability
TSG1
IEEE 729

The ability of two ore more systems to exchange information and to mutually use the information that has been exchanged. (IEEE 729).

Location in space
SC21 7054

An interval of arbitrary size in space at which an atomic action can occur.

Location in time
SC21 7054
An interval of arbitrary size in time at which an atomic action can occur.

Location
SC21 7054
An interval of arbitrary size in time and space at which an atomic action can occur.

Object
Objects in programming languages are collections of operations that share a state.
…
The collection of methods of an object determines its interface and its behaviour.

Object
Object denotes the whole representation of an application data model. …

Object
SC21 7054
A model of an entity. An object is characterized by its *behaviour* and, dually, by its state (…)
…

Object
CSMF SOU-07
An object is a representation of a real world object. An Object has an identifier that identifies itself in an application data model occurrence. …

Object
IRDS-part 1
Something toward which a cognitive act (i.e. a though) or an action is directed. A material body in the mid-world.

Object
ISO/IEC JTC1 TC22 N1712
(External). Hardware and/or software unit that has a behaviour and interaction modes, with Ada programs, defined by this standard

Object
ISO/IEC JTC1 TC22 N1712
(Logical). Conceptually, a software object that is created and associated with an external object and that has an acccessor. [Software object is not defined in this document].

Portability
TSG-1
(Software). The ease with which software can be transferred from one application platform to another.

Portability
EWOS/ETG 012 TSG-1
(Software). The ease with which software can be transfered from one information processing system to another.

Portability
TSG-1
(Application). The ease with which an application can be transferred from one application platform to another.

Portability
EWOS/ETG 012 TSG-1
(Application). The ease with which an application can be transferred from one application platform to another.

Portability
EWOS/ETG 012 ISO-2382-1
(Program). The capability of a program to be executed on various types of data processing systems without converting it to a different language and with little or no modification.

Process
IRDS-part 1 ISO TR9007
A collection of activities performed in a set order on a prescribed set of constructs under the constraint of rules.

State
SC21 7054
At a given instant in time, the condition of an object that determines the set of all sequences of actions in which the object can take part.
…

System
…, a system appears as a coherent collection of products, both hardware and software. …

(p.328)

System
EWOS/ETG 012 *OED*
A set of connected things, parts, elements working together in a regular relation.
A set of connected things, parts, or elements working together to achieve a common objective.
Ordered set of ideas, concepts, principles.

System
BSI DD 210 *OED*
(business). A system which performs one or more of the tasks of one ore more business funcions by transforming a set of inputs, using a set of rules and procedures, to produce a set of outputs.

System
IRDS-part 1
A methodical or logical plan or arrangement governed by a set of principles, or business rules and procedures.

Template
SC21 7054
The specification of the common features of a collection of objects.
…

Type
SC21 7054
(of an object). A predicate. An object is of the type, or satisfies the type, if the predicate holds for the object.

Universe of Discourse
IRDS-1
Those entities and happenings of interest that have been, are or ever might be and about which there exists a collection of represented information having a common understanding.

II. 1 Category

This part concerns the use of the theory of categories. This theory is certainly one of the most abstract approaches of modern algebra. The understanding of its axiomatic foundations is difficult but essential for its strict overall use. In particular, the axioms distinguish two concepts: classes and sets, in order to avoid certain well-known paradoxes of set theory.

Nevertheless, category theory introduces the restrictive notion of "small category", which is simpler than the general notion of "category" and better suited to our needs. The following elements provide a review of the theory which is sufficient for this project.

The modelling of a system generally requires the engineer to state his physical hypotheses concerning the system; these hypotheses are assertions concerning the natural properties of the system, making it possible to choose the abstract elements representing the concrete elements of the investigation domain. We shall now examine how mathematical elements can be used for this purpose.

II.1.1 Definition

def. 17: Consider an entity **C**[37]; this entity is a "small category" if we can assign the following properties to it[38].

a) Initial definitions

Considering,

def. 18: a set: **obj**(**C**) = {X, Y, Z, … }; X, Y, Z, … are the "objects" of **C**;

def. 19: a set: **mor** (X,Y) = {f}, associated with any pair (X,Y) of objects of **C**;

each element f is a “morphism” of the “domain” X towards the “co-domain” Y and noted:

$f: X \rightarrow Y$

In our study, this set is either empty, either a singleton;

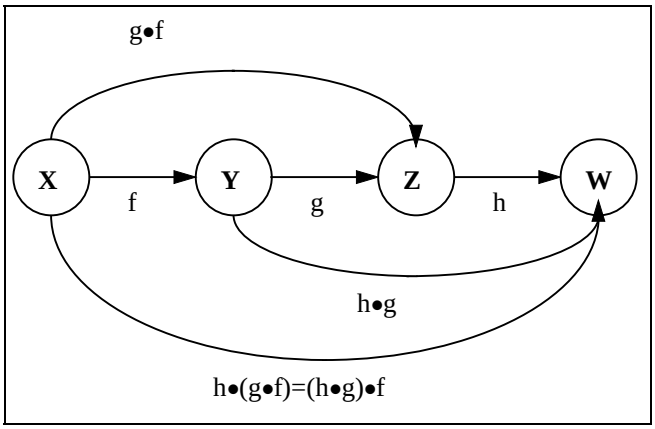


Figure 31: Category

def. 20: a function •: **mor** (X,Y)× **mor**(Y,Z)→ **mor**(X,Z), associated with any triplet (X,Y,Z) such as **mor**(X,Y)≠∅and **mor**(Y, Z) ≠∅ and **mor**(X, Z) ≠∅;

this function determines a composition:

if f∈**mor**(X, Y) and g∈**mor**(Y, Z), then there is exactly one k∈**mor**(Y, Z) and k = g•f.

37 Within the framework of the general theory of categories, this is a "small category".

38 There are restrictions in the definition of a category: category theory uses the notion of “class” which is more general than the notion of "set" and extends the notion of function to classes.

b) Axioms

axiom 1: the composition • is associative;

if f∈**mor**(X,Y) and,
g∈**mor**(Y,Z) and,
h∈**mor**(Z,W),
then (h•g)•f=h•(g•f);

axiom 2: for any element Y, there is a **mor**(Y,Y) having at least one element u_Y; this element is such that:

for any X and any f∈**mor**(X,Y) ≠∅,
u_Y•f=f,
and,
for any Z and any g∈**mor**(Y,Z) ≠∅,
g • u_Y = g

c) Comment

In the definition (def 19), the case of the existence of an empty set of morphisms must be discussed.

If **mor**(X, Y)=∅ (resp. **mor**(Y, Z)=∅), then f (resp. g) is nonexistent; its composition with any other morphism is consequently nonexistent. This circumstance is compatible with the axioms. Let us now suppose that this **mor**(X, Y)≠∅ and **mor**(Y, Z)≠∅ but that **mor**(X, Z)=∅. This circumstance, possible in a model, leads to a contradiction with the existence of the composition law. The associativity axiom is contradicted, and consequently the model is no longer a category.

II.1.2 Opposite category

def. 21: Let **C** be a category. The opposite category $\mathbf{C}^{\mathbf{op}}$ is such that:
- **obj**($\mathbf{C}^{\mathbf{op}}$) = **obj**(**C**);
- there is a bijection between **mor**(X, Y) and $\mathbf{mor}^{\mathbf{op}}$(Y, X).

A mnemonic means of defining $\mathbf{C}^{\mathbf{op}}$ consists in reversing all the arrows in the graphic representation of **C**.

Example 38: A series of data having a "$skip^+$() function is a category if $skip^+$ (0) and $skip^+$ (n+m)=$skip^+$ (m) • $skip^+$ (n) are defined. The opposite category is obtained with the function $skip^-$().

II.1.3 Product of categories

Let **C** and **C'** be two categories.

def. 22: The product **C**'' = **C** × **C'** is such that:
- **obj**(**C''**) = **obj**(**C**) × **obj**(**C'**),
- if f ∈ **mor**(X,Y) in **C**, and f' ∈ **mor**(X',Y') in **C'**, then f'' = (f,f') ∈ **mor**((X,X'),(Y,Y')),
- if f '' ∈ **mor**(X'',Y'') and g'' ∈ **mor**(Y'',Z'') in **C''**, then h'' = g''•f'' = ((g•f),(g'•f')).

II.1.4 Ordered set

The ordered set is an example of an entity assimilable with a category; this example is essential in our methodology.

Let us consider a set S = {X, Y, …}. S is ordered if there is a binary relation " ≤ " which is transitive, associative and reflexive, and a set O⊂S×S such that:

$O = \{(X,Y)/\ X \leq Y\}$

(i) If **C is the entity**, we declare that:
- S=**obj**(**C**);
- for any pair $(X,Y) \in S \times S$ there is **mor**(X,Y), which is either empty or equal to the singleton $\{\leq\}$;
- The transitivity of $\leq$ leads to the existence of:
  $\bullet$: **mor**(X,Y)$\times$ **mor**(Y, Z) $\rightarrow$ **mor**(X, Z).

(ii) Let us note that:
- The transitivity of $\leq$ implies the associativity of $\bullet$.
- Reflexivity implies the existence of the singleton $\{u_X\}$ = **mor** (X,X) for any $X \in S$.

