In the Journal "Current Issues In Electronic Modeling" (CIEM)
Vol 4- "High-level System Modeling: Specification and Design Methodologies"
Kluwer Academic Publishers, Editors: J.M. Bergé, O. Levia, J. Rouillard

A System Specification Model and Method

Jean Paul Calvez

IRESTE
La Chantrerie CP 3003 44087 Nantes cedex 03 France
fax: (33).40.68.30.66 Email: jcalvez@ireste.fr

Abstract

This paper describes a system-level specification model and its associate method which is part of the MCSE methodology. The specification step of the system development life cycle is useful to formally express what the system must do. The resulting specification document includes functional and non-functional specifications.

After an overview of the whole development process, the meaning and objective of a specification is explained. Among many specification models and methods, our proposition is located between functional analysis methods such as SART and object-oriented analysis methods such as OMT. The model for functional specification requires three complementary views: object/ data modeling, behavior modeling, activity modeling. Regarding non-functional requirements, we explain the specification of performance constraints and link them with the functional modeling.

The specification process is explained in depth with an emphasis on functional versus physical analysis, the different approaches to express functional specifications, and the importance of selecting appropriate modeling viewpoints. The activity diagram is shown as a result of synthesis more than as a decomposition technique. An example of the embedded Hw/Sw CoDesign problem is completely detailed to illustrate both the specification model and the recommended method. It clearly shows the difference between modeling at the functional level and at the physical level. We also discuss specific features of the model and the benefits of our method.

1 Introduction

In the development life cycle of real-time systems, steps mark the progression from the initial concept to the operational status. During the first step, called the *definition step,* the features of the system are defined from virtually nothing. The first document, which concerns *user requirements*, describes the functions the system will perform, the constraints it will respect, the characteristics of the environment and its interfaces with the system, an many non-functional requirements such as size, power consumption, performance, reliability, safety, cost, deadline, etc. The *specification step* is then useful to formally express what the system must do. The following steps - *Design and implementation steps* - lead to a final product. The solution to a given problem is far from unique. The diversity of techniques allows designers to choose a solution located between a purely hardware solution and a software one. Performance, cost, deadline, time-