II. 2 Fonctor

II.2.1 Definition

def. 23: Let **F** be an entity; this entity is a fonctor if we can assign the following properties to it.

a) Notations

Let **C** and **C'** be two categories and F an: **obj**(**C**) $\rightarrow$ **obj**(**C'**).
Let X and Y be any two categories of **C** and their respective images X' and Y' in the objects of **C'.**

b) Axioms

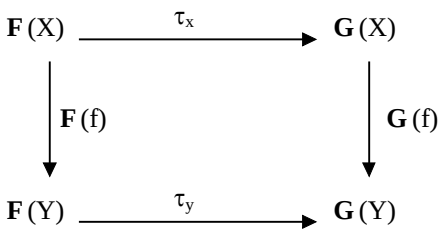
axiom 3: for any couple (X, Y) F determines an application F: **mor**(X, Y) $\rightarrow$ **mor**(X', Y') with the condition:
if Y = f.X, then Y' = f'.X' with f' = F(m).

axiom 4: F conserves the morphism composition law:
if k = g$\bullet$f, then k' = g'$\bullet$f '
The following equalities result from this axiom:
Z' = k'.X' = (g'$\bullet$f ').X' = g'.Y'

axiom 5: F conserves the associativity of the composition law:
if k' = h'$\bullet$g'$\bullet$f', then k' = (h'$\bullet$g')$\bullet$ f' = h'$\bullet$ (g'$\bullet$f')
The following equalities result from this axiom:
W' = l'.X' = (h'$\bullet$g'$\bullet$f').X' = (h'$\bullet$g').Y' = h'.Z'

II.2.2 Natural transformation of a fonctor

def. 24: Let **F** and **G** be two functions such that: **F**, **G**: **C** $\rightarrow$ **C'**. τ is a natural transformation if it associates with any $X \in$ **obj**(**C**) a morphism τ_x: **F**(X)$\rightarrow$ **G**(X) such that for any $f \in$**mor**(X, Y) the following diagram commutes:



II.2.3 Universal element

def. 25: Consider a fonctor **F**: **I** → **O**. The objects of **I** and **O** are sets. A universal element **F** is a pair (u, R) composed of an object R ∈ **obj** (**I**) and an element u ∈ **F** (R) having the following property: for any object X ∈ **obj** (**I**) and any s ∈ **F** (X), there is a unique morphism f: R → X such that **F**(f)u = s.

Example 39: Consider two countable series having the number of records n: D={d_i}, N={n_i}. Now consider a category C such that:
obj(**C**)={N, D},
mor(N, N) = **mor**(N, D) = **mor**(D, N) = **mor**(D, D) = {Ω_i}.
Any Ω_i is a circular permutation matrix [$\varpi_{\alpha\beta}$] in which $\varpi_{\alpha\beta}$ = 1 if $\alpha = \beta$-i+1 (mod α_{max}), otherwise 0. These permutation matrices are composable and this composition is associative; the result is a cyclic permutation matrix; this composition is associative. For any X (X = D or N), we say that Ω_i: N → X selects exhaustively the record i in X. This assertion is obviously analogous to Ω_i: n_1 → x_i (= n_i or d_i). We now state that N is a given permutation (not restricted to a cyclic permutation) of **mor**(N, D): N appears as an index file for the file D.
We give a transformation represented by the fonctor **F**. We choose F such that:
F{Ω_i} = {Ω_i},
any **F**(Ω_i): **F**(N) → **F**(X) points to the record i in **F**(X).

II.2.4 Representation

def. 26: Consider a fonctor **F**: **I** → **O**. The objects of **I** and **O** are sets. A representation for **F** is a pair (R, φ) composed of an object R ∈ **obj** (**I**) and of a family of bjiections φ_X: **mor**$_X$(R → X) ≡ **F**(X).

The latter condition implies that, if ν_1 = **F**(n_1), then for any i, x_i = **F**(Ω_i)ν_1. Consequently, (ν_1,N) is universal for **F**.

III DEMONSTRATIONS

dem.1: Temporal translation of a process

The processes, elements of $\gamma'_{\tau+t}$ represent the same phenomena as the processes which are elements of γ'_{τ} to within a translation t with respect to time. Conversely, any value of t determines in Γ' an operator transforming each γ'_{τ} into its translation $\gamma'_{\tau+t}$. Consequently, there is Γ_t: $\gamma'_{\tau} \rightarrow \gamma'_{\tau+t}$.

In particular, for $\tau=0$, Θ_t: $\gamma'_0 \rightarrow \gamma'_t$.

dem.2: Sampling and quantization

- $\tau \in [\tau_1, \tau_2] \cap [t_1, t_2]$

  Hypothesis 12 leads to: $\delta[\tau] \otimes g'' = \delta[\tau] \otimes (\delta[\tau] \bullet g'')$;
  Hypothesis 13 entails directly: $\delta[\tau] \otimes (\delta[\tau] \bullet g'') = \delta[\tau] \otimes (\delta[\tau] \bullet g)$;
  Hypothesis 12 again leads to: $\delta[\tau] \otimes (\delta[\tau] \bullet g) = \delta[\tau] \otimes g$; consquently, the first relation is demonstrated.

- $\tau \notin [\tau_1, \tau_2] \cap [t_1, t_2]$

This condition leads to $\delta[\tau] \bullet g'' = \delta[\tau] \bullet g = \delta[\tau] \bullet g_0$, which enables us to substitute g_0 for g in all the above relations; the second relation is thus also demonstrated.

dem. 3: Topology of Γ'

By construction, there is a bjiection between T and Γ'; Γ' consequently has the same topology as T and consecutively the power of the continuum. Let us note this topology as $T_T(\Gamma')$.

dem. 4: Topology of substrate space

Let us consider a particular process g of domain $[t_1, t_2]$. For any interval $[t_1, \tau_2]$ included in the preceding (or equal), there is $g'' = \Delta_{[t1, \tau2]} \bullet g$. By construction, the set {g''} has the same power as $[t_1, t_2]$, i.e. the power of the continuum; also by construction, the elements of {g''}are moreover different two by two. But, the definition of the equivalence relation ρ indicates that any element of {g''} has an equivalent g'' in γ'_0; still by construction, these elements are different two by two; their set {g''}has at least the power of {g''}. {g''}is a subset of γ'_0; γ'_0 thus has at least the power of the continuum.

dem. 5: Portability

Let us note first of all that any symbol may be chosen in the definition 20 (Page 22); in particular, we can permute the symbols **F** and **G** without altering this definition.

- The condition is sufficient:

If all the τ_X are invertible, the inversion of the morphisms leads to the plotting of diagrams identical to that of the definition, allowing for the symbols chosen. The preceding remark means that we find the definition of the natural transformation.

- The condition is necessary:

Let us suppose that a single morphism is not invertible, for example τ_X: $\mathbf{F}(\mathbf{X}) \rightarrow \mathbf{G}(\mathbf{X})$. In this case, the construction of the diagrams shows that the transformation is in conformity with the definition for τ_X and τ_Y and false for τ_X and τ_Y^{-1}. The supposition leads to the impossibility of applying definition 22. The condition is thus necessary.

IV EXAMPLE OF LANGUAGE PRODUCTION

IV.1 General

There is an example of production of a description language named « Dad ». This example was first published in its preliminary version in 1983[39]. A revised version was proposed in an appendix of a report on the use of Ada for system description. This report was submitted to the Commission of the European Communittee in 1984 and published by the Cambridge University Press in 1985, under the title « Ada for specification: possibilities and limitations »[40].

IV.2 Basic rules of production

The rules of production are those used in the definition of Ada in its first standardized version[41]. The production rules for which we propose adaptations are given only after that introduction, labelled with their initial reference. Within these new rules, the notions or the parts of notions chosen as the roots of the extensions are indicated by capital letters. The meta-rules and rules of meta-production associated with these roots are given in the two following sections.

IV.3 Rules of production

Bold-face references in the first column are paragraph numbers in the referenced Ada manual[41].

```
2.8.    REFERENCE          ::=   REFERENCE'SYMBOL identifier [(argument-association
                                                    {, argument-association})]
3.1.    basic-             ::=   object-declaration
        declaration              / FLUX-declaration
                                 / DEVICE-declaration
                                 / PROCESS-declaration
                                 / exception-declaration
                                 / renaming-declaration
                                 / number-declaration
                                 / SUBFLUX-declaration
                                 / SYSTEM-declaration
                                 / generic-declaration
                                 / generic-instantiation
                                 / defered-constant-declaration

3.2.    object-            ::=   identifier-list : [constant] SUBFLUX-indication
        declaration                                              [:= expression];
                                 / identifier-list: [constant] constrained-array-definition
                                                                 [:= expression];
3.3.1.  FLUX-declaration   ::=   full-FLUX-declaration
                                 / incomplete-FLUX-declaration
                                 / private-type-declaration
        full-FLUX-         ::=   FLUX'SYMBOL identifier [discriminant-part]
        declaration              is FLUX-definition ;
        FLUX-definition    ::=   enumeration-definition
                                 / real-type-definition
                                 / record-type-definition
                                 / derived-type-definition
                                 / integer-type-definition
                                 / array-FLUX-definition
                                 / access-type-definition
3.3.2.  SUBFLUX-           ::=   SUBFLUX'SYMBOL identifier is SUBFLUX-indication ;
```

39 Savoysky, 1983. *Eléments théoriques pour la description de systèmes automatiques. Thèse de Doctorat d'Etat, Univ. Pierre et Marie Curie, Paris-6, 1983.*

40 Goldsack and *al*, 1985: xvii.

41 Alsys, 1983: ANSI/MIL-STD 1815 A.

```
                declaration
                SUBFLUX-          ::=    FLUX-mark [constraint]
                indication
                FLUX-mark         ::=    « FLUX »-name
                                       / « SUBFLUX »-name
3.5.1.          enumeration-      ::=    (enumeration-literal-specification
                FLUX-definition                    {, enumeration-literal-specification })
3.6.            array-FLUX-       ::=    unconstrained-array-definition
                definition              / constrained-array-definition
                unconstrained-    ::=    array (index-subtype-definition
                array-definition                              {, index-subtype-definition})
                                         of component-SUBFLUX-indication
                constrained-      ::=    array index-constraint
                array-definition         of component-SUBFLUX-indication
                index-SUBFLUX-    ::=    FLUX-mark range « »
                definition
3.8.1.          incomplete-FLUX-  ::=    FLUX'SYMBOL identifier [discriminant-part]
                declaration
3.9.            MORPHOLOGICAL-    ::=    {basic-declaration-item}{later-declaration-item}
                part
                basic-            ::=    basic-declaration
                declaration-item        / representation-clause
                                        / use-clause
                later-            ::=    body
                declaration-item        / DEVICE-declaration
                                        / SYSTEM-declaration
                                        / PROCESS-declaration
                                        / generic-declaration
                                        / use-clause
                                        / generic-instantiation
                propoer-body      ::=    DEVICE-body
                                        / SYSTEM-body
                                        / PROCESS-body
4.4.            relation          ::=    simple expression [relational-operator simple-expression]
                                        / simple expression [not] in range
                                        / simple expression [not] in FLUX-mark
5.1.            BEHAVIOUR         ::=    GROUP {GROUP}
                ACTION            ::=    {label} simple-ACTION
                                        / {label} compound-ACTION
                simple-ACTION     ::=    null-ACTION
                                        / assignment-ACTION
                                        / procedure-call-statement
                                        / exit-statement
                                        / return-statement
                                        / goto-statement
                                        / EXCHANGER-call-ACTION
                                        / delay-ACTION
                                        / ORDERED-ACTION
                                        / raise-statement
                                        / code-statement
                compound-ACTION   ::=    if-ACTION
                                        / case-ACTION
                                        / ITERATIVE-ACTION
                                        / block-statement
                                        / EXCHANGE-ACTION
                                        / select-ACTION
                null-ACTION       ::=    null S
5.2.            assignment-       ::=    « variable »-name
                ACTION                   [PREPOSITION-PART] TIME-OPERATOR'SYMBOL expression S
5.3.            if-ACTION         ::=    if condition then
                                           BEHAVIOUR
                                         {elsif condition then BEHAVIOUR}
                                         [else BEHAVIOUR]
                                       endif S
5.4.            case-ACTION       ::=    case expression is
                                           case-ACTION-alternative {case-ACTION-alternative}
                                       end case S
                case-ACTION-      ::=    when choice {/ choice} ( BEHAVIOUR
                alternative
5.5.            ITERATIVE-ACTION  ::=   [ITERATIVE-simple-name :]
                                        [iteration-scheme] ITERATION'SYMBOL
                                          BEHAVIOUR
                                         end ITERATION'SYMBOL [simple-name] S
6.1.            DEVICE-           ::=    DEVICE-specification
                declaration
                DEVICE-           ::=    procedure identifier [formal-part]
                specification           / function designator  [formal-part] return FLUX-mark
                parameter-        ::=    identifier list : mode-FLUX-mark [:= expression]
                specification
6.3.            DEVICE-body       ::=    DEVICE-specification is [MORPHOLOGICAL-part]
```

```
                                    begin BEHAVIOUR
                                   [exception
                                      exception-handler
                                     {exception-handler}]
                                    end designator ;

6.4.    actual-parameter ::=    expression
                              / « variable »-name
                              / FLUX-mark (« variable »-name)
9.1.    PROCESS-         ::=    PROCESS-specification
        declaration
        PROCESS-         ::=    PROCESS'SYMBOL [FLUX'SYMBOL] identifier
        specification            [is {EXCHANGER-declaration}
                                     {representation-clause}
                                  end[« PROCESS »-simple-name]]
        PROCESS-body     ::=    PROCESS'SYMBOL body « PROCESS »-simple-name is
                                 [MORPHOLOGICAL-part]
                                  begin BEHAVIOUR
                                 [exception
                                    exception-handler
                                   {exception-handler}]
                                  end « PROCESS »-simple-name ;
9.5.    EXCHANGER-       ::=    ECHANGER'SYMBOL identifier [(discrete-range)]
        declaration                                                [formal-part] ;
        EXCHANGER-call-  ::=    « EXCHANGER »-name [actual-parameter-part] S
        ACTION
        EXCHANGER-ACTION ::=    ECHANGE'SYMBOL « EXCHANGER »-simple-name
                                [(EXCHANGER-index)][formal-part]
                                 [do
                                  BEHAVIOUR
                                  end [« EXCHANGE »-simple-name]] S
        EXCHANGER-index  ::=    expression
9.6.    delay-ACTION     ::=    delay simple-expression S
9.7.    select-ACTION    ::=    selective-wait
                              / conditional-EXCHANGER-call
                              / timed-EXCHANGER-call
9.7.1.  selective-wait   ::=    select
                                  select-alternative
                                 {or
                                  select-alternative}
                                 [else BEHAVIOUR]
                                  end select S
        select-          ::=    EXCHANGE-alternative
        alternative             / delay-alternative
                                / terminate-alternative
        EXCHANGE-        ::=  EXCHANGE-ACTION [BEHAVIOUR]
        alternative
        delay-           ::=  delay-ACTION [BEHAVIOUR]
        alternative
9.7.2.  conditional-     ::=    select
        EXCHANGER-call           EXCHANGER-call-ACTION
                                [BEHAVIOUR]
                                else
                                 [BEHAVIOUR]
                                end select S
10.2.   body-stub        ::=    DEVICE-specification is separate ;
                              / SYSTEM'SYMBOL  body « SYSTEM »-simple-name  is separate ;
                              / PROCESS'SYMBOL body « PROCESS »-simple-name is separate ;
13.1.   representation-  ::=    FLUX-representation-clause
        clause                / address-clause
        FLUX-            ::=    lenght-clause
        representation-       / enumeration-clause
        clause                / record-representation-clause
```

IV.4 Metarules

```
ACTION           :    STATEMENT,GROUP_ITEM,LAST_GROUP_ITEM
BASIC_ORDER      :    [[PREPOSITION_PART] VERB_PART]
GROUP            :    STATEMENT,[{GROUP_ITEM} LAST_GROUP_ITEM]
ORDERED_ACTION   :    BASIC_ORDER,BASIC_ORDER (ORDERED_ACTION),BASIC_ORDER name
PREPOSITION_PART :    {PREPOSITION'SYMBOL [expression]}
S                :    STATEMENT_SEPARATOR,GROUP_ITEM_SEPARATOR,
                      LAST_GROUP_ITEM_SEPARATOR
VERB_PART        :    COMMAND'SYMBOL,TIME_OPERATOR'SYMBOL,OTHER_OPERATOR'SYMBOL
```

IV.5 Rules of meta-production

COMMAND'SYMBOL	:	**abort**	**initiate**,**run**
DEVICE	:	subprogram	device,subsystem
DEVICE'SYMBOL	:	**procedure**,**function**	**device**,**subsystem**,**component**, **binder**, **adapter**
EXCHANGE	:	accept	receive,send,connect
EXCHANGE'SYMBOL	:	**accept**	**receive**,**send**,**connect**
EXCHANGER	:	entry	receiver,sender,connector
EXCHANGER'SYMBOL	:	**entry**	**receiver**,**sender**,**connector**
FLUX	:	type	category,flux
FLUX'SYMBOL	:	**type**	**category**,**flux**
GROUP_ITEM	:		action
GROUP_ITEM_SEPARATOR	:		,
ITERATIVE	:	loop	repeat
ITERATIVE'SYMBOL	:	**loop**	**repeat**
LAST_ACTION	:	statement	action
LAST_GROUP_ITEM	:		action
LAST_GROUP_ITEM_SEPARATOR	:		;
MORPHOLOGICAL	:	declarative	morphological
OPERATOR'SYMBOL	:		**hold**,**sample**,**translate**
PREPOSITION'SYMBOL	:		**since**,**to**,**at**,**until**
PROCESS	:	task	fonctor,process
PROCESS'SYMBOL	:	task	**fonctor**,**process**
REERENCE'SYMBOL	:	**pragma**	**model**
REFERENCE	:	pragma	model
STATEMENT	:	statement	
STATEMENT_SEPARATOR	:	;	
SUBFLUX	:	subtype	subcategory,subflux
SUBFLUX'SYMBOL	:	**subtype**	**subcategory**,**subflux**
SYSTEM	:	package	system
SYSTEM'SYMBOL	:	**package**	**system**
TIME_OPERATOR'SYMBOL	:		**hold**,**sample**,**translate**

IV.6 Tutorial for expressing a behaviour

The notions developed in this section were presented at different states of progressions in some IFAC/IFIP workshops on real-time systems and published by the IFAC. A more complete and achieved presentation was first given in French, in the thesis of the author, and was after inserted in a report of a study group, submitted to the Commission of the European Communities. Most of the comments thereafter are extracted from this report.

IV.6.1 General conventions

First, a basic distinction between the notion of « behaviour » and that of « sequence of statements » must be done. Assume the following sequence written in Ada:

Example 40:
```
Measure:= Sample(Voltage);
accept Output (Measure: out Pulse);
```

This sequence is assumed to belong to the description of a concrete-process command expressed, say in the form of a control program; it defines two of the statements of this program, to be executed in the order in which they are written. Now, assume a quite similar expression in the description language Dad:

Example 41:
```
Measure:= sample Voltage ,
send Output (Measure: out Pulse);
```

Under the convention we propose, this text no longer belongs to the description of the concrete-process command but to the description of a group of possible actions of the concrete process itself . We describe in this example how the element `Measure` is emitted as it is produced. For this purpose we write, not two statements that are runable sequentially, but two relationships, simultaneously true, between the interactive elements of a system that exists continuously in time. We now present the essential conventions chosen for Dad, with the understanding that this treatment does not claim to be exhaustive.

(I) Any behaviour is a sequence of one or more different groups of actions ordered in time; their expressions are separated by semi-colons.
(II) Every group is a set of one or more concurrent actions; the expression of actions within a group are separated by commas.

Example 42
```
begin
 -- this is the first group of parallel actions:
 Measure(1):= sample Voltage(1),
 Measure(2):= sample Voltage(2),
 initiate Measures_Edition;
 -- This is the second group following in time the previous one:
 Measure(1):= sample Voltage(1),
 Measure(2):= sample Voltage(2),
end;
```

We will use the qualifiers « descendent » and « parent » in order to situate reciprocally the groups and actions which may be recursively embedded.

IV.6.2 Real-Time

Any element of a system, and in consequence the system as a whole, is intended to function: the « action » is the expression specifying its possible ways of functioning. Any element implies the existence of a first form of modelling, based on a substrate space (see definition: page 29) of which each element defines, up to a translation in time, a functioning of the represented element of the system. Specifying all the possible functionings of this element therefore consists of localising its first model in time by a command and possibly, of specifying any modification made by an operator to this model at the amount of localisation. An action belongs to a group and any group belongs to the behaviour of an element, itself activated by at least one action, except for the system as a whole.

A) Command, significant instant

The localisation in time of a group of actions in a behaviour or of an action in a group is specifying by a « command », sometime implicit. Once again, we use the first form of modelling (category) to specify its use.

First, let us consider a single action.

Example 43:
```
at T_Start run X;
```

X is a part of a system; X is represented by a category **C**. Each element of the space, subjacent to the definition of the category has a time domain; their reunion is the domain of X. Hence, any definition of such a category **C** implies the existence of a time axis θ intrinsic to that item and supporting the domain of X; let 0(θ) the origin of this axis. A command localises the category **C** in time; the result is a commanded action; then the command localises 0(θ) on the time axis T associated with the parent action of the commanded action; the instant t_c of localisation on this axis T is known as the instant of command. Progressing in this way, the domain of each activated part X of the system is situated on the time axis associated with the whole system which represents the absolute time. The lower bound of this domain is the instant of activation; the upper bound is the instant of inhibition. In the example 43, `T_Start` is the instant of command and the instant of activation; the instant of inhibition is generally the instant of the normal end of `X`. The instant of activation may be different of the instant of command like in the example 44 thereafter.

Let us now consider a group of actions activated with the same command. The instant of activation of the group is the lowest bound of the different associated domains and the instant of inhibition is the latest of their upper bounds.

The instant of activation and the instant of inhibition of a group of actions are the significant instants of this group.

B) Operator

The localisation in time is often accompanied by modifications of the model representing modifications of temporal properties of the part of the system; these modifications are specified by an operator. They may concerns the specification of the instants of activation and inhibition.

Example 44: `at T_Start run (since T_1 until T_2 sample X);`

C) Critical and free time-part of an action

This dependency among groups and actions, previously qualified as « descendent » or « parent », implies relationships between significant instants. The distinction between a parent entity and its descendent entities, together with the definition of the time relations among them, depends on the conventional meaning we attach to the terms used in their expressions: we examine these conventions in the paragraph that follows. We firs posit the following conventions:

(I) The intrinsic time axis of the system as a whole is chosen as the absolute axis; its origin is the absolute origin. This origin represents the instant of command for activating the entire system. We state that this command at the highest level is always implicit

(II) Let be a sequence of groups of actions embedded in a parent action. For the first group, the instant of command and of activation is the instant of activation of the parent action. For any other group the instant of command and of activation is the end of inhibition of the critical part of the preceding group.

(III) For all action produced by a single parent group, the instant of command, in the absence of any condition or of any specification on this instant, is the instant of activation of the group. The instants of activation and of inhibition depend of the conventional meaning we attach to the terms used in the action expression.

(IV) Any action has a portion that is critical for its parent group and which begins at the instant of command; its normal duration, in the absence of inhibition of the parent action by an abort, is the by the meaning of the terms used in its expression; the instant that is the latest of all the ends of the critical portions of actions having the same parent group sets the earliest instant of normal inhibition of that parent group. Note that the duration of this critical portion may be null.

(V) Any critical portion may be followed by a free portion. The instant at which the free portion begins is the instant at which the critical portion preceding it ends; the instant at which it normally ends, in the absence of inhibition of the parent action by abort, is set by the meaning of the terms used in its expression.

(VI) The inhibition of an action by abort implies the recursive inhibition of the descendent or derived groups and actions that would not yet normally inhibited. Any free portion is without effect upon the instant of normal inhibition of the parent group.

IV.6.3 Creation and modification of an active element

The purpose is to specify the production of new elements in the system and to active these new elements .

The « `assignment_ACTION` » rule is used for this purpose. The first member designates the element created or modified. The second member specifies a combination of elements that are already active and of which the significant instants are already determined.

IV.6.4 Simple action

The purpose of these expressions is to activate the elements of system by ordering them, and, possibly, to transform them.

These expressions are all derived from the rules of metaproduction proposed for the « `ORDERED_ACTION` » The expressions obtained by the use of this rule are extensions of Ada, except for the expression « `abort-statement` ». For each action, the significant instants and the bound between the critical and free portions result from the meaning attached by convention to the term used.

We distinguish basically two kinds of simple actions: the triggered action and the on/off action.

A) Triggered action

The expression of a triggered action uses basically the command « **`initiate`** » preceding the name of the element which is activated.

Example 45:
```
initiate Device;
```

The instant of activation is the instant of command. The instant of inhibition is the instant of normal end of functioning of the activated element or the instant of an abort concerning it. The triggered action is entirely free for its parent and for the following action in a sequence.

The operator initiate may be implicit. As well the expression « `Device` » is equivalent to the expression « **`initiate`** `Device` ».

Example 46:
```
initiate Device ,
initiate Other_Device;
```

Example 47:
```
initiate Device;
initiate Other_Device;
```

Let us note that in the examples 46 and 47 above, the instant of command of the second action is equal to the instant of command of the first action.

B) On/off action

The expression of a on/off action uses basically the command « **`run`** » preceding the name of the element which is activated.

Example 48:
```
run Device;
```

Example 49:
```
run Device;
run Other_Device;
```

Example 50:
```
run Device ,
run Other_Device;
```

The instant of activation is the instant of command. The instant of inhibition is the instant of normal end of functioning of the activated element or the instant of an abort concerning it. The on/off action is entirely critical for its parent and for the following action in a sequence.

In the example 49 the instant of command and of activation of the second action is the end of the preceding. In the example 50 the two actions are parallel and their respective instants of command and of activation are equal.

IV.6.5 Synchronisation of behaviours

A) Exchange action

The exchanges actions are used for that purpose. An exchange action may involves a derivative action. The instant of command (receive/send) is the instant of activation of the parent group. The instant of activation is the instant of command. The instant of inhibition is at the earlier of:

(I) the instant of inhibition of the parent group by abort,
(II) the latest end of the exchange of a maximum form for each element of the formal list.

The exchanger is required to recognise a form in time for each of the element exchanged defined in the formal lit. The formal list specifies, for each element exchanged, a category model; this model determines various possible forms for each element exchanged which are the elements of the substrate space. Each element of the substrate space has an associated time interval which is its domain. The element having a domain of which the upper bound is less the upper bound of all other domains is called a minimum form; correspondingly one can define a maximum form. For each exchanger action, the instant of activation of the derivative action is considered ion addition to the significant instant: this instant ends the minimum duration

needed to exchange a minimum form for each element of the formal list. In general, the starting time of an exchange and the instant which ends the exchange of a minimum differ from one element exchanged to another; we agree finally that the exchange of an element continues after the exchange of a minimum form, but ceases as soon as the exchange of its maximum form ends. The exchange of this element can then no longer be repeated without a new command.

The exchange action is totally critical for the parent group.

B) Derivative (do) action

The instant of command is the instant of derivative activation of the exchange action; the instant of activation is the same as the foregoing; the instant o inhibition is, at the earliest:

(I) the instant of inhibition by abort, of the parent action group of the exchanger group,
(II) the instant of inhibition of its last descendant group.

The derivative action is free for the exchange action and for the parent group of the exchanger action.

V EXAMPLE OF HIGH LEVEL DESCRIPTION: A WORK PLANT

V.1 General

This simple example was designed by the EWICS[42] during the years 80s in order to check the features of descriptions tools and methodologies used in industrial control.

In this example, each primary morphology introduces a list of descendent items which may be,

- either described in the same way, with the same language,
- or described differently.

The « use clause » is available in order to point at these description. This clause is not presented in this example.

V.2 Fisrst level of analyse

```
process Weighing_Mixing_System is
 receiver Feed         (x : in Power);
 receiver Start        (x : in Pulse);
 receiver Stop         (x : in Pulse);
 sender   issue        (x : out Produce);
end Weighing_Mixing_System;

process body Weighing_Mixing_System is
 process Tank_A;
 process Tank_B;
 process Store;
 process Weight_Unit;
 process Mixer;
 process Control;

 binder  Feeder
 binder  Status;
 binder  Command;
 binder  Network;
 binder  Fluid;
 binder  Solid;

begin

 receive     Feed   (Energy : in Power)
 do
  repeat
   receive   Start  (P : in Pulse)
   do
    initiate Feeder;
    initiate Tank_A, Tank_B, Store, Weight_Unit, Mixer, Control,
    initiate Status, Command, Network, Fluid, Solid;
   end       Start;
   receive   Stop   (P : in Pulse);
    abort    Tank_A, Tank_B,  Store,   Mixer, Control,
    abort    Status, Command, Network, Fluid, Solid;
    abort    Feeder;
   end       Stop;
  end        repeat;
 end         Feed;

end Weighing_Mixing_System;
```

42 European Workshop for Industrial Computer System. This Workshop was the European branch of the Purdue Workshop.

```
-- Analysis, 1st level, visible morphologies

process Tank_A is
 receiver Feed    (W : in  Power);
 sender   Status  (S : out Level);
 sender   Valve   (F : out Fluid_A);
 receiver Command (P : in  Raising_Edge);
end Tank_A;

process Tank_B is
 receiver Feed    (W : in  Power);
 sender   Status  (S : out Level);
 sender   Valve   (F : out Fluid_B);
 receiver Command (P : in  Raising_Edge);
end Tank_B;

process Store is
 receiver Feed    (W : in  Power);
 sender   Status  (S : out Level);
 sender   Gate    (B : out Bricks);
 receiver Command (P : in  Raising_Edge);
 sender   Network (D : out Data);
end Store;

process Weight_Unit is
 receiver  Feed            (W : in    Power);
 sender    Status          (S : out   Level);
 receiver  Before_Weighing (F : in    Fluid);
 sender    After_Weighing  (F : out   Fluid);
 connector Network         (D : inout Data);
 receiver  Command         (P : in    Raising_Edge);
end Weight_Unit;

process Mixer is
 receiver Feed       (W  : in  Power);
 sender   Status     (S  : out Level);
 sender   Up_Limit   (HL : out Level);
 sender   Down_Limit (DL : out Level);
 receiver All        (X  : in  Produce);
 sender   issue      (X  : out Produce);
 receiver Up         (P  : in  Raising Edge);
 receiver Down       (P  : in  Raising_Edge);
end Mixer;

process Control is
 receiver  Status(1..16) (S : in    Level);
 connector Network       (D : inout Data);
 sender    Command(1..16)(P : out   Pulse);
end Control;

binder  Feeder is
begin
Tank_A.Feed     :=Weighing_Mixing_System. Feed,
Tank_B.Feed     :=Weighing_Mixing_System. Feed,
Store.Feed      :=Weighing_Mixing_System. Feed,
Weight_Unit.Feed:=Weighing_Mixing_System. Feed,
Mixer.Feed      :=Weighing_Mixing_System. Feed,
end Feeder;

binder  Status is
Control.Status(1):=Tanker_A.Status,
Control.Status(2):=Tanker_B.Status,
Control.Status(3):=Store.Status,
Control.Status(4):=Weight_Unit.Status,
Control.Status(5):=Mixer.Status,
Control.Status(8):=Mixer.Up_Limit,
Control.Status(9):=Mixer.Down_Limit,
end Status;

binder  Command is
Tanker_A.Command     :=Control.Command(1),
Tanker_B.Command     :=Control.Command(2),
Store.Command        :=Control.Command(3),
Weight_Unit.Command  :=Control.Command(4),
Mixer.Command        :=Control.Command(5),
Mixer.Up             :=Control.Command(8),
Mixer.Down           :=Control.Command(9),
```

```
end Command;

binder  Network is
Control.Network :=Store.Network,
Control.Network :=Weight_Unit.Network,
end Network;

binder Pipes is
Weigh_t_Unit.Before_Weighing :=Tanker_A.Valve,
Weigh_t_Unit.Before_Weighing :=Tanker_B.Valve,
Mixer.All                    :=Weight_Unit.After_Weighing,
Weighing_Mixing_System.Issue :=Mixer.Issue,
end Pipes;

binder Belt is
Mixer.All :=Store.Gate,
end Belt;
```

V.3 Second level of analyse

```
process body Control is

flux High ;
flux Down ;
flux Bottom_Up ;
flux Top_Down ;
subflux Two is Data;
subflux A   is Data;
subflux AB  is Data;
P1, P2 : Pulse ;

begin

-- A check and a reset of all the devices must preceed the following phase.
-- The check and reset phase is not described in this simplified example.

repeat

 -- Tank_A.Valve opening
 -- Trigger for Tank_A.Valve opening :
 send Command(1)        (P   : out Pulse);

 -- Weighing Fluid_A
 receive   Status ( 1)  (s1  : in  Bottom_Up)
 do
  receive  Network      (x   : in  A) -- Waiting for Fluid_A as required
  do
   send    Command( 1)  (P   : out Pulse); -- Trigger for Tank_A.Valve closing
  end       Network;
 end       Status;
 receive Status( 1)     (s1  : in  Top_Down); -- Waiting for Tank_A closing

 -- Tank_B.Valve opening
 send      Command( 2)  (P   : out Pulse); -- Trigger for Tank_B.Valve opening

 -- Weighing Fluid_B
 receive   Status ( 2)  (s2  : in  Bottom_Up)
 do
  receive  Network      (x   : in  AB) -- Waiting for Fluid_B as required
  do
    send   Command( 2)  (P   : out Pulse); -- Trigger for Tank_B.Valve closing
  end       Network;
 end       Status;
 receive   Status( 2)   (s2  : in  Top_Down); -- Waiting for Tank_B closing

 -- Mixing Fluid_A and Fluid_B
 send      Command( 5)  (P   : out Pulse), -- Trigger for Mixer starting
 send      Command( 3)  (P   : out Pulse), -- Trigger
                                           -- for Weight_Unit
                                           -- after_Weighing opening
```

```
-- Adding two Bricks
 send      Command( 3)  (P   : out Pulse); -- Triger for Store.Gate opening
 receive   Network      (x   : in  Two)
 do
  send     Command( 3)  (P   : out Pulse); -- Trigger for Store.Gate closing
 end       Network;

 -- Mixing Fluids and Bricks
 begin
  P2 := translate(Timing) P1,
 send      Command( 5)  (P2  : out Pulse);
 end

 -- Mixer emptying
 send      Command(10)  (P   : out Pulse)
 receive   Status(10)   (s10 : in  Down)
 do
  send     Command( 9)  (P   : out Pulse);
 end       Status;
 receive   Status( 9)   (s09 : in  UP);
 end repeat;
end        Control;


process body Tank_A is

flux Power;
flux Level;
flux Fluid_A;
flux Raising_Edge;
subflux On  is Level;
subflux Off is Level;
S : Level;

begin

 receive  Feed   (W : in Power),
 S := Off,
 send     Status (S : out Level),
 repeat
  receive Valve  (X : in Raising_Edge)
  do
   S := On,
  end     Valve;
  receive Valve  (X : in Raising_Edge)
  do
   S := Off,
  end     Valve;
 end      repeat;
end Tank_A;
```

VI EXAMPLE OF LOW LEVEL DESCRIPTION: AN ANALOG TO DIGIT CONVERTER

VI.1 General

(to be completed later).

VI.2 Fisrst level of analyse

```
process       ADC is

 receiver     Voltage   (V     : in  Signal) ;
 receiver     Analog    (A     : in  Signal) ;
 receiver     Trigger   (Start  : in  Raising_Edge) ;
 sender       Result    (Word   : out Parallel_Levels) ;
 sender       Status    (Busy   : out Bivalent, Fault : out Bivalent) ;

end           ADC ;

process body  ADC is

 flux         Signal ;
 subflux      Level              is  Signal ;
 subflux      Pulse              is  Signal ;
 subflux      Low                is  Level;
 subflux      High               is  Level;
 subflux      Null               is  Level;
 subflux      Raising_Edge       is  Pulse;
 flux         Bivalent           is (Low, High) ;
 flux         Parallel_Levels    is  array (0..7) of Level ;

 receiver     Stop1    (Over   : in  Pulse) ;
 receiver     Stop2    (End    : in  Pulse) ;
 sender       Reset    (R      : in  Pulse) ;
 process      Compare ;
 process      DAC ;
 process      Counter ;
 process      Pulse_Generator ;
 process      Store ;
```

```
begin

 receive      Voltage  (V      : in  Voltage)
 do

 repeat
  receive     Trigger  (Start  : in  Pulse)
  do
   initiate   Compare, DAC, Counter, Pulse_Generator, Store,
   send       Status   (Busy   : out High, Fault : out Low) ;
              -- Busy without failing
   send       Reset    (R      : out Pulse) ;
   for I in (0..7)
   loop
    Word(I):= Low,
   end loop,
   send       Result   (Word   : out Parallel_Levels),
  end         Trigger ;

  receive     Analog   (A      : in  Signal),
  send        Result   (Word   : out Signal),
  select
   receive    Stop1    (Over   : in  Pulse) ;
   do
    send      Status   (Busy   : out High, Fault : out High) ;
              -- Busy and failing
    abort     DAC, Counter ;
   end        Stop1
  or
   receive    Stop2    (End    : in  Pulse)
   do
    send      Status   (Busy   : out Low,  Fault : out Low) ;
              -- Normal end
    send      Result   (Word   : out Parallel_Levels),
    abort     Compare, DAC, Counter, Pulse_Generator, Store ;
   end        Stop2 ;
  end         select ;
 end          repeat ;
end           Voltage ;

end           ADC ;
```

VI.3 Second level of analyse

```
process       Compare is

 connector    Plug(1..4) (S   : inout Signal) ;

end           Compare ;

process body  Compare is

 flux         Pulse ;

begin

 receive      Plug(1)    (Ss1 : in    Signal) -- receives the voltage supply
 do
  receive     Plug(2)    (Ss2 : in    Signal) -- receives the analog scale
  do
   receive    Plug(3)    (Ss3 : in    Signal) -- receives the analog signal
   do

    repeat
     Sp1 :=   sample Ss1,
     Sp2 :=   sample Ss2,
     Sp3 :=   sample Ss3,
     if
      Sp2>Sp3
     then
      send    Plug(4)    (Sp1 : out   Pulse) ;
              -- This pulse, sent to Stop2, indicates the normal end
     endif ;
    end       repeat ;
   end        Plug ;
  end         Plug ;
 end          Plug ;

end           Compare ;


process       DAC is

 receiver     Voltage     (V      : in  High) ;
 receiver     Digit_DAC   (Digit  : in  Parallel_Levels) ;
 sender       Analog_DAC  (Scale  : out Analog_Scale) ;

end           Dac ;


process body  DAC is

 subflux      Level             is  Signal ;
 subflux      Low               is  Level;
 subflux      High              is  Level;
 flux         Parallel_Levels   is  array (0..7) of Level ;

 subflux      Analog_Scale is Signal ;

begin

 receive      Voltage      (V     : in  High)
 do
  repeat
   receive    Digit_DAC    (Digit : in  Parallel_Levels)
   do
    send      Analog_DAC   (Scale : out Analog_Scale) ;
   end        Digit_DAC ;
  end         repeat ;
 end          Voltage ;

end           DAC ;
```

```
process      Counter is

 receiver    Voltage      (V        : in  Signal) ;
 receiver    Trigger      (Start    : in  Raising_Edge) ;
 receiver    Reset        (R        : in  Raising_Edge) ;
 receiver    Clock        (String   : in  Signal) ;
 sender      Binary_Value (Word     : out Parallel_Pulses) ;
 sender      Full_Scale   (Overflow : out Pulse) ;

end          Counter ;

process body Counter is

 flux        Pulse ;
 subflux     Raising_Edge    is Pulse ;
 flux        Level ;
 flux        Parallel_Pulses is array (0..8) of Pulse ;
 flux        Parallel_Levels is array (0..8) of Level ;
 sender      Reset_General        (Parallel_R       : out Parallel_Pulses) ;
 sender      Reset_Element(0..8)  (Single_R         : out Pulse) ;
 sender      Binary_Element(0..7) (Bit              : out Level) ;
 process     Flip_Flop (0..8) ;

 adapter     Share_Out ;
 adapter     Word_Integrator ;
 binder      Voltage_Supply ;
 binder      Reset ;
 binder      Binary_Count ;
 binder      Binary_Store ;

begin

 for I in (0..8)
 loop
  initiate  Flip_Flop(I) ,
  end loop ;
 receive    Voltage       (V           : in  High)
 do
  receive   Trigger       (Start       : in  Raising_Edge)
  do
   receive  Reset         (R           : in  Raising_Edge)
   do
    send    Reset_General (Parallel_R  : out Pulse) ;
    receive Clock         (String      : in  Signal),
    send    Binary_Value  (Word        : out Parallel_Pulses),
    send    Full_Scale    (Overflow    : out Pulse),
   end      Reset ;
  end       Trigger ;
 end        Voltage ;

end         Counter ;


process        Pulse_Generator is

 receiver      Voltage (V    : in High) ;
 sender        Clock (S : out Signal) ;

end            Pulse_Generator ;

process        body Pulse_Generator is

 subflux       Pulse_and_Delay      is Signal ;

begin

 receive       Voltage (V : in  High)
 do
  repeat
   S :=        Pulse_and_Delay,
   send        Clock   (S : out Signal) ;
  end          repeat
 end           Voltage ;

end            Pulse_Genarator ;
```

```
process            Store is

 receiver          Voltage      (V    : in  High) ;
 sender            Memory       (Word : out Parallel_Levels) ;
 receiver          Reset        (R    : in  Raising_Edge) ;

end                Store ;

process body       Store is

 subflux           Level            is  Signal ;
 subflux           Low              is  Level ;
 subflux           Bottom_Up        is  Signal ;
 subflux           Top_Down         is  Signal ;
 subflux           Nothing          is  Signal ;
 flux              Change           is (Bottom_Up, Top_Down, Nothing) ;
 subflux           Raising_Edge     is  Signal ;
 flux              Parallel_Levels  is  array (0..9) of Level ;
 flux              Parallel_Changes is  array (0..9) of Change ;

 receiver          Binary_Value (W    : in  Parallel_Changes) ;

begin

 receive           Voltage      (V    : in  High) ;

 do
  receive          Reset        (R    : in  Raising_Edge) ;
  do
   for I in (0..7)
   loop
    Word(I) :=     Low,
   end loop,
   send            Memory       (Word : out Parallel_Levels),
  end              Reset ;
  repeat
   receive         Binary_Value (W    : in  Parallel_Changes) ;
   do
    for I in (0..7)
    loop
     case W(I)
      when Top_Down =>
       Word(I) :=  Low,
      when Bottom_Up =>
       Word(I) :=  High,
      others =>
       nill,
     end           case
    end            loop
    send           Memory       (Word : out Parallel_Levels) ;
  end              repeat,
 end               Voltage
end                Store ;
```

VI.4 Third level of analyse

```
process         Flip_Flop is

 receiver       Voltage                (V    : in  High) ;
 receiver       Reset                  (R    : in  Raising_Edge) ;
 sender         Binary_Element (0..1)  (Bit  : out Pulse) ;
 receiver       Enter                  (Edge : in  Raising_Edge) ;

end             Flip_Flop ;

process body    Flip_Flop(0..8) is

 subflux        Pulse        is    Signal ;
 subflux        Raising_Edge is    Pulse ;
 subflux        Zero         is    Low ;
 subflux        One          is    High ;
 flux           Status       is   (Zero, One) ;
 Value:         Status,

begin

 receive        Voltage           (V : in High) ;
 do
  receive       Reset             (R : in Raising_Edge)
  do
   Value   :=   Zero,
   Bit0    :=   sample Zero,
   Bit1    :=   sample One,
   send         Binary_Element(0) (Bit0 : out Pulse),
   send         Binary_Element(1) (Bit1 : out Pulse),
  end           Reset ;
  repeat
   receive      Enter             (Edge : in Raising_Edge)
   do
    case        Value is
    when        Value=Zero =>
     Value :=   One,
     Bit0  :=   sample Zero,
     Bit1  :=   sample One,
     send       Binary_Element(0) (Bit0 : out Pulse),
     send       Binary_Element(1) (Bit1 : out Pulse),
    when        others     =>
     Value :=   Zero,
     Bit0  :=   sample One,
     Bit1  :=   sample Zero,
     send       Binary_Element(0) (Bit0 : out Pulse),
     send       Binary_Element(1) (Bit1 : out Pulse),
    end         case ;
   end          Enter ;
  end           repeat ;
 end            Voltage ;

end             Flip_Flop ;
```

```
-- Links within ADC

binder Card is
begin

 Compare.Plug(1)          :=ADC.Compare(1),
 Compare.Plug(2)          :=ADC.Analog,
 Compare.Plug(3)          :=DAC.Analog_DAC,
 DAC.Voltage              :=ADC.Voltage,
 Counter.Voltage          :=ADC.Voltage,
 Pulse_Generator.Voltage  :=ADC.Voltage,
 Store.Voltage            :=ADC.Voltage,
 Counter.Reset            :=ADC.Reset,
 Counter.Trigger          :=ADC.Trigger,
 Counter.Clock            :=Pulse_Generator.Clock,
 DAC.Digit_DAC            :=Counter.Binary_Value,
 Store.Digit_Store        :=Counter.Binary_Value,
 ADC.Stop1                :=Counter.Fullscal,
 ADC.Stop2                :=Compare.Plug(4),
 ADC.Result               :=Store.Result,

end Card ;

-- Adapters and links within Counter

adapter Share_Out is
 begin

  Reset_Element (0).Single_R :=Reset_General.Parallel_R(0),
  Reset_Element (1).Single_R :=Reset_General.Parallel_R(1),
  Reset_Element (2).Single_R :=Reset_General.Parallel_R(2),
  Reset_Element (3).Single_R :=Reset_General.Parallel_R(3),
  Reset_Element (4).Single_R :=Reset_General.Parallel_R(4),
  Reset_Element (5).Single_R :=Reset_General.Parallel_R(5),
  Reset_Element (6).Single_R :=Reset_General.Parallel_R(6),
  Reset_Element (7).Single_R :=Reset_General.Parallel_R(7),
  Reset_Element (8).Single_R :=Reset_General.Parallel_R(8),

end Share_Out ;

 adapter Word_Integrator is
 begin

  Binary_Value.Word(0) :=Binary_Element(0).Bit,
  Binary_Value.Word(1) :=Binary_Element(1).Bit,
  Binary_Value.Word(2) :=Binary_Element(2).Bit,
  Binary_Value.Word(3) :=Binary_Element(3).Bit,
  Binary_Value.Word(4) :=Binary_Element(4).Bit,
  Binary_Value.Word(5) :=Binary_Element(5).Bit,
  Binary_Value.Word(6) :=Binary_Element(6).Bit,
  Binary_Value.Word(7) :=Binary_Element(7).Bit,

 end Word_Integrator ;

 binder Voltage_Supply is
 begin

  Flip_Flop (0).Voltage:= Counter.Voltage(0),
  Flip_Flop (1).Voltage:= Counter.Voltage(1),
  Flip_Flop (2).Voltage:= Counter.Voltage(2),
  Flip_Flop (3).Voltage:= Counter.Voltage(3),
  Flip_Flop (4).Voltage:= Counter.Voltage(4),
  Flip_Flop (5).Voltage:= Counter.Voltage(5),
  Flip_Flop (6).Voltage:= Counter.Voltage(6),
  Flip_Flop (7).Voltage:= Counter.Voltage(7),
  Flip_Flop (8).Voltage:= Counter.Voltage(8),

 end Voltage_Supply ;
```

```
binder Reset is
begin

 Flip_Flop (0).Reset:= Counter.Reset_Element(0),
 Flip_Flop (1).Reset:= Counter.Reset_Element(1),
 Flip_Flop (2).Reset:= Counter.Reset_Element(2),
 Flip_Flop (3).Reset:= Counter.Reset_Element(3),
 Flip_Flop (4).Reset:= Counter.Reset_Element(4),
 Flip_Flop (5).Reset:= Counter.Reset_Element(5),
 Flip_Flop (6).Reset:= Counter.Reset_Element(6),
 Flip_Flop (7).Reset:= Counter.Reset_Element(7),
 Flip_Flop (8).Reset:= Counter.Reset_Element(8),

end Reset ;

binder Binary_Count is
begin

 Flip_Flop (0).Enter:= Counter.Clock,
 Flip_Flop (1).Enter:= Flip_Flop(0).Binary_Element(0),
 Flip_Flop (2).Enter:= Flip_Flop(1).Binary_Element(0),
 Flip_Flop (3).Enter:= Flip_Flop(2).Binary_Element(0),
 Flip_Flop (4).Enter:= Flip_Flop(3).Binary_Element(0),
 Flip_Flop (5).Enter:= Flip_Flop(4).Binary_Element(0),
 Flip_Flop (6).Enter:= Flip_Flop(5).Binary_Element(0),
 Flip_Flop (7).Enter:= Flip_Flop(6).Binary_Element(0),
 Flip_Flop (8).Enter:= Flip_Flop(7).Binary_Element(0),

end Local ;

binder Binary_Store is
begin

 Counter.Binary_Element (0):= Flip_Flop(0).Binary_Element(1),
 Counter.Binary_Element (1):= Flip_Flop(1).Binary_Element(1),
 Counter.Binary_Element (2):= Flip_Flop(2).Binary_Element(1),
 Counter.Binary_Element (3):= Flip_Flop(3).Binary_Element(1),
 Counter.Binary_Element (4):= Flip_Flop(4).Binary_Element(1),
 Counter.Binary_Element (5):= Flip_Flop(5).Binary_Element(1),
 Counter.Binary_Element (6):= Flip_Flop(6).Binary_Element(1),
 Counter.Binary_Element (7):= Flip_Flop(7).Binary_Element(1),
 Counter.Full_Scale        := Flip_Flop(8).Binary_Element(1),
 -- Counter overflow

end Binary_Store ;
```

VII BIBLIOGRAPHY

VII.1 General

i **Ayeb** (B. el). Toward Systematic Construction of Diagnogstic Systems for Large Industrial Plants: Methods, Languages, and Tools. *in:* IEEE Trans. on Knowledge and data Engineering. Oct. 1994, vol.6, n°5, ISSN 1041-4347. pp.698/712.

ii **Battiston** (E.), Cindio (F. de), Mauri (G.). OBJSA nets: a class of high-level nets having objects as domains. *in:* Advances in Petri nets. Springer-Verlag, Berlin, 1988. pp.20/43. ISBN: 3 540 50580 6.

iii **Belli** (F.). Model-based construction and implementation-oriented evaluation of complex systems. *in:* IFIP Transactions A, vol.A-36. IFIP TC8/WG8.5 Working Conference on System Engineering in Public Administration. 3-5 March 1993. pp.125/143. ISSN: 0926-5473.

iv **Berthomieu** (B.), Choquet (N.), Colin (N.), Loyer (B.), Martin (J.M.), Mauboussin (A.). Abstract data nets: combining Petri nets and abstract data types for high level specifications of distributed systems. *in:* Seventh European Workshop on Application and Theory of Petri Nets. Proceedings. Sheffields City Polytech., Sheffield, UK, 1986. p456.

v **Beth** (t). Algebraic system modelling and Implementation. *in:* Computer Aided System Theory. EUROCAST'91. Second International Workshop. Proceedings. Springer-Verlag, Berlin, 1992. pp.21/31. ISBN: 3 540 55354 1.

vi **Conception** (Arturo I.), Zeigler (Bernard P.) DEVS Formalism: A Framework for Hierarchical Model Development. *in:* IEEE Trans. on Soft. Eng., Vol.14, n°2, febr. 1988. pp.228/241.

vii **Dutch National Body**. A Framework of Information System Concepts. Working paper. ISO/IEC TC21 WG3 CSMF. Ottawa meeting, July 1995. (OTT-26).

viii **Ehrich** (H.-D.). On the theory of Specification, Implementation, and Parametrization of Abstract Data Types. *in:* J. of the Assoc. for Comp. Mach., vol.29, n°1, jan. 1982. pp.206/227.

ix **Ehrich** (Hans Dieter). *Algebraische Spezifikation abstrakter Datentypen (Algebraic specification od abstract data types). Leitfaden und Monographien der Informatik. B.G. Teubner, Stuttgart, 1989. xp., 236p. ISBN: 3-519-02266-4*.

x **Ehrich** (Hans Dieter). From data types to object types. 41th Workshop On Mathematical Aspects of Computer Sciences, Magdeburg, 1988. *in:* Journal of Information Processing and Cybernetics, n°1-2. ISSN: 0863-0593.

xi **Ehrich** (Hans Dieter). Objects, Object types, and object identification. Categorical methods in computer science. pp142/156. Berlin, 1988. Springer, Berlin, 1989.

xii **Ehrich** (H. D.), Goguen (J.A.). A categorical theory of objects as observed processes. Lecture Notes in Computer Science. Springer, Berlin, 1991. pp.203/208. ISBN: 3-519-02266-4.

xiii Encyclopedia-1993**Encyclopedia** of **Computer Science**. 3d edition. Editors: Anthony Ralston, Edwin D. Reilly. Van Nostrand Reinhold, New York, 1993. ISBN 0-442-27679-6. xxivp., 1558p. (maj.: 930207).

xiv **Gibbs** (W. Wayt). Software's Chronic Chisis. *in:* Scientific American, sept. 1994, vol 271, n°3. pp.72/81. ISSN 0036-8733.

xv **Goguen** (Joseph A.). Sheaf semantics for concurrent interacting objects. International Conference on Symbolic Computation, Zurich, 1990. *in:* Mahtematical Structures in Computer Science, n°2, 1992. ISSN: 0960-1295.

xvi **Goguen** (Joseph A.). Sheaf semantics for concurrent interacting objects. International Conference on Symbolic Computation, Zurich, 1990. *in:* Mahtematical Structures in Computer Science, n°2, 1992. ISSN: 0960-1295.

xvii Davis (N.), Dacker (B.), **Goldsack** (S.J.), Halling (H.), Jarray (J.), Ludewig (J.), McGuettrick (A.), Page (J.), Pyle(I.), Savoysky (S.). Ada for specification: possiblities and limitations. The Ada Companion Series. Ed. by S.J. Goldsack. Cambridge Univ. Press, Cambridge, 1985. ISBN 0-521-30853-4. xvi p., 265p.

xviii **Gomm** (D.), Walther (R.). The distributed termination problem: formal solution and correctness based on Petri nets. *in:* Aspects and Prospects of Theoretical Computer Science. 6th International Meeting of Young Computer Scientists. Proceedings. Springer-Verlag, Berlin, 1990. pp159/168. ISBN: 3 540 53414 8.

xix Category Theory Applied to Computation and Control. Edited by G. **Goos** and J. Hartmanis. Lecture Notes in Computer Science. Springer-Verlag, Berlin, …, 1975. 246p. ISBN 3-540-07142-3.

xx **Haurat** (A.), Piard (F.). The « OLYMPIOS » model: an algebraic specification for modelling the information system of a manufacturing enterprise. *in:* 1993 CompEuro Proceedings. Computer in Design, Manufacturing, and Production. IEEE, Comp. Soc. Press., Los Alamitos, 1993. pp.330/335. ISBN: 0 8186 4030 8.

xxi **Hotaka** (Ryosuke), Björn (Michael). Data oriented Approach to Business Information Modelling. Univ. of Tsukuba, 1993. 19p.

xxii **Jensen** (K.). An introduction to the theoretical aspects of coloured Petri nets. *in:* Decade of Concurrency. Reflections and Perspectives. REX School/Symposium Proceedings. 1-4 June 1993. Springer-Verlag, Berlin, 1994. pp230/272. ISBN 3 540 58043 3.

xxiii **Korff** (M.). Single pushout transformation of equationally defined graph structures with applications to actor systems. In: Graph Transformations in Computer Science. International Workshop Proceedings. 4-8 Jan. 1993. Springer-Verlag, Berlin, 1994. pp.234/247. ISBN: 3 54057787 4.

xxiv **Lahdelma** (R.). An object-oriented mathematical modelling system.. *in:* Acta Polytechnica Scandinavica, Mathematics and Computer Science Series, n°MA66. pp.1/77. ISSN: 0355-2713.

xxv **Lieberherr** (Karl, J.), Xiao (Cun). Object-Oriented Software Evolution. *in:* IEEE transactions on software engineering. April 1993, vol.19, n°4. (ISSN 0098-5589). pp.313/343.

xxvi **McLane** (S.), **Birkhoff** (G.). *Algèbre. Tome I. Structures fondamentales. Traduit de l'américain par J. Weil. Préface de J. Dieudonné. Nouveau Tirage. Cahiers Scientifiques. Publiés Sous la Direction De M. Gaston Julia. Fascicule XXXV. Gauthier-Villars, Paris, 1971. xxivp., 410p.*

xxvii **McLane** (S.), **Birkhoff** (G.). *Algèbre. Tome II. Les grands théorèmes. Traduit de l'américain par J. Weil. Préface de J. Dieudonné. Nouveau Tirage. Cahiers Scientifiques. Publiés Sous la Direction De M. Gaston Julia. Fascicule XXXVI. Gauthier-Villars, Paris, 1971. xivp., 344p.*

xxviii **Najm** (E.), **Budkowski** (S.), **Gilot** (T.), **Lumbroso** (L.). General presentation of SCAN. A distributed system modelling and validation tool. Agence de l'Informatique. Paris la Défense, France. North-Holland, Amsterdam, 1986. xip., 544p. ISBN: 0 444 87881 5.

xxix **Noaks** (D. R.), **Wood** (K.). System modelling for safety and fault analysis using the software tool NP-Circuit. *in:* IEE Colloquium on « Structured Methods for Hardware Systems ». Digest N°1994/110. IEE, London, 1994.

xxx **Phoa** (W.), **Fourman** (M.). A proposal categorical semantics for pure ML. *in:* Automata, Languages and Programming. 19th International Colloquium Proceedings. Springer-Verlag, Berlin, 1992. pp.533/544.

xxxi **Rattray** (C.). The shape of complex systems. Computer Aided System Theory. EUROCAST'93. Third International Workshop on Computer Aided System Theory Proceedings; 22-26 Feb. 1993. Springer-Verlag, Berlin, 1994. pp.72/82. ISBN: 3 540 57601 0.

xxxii **Reggio** (G.). Event logic for specifying dynamic data types. *in:* Recent Trends in Data Type Specification. 8th Workshop on Specification on Abstract Data Types joint with 3rd COMPASS Workshop. Selected papers. Springer Verlag, Berlin, 1993. pp. 292/309. ISBN: 3 540 556379 2.

xxxiii **Savoysky** (S.). *Description sommaire d'une méthode d'utilisation de l'algèbre multilinéaire pour la représentation de systèmes. in: Réflexions sur de nouvelles approches dans l'étude des systèmes. Actes du Coll. org. à Paris, les 10, 11, et 12 juin 1975, par l'Ec. centr. des arts et man. et l'Ec. nat. sup. de techn. av.,… Association nationale de la recherche technique, Paris, 1975. pp. 177/200.*

xxxiv **Savoysky** (S.). Analysis and description of automatic control system. *in:* Real Time Programming 1980. Proc. of the IFAC/IFIP Workshop. Schloss Retzhof, Leibnitz, Austria, 14-16 April 1980. Ed. by V.H. Haase. Pergamon Press, Oxford,…, 1980. ISBN 0 08 027305 X. pp. 45/55

xxxv **Savoysky** (S.). The Use of Ada for the Specification of Automata in Civil Engineering. in: Real Time Programming 1981. Proc. of the IFAC/IFIP Work. Kyoto, Japan, 31 August - 2 Sept. 1981. Pergamon Press, Oxford,…, 1982. ISBN 0-08-027613-X. pp. 129/138.

xxxvi **Savoysky** (S.). Specification of exchange mechanisms between elements of industrial systems. *in:* IECON'85, Proceedings, Industrial applications of mini, micro, and personnal computers. San Francisco, Ca., nov. 18-22, 1985., vol.2. IEEE, sl., 1985. IEEE Cat. Numb. 85CH2160-0. Libr. of Congr. Cat. Numb. 85-60211. pp. 794/799.

xxxvii **Sowa** (John F.). Knowledge Representation. Logical, Philosophical, ad Computational Foundations. July 17th, 1995. Draft of a book in preparation.423p.

xxxviii **Thom** (René). *La théorie des catastrophes. in: Modèles mathématiques de la morphogenèse. Chr. Bourgoi Ed., Paris, 1908. pp.81/90.*

xxxix **Tucker** (J.V.), **ZUCKER** (J.I.). Toward a general theory of computation and specification over abstract data types. *in:* Advances in Computing and Information. ICCI'90. International Conference Proceedings. Springer-Verlag, Berlin, 1990. pp.129/133. ISBN 3 540 53504 7.

xl **Tuzhilin** (A.A.). Category theory of structural sets with application to mathematical modelling and systems analysis. *in:* Mathematical Modelling, vol.7, n°1. pp.27/48. ISSN: 0270-0255.

xli **Vries** (J.A. de), **Breedveld** (P.C.), **Meindertsma** (P.). Polymorphic modelling of engineering systems. *in:* International Conference on Bond Graph Modelling ICBGM'93. 1993 Western Simulation Multiconference. SCS, San Diego (CA), 1993. pp.17/22.

xlii **Zhao Xudong**, **Feng Yulin**. Automatic and hierarchical verification for concurrent systems. *in:* Journal of Computer Science and Technology (E), vol.5, n°3, pp.241/249. China, 1990. ISSN: 1000 9000.

VII.2 CSMF, working drafts

CSMF ABQ-010 US Contribution on Abstract Conceptual Schema Language (ACSL) Syntax and Semantics [SC21/WG3 N1781 Clause 8.2, revised Clause 8.1].

CSMF ABQ-015 **OEII** (J.L.H.). A Meta Model Transformation approach towards harmonisation in information systems modelling. Nov. 2, 1994.

CSMF ABQ-024 **WIELINGA** (Bob) and *al*. Framework and Formalism for Expressing Ontologies. KACTUS. Esprit Project 8145. Univ. of Amsterdam, 1994.

CSMF AIX-007 ⇒ ISO/IEC TC97/SC5/WG3

CSMF AIX-008 ⇒ ISO/TC 184/SC4 DIS 10303-11

CSMF AIX-009 ⇒ ISO/IEC JTC1/SC21

CSMF SOU-007 A Data Modelling Facility: JDMF/MODEL-1992 R1.1.

VII.3 Normative documents

AFNOR CGTI/CN21 F2236 **AFNOR CGTI/CN21 F2236**. A direct computational language semantics for Part 4 of the RM-ODP. AFNOR, Paris, 1994/07/05. 12p.

ANSI/MIL-STD 1815 A **ANSI.** Reference manual for the Ada® programming language. Alsys, La Celle-Saint-Cloud, 1983.

BSI DD 210: 1992 **BSI**. Guide to A framework for user requirements for Information Technology. Draft for Development. DD 210: 1992. 23p.

EWOS/ETG 012 **EWOS**. Guide to profiles for the open system environement. Approved by EWOS/TA (1991)..

IRDS part 2 93-196 **IRDS**. Conceptual Schema. Part 2: Modelling Language Analysis. X3H4/93-196. xp., 98p.

IRDS part 1 93-196 **IRDS**. Conceptual Schema. Part 1: Conceptual Schema for IRDS. X3H4/936196. xp., 160p.

ISO/IEC TC97/SC5/WG3 9007 **ISO**. Concepts and terminology for the conceptual schema and the information base. ISO TR 9007.

ISO TC184/SC4/ WG3 N103 **ISO TC184/SC4/WG3**. Technical Report on the Semantic Unification Meta-Model. Volume 1: Semantic Unification of Static Models. A technical report of the Dictionary : Methodology Committee of the IGES/PDE Organization on the application of formal semantics and symbolic logic to the integration and unification of models. Prepared by the Dictionary / Methodology Committee of the IGES / PDES Organization. Version 1.0. October 19, 1992. N103. 124p., Appendix,

ISO TC184/SC5/ WG4 N116 **ISO TC184/SC5/WG4**. Industrial Atomation Systems - MAPLE Architecture. Draft of Committee Draft. for DIS Balloting. May 1995.

ISO/IEC IS 10746-2 **ISO/IEC International Standard 10746-2**. Information Technology. Open Distributed Processing. Reference Model. - Part 2: Foundations. [IUT Recommendation X.902].

ISO/IEC IS 10746-3 **ISO/IEC International Standard 10746-3**. Information Technology. Open Distributed Processing. Reference Model. - Part 3: Architecture. [IUT Recommendation X.902].Draft.

ISO/IEC JTC1 SC22 N 1712 **ISO/IEC JTC1 TC 22**. N1712. Draft Technical Report for EXTensions for Real-time Ada. Work Item Number: JTC 1.22.35. ISO/IEC TR 11735. October 30, 1994. 69p.

ISO/IEC JTC1 TSG 1 **ISO/IEC JTC1 TSG 1**. Standards necessary to define Interfaces for Application Portability. (IAP). Final report. April 1993. 69p.

ISO/IEC JTC1 N 3095R **ISO/IEC JTC1**. Strategic plan for information technology standardisation. A Presentation of the Strategic Objective, Assumptions, Trnds and Needs. SPRG N31. 1 November1994. 4p.

ISO/IEC JTC1/SC21 236 **ISO/IEC JTC1/SC21**. Information Retrieval, Transfer and Management for OSI. N236.

ISO/IEC JTC1/SC21 8218 **ISO/IEC JTC1/SC21**. Information Technology. Open Distributed Processing. Reference Model .- Part 1: Overview and Guide to Use. N8218.

ISO/IEC JTC1/SC21 8913 **ISO/IEC JTC1/SC21**. LOTOS. A formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Amd 1: G-LOTOS. N8913. 118p.

ISO/IEC JTC1/SC21 9563 **ISO/IEC JTC1/SC21**. Information Technology. Open Distributed Processing. Reference Model. - Part 4: Architectural Semantics. N9563.

ISO/IEC JTC1/SC21/ WG1 **ISO/IEC JTC1/SC21/WG1**. Enhancements to LOTOS. Working Draft on Enhancements to LOTOS. Project 1.21.20.2.3. October 1994.. N1349. Pag. mult.

ISO/IEC JTC1/SC21/ WG3 1865 **ISO/IEC JTC1/SC21/WG3**. Conceptual Graphs, A presnetation Language for Knowledge in Conceptual Models.. WD of Proposed American National Standard. X3 Project N°. 1059-D. May 10, 1995. N1865.

ISO/IEC JTC1/SC21/ WG3 1866 **ISO/IEC JTC1/SC21/WG3**. Knowledge Interchange Format Reference Manual.. WD for an ANS. N1866.

ISO/IEC JTC1/SC7/ WG11 BRI22 **ISO/IEC JTC1/SC7/WG11. FULTON** (Dr. James A.) Strategy for the Integration of Knowledge-Based Engineering Data. Boeing Information and Support Services Research and Technology. Working Draft 1.2, June 19, 1995. 22p. ISO TC184/SC4/WG5 N234, ISO/IEC JTC1/SC7/WG11 BRI22. (OTT20).

ISO/TC 184/SC4 DIS 10303-11 **ISO/TC184/SC4**. Industrial automation systems - Product data representation and exchange - Part 11: Description method: The EXPRESS language reference manual. XIVp., 110p. ISO/DIS 10303-11.

SC2 1 N7054 **SC21**. Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model.

BIOGRAPHY

Born in 1933, Serge Savoysky earned his « *Licence ès Science* » in 1956 and he received his State Engineer degree in 1957 from the « *Ecole Nationale Supérieure de Mécanique et d'Aérotechnique* » ; he received later his French State « *Doctorat ès Sciences, Mathématiques* » from the « *Université Pierre et Marie Curie, Paris* » in 1983. He first entered as engineer in the Flying Trying Centre of the French Air Forces in 1957. During his military duty, he continued to serve as officer in these Air Forces (1957/1960), in Algeria. In 1960, he entered a Building Design Office again as engineer and he introduced in this company the data processing technologies. In 1970 he became the first head of Data Processing Department in the *« Laboratoire Central des Ponts et Chausées* », main research centre of the French State Roads and Bridges Laboratories Network. Since this time until 1992, he managed data processing in this network, mainly for laboratory and worksite process control, and conducted research in this domain where he met the subject of his doctorate thesis : Proposal of Theoretical Element for System Description; his personal main interest was the formal description of industrial systems. During the same time, he taught process control technologies in « *Ecole Nationale des Travaux Publics de l'Etat* ». Now retired, he continues to investigate in the field of formal system description methodologies. He enjoys collecting calculating machines and apparatus and books of the past. He is married and has two adult children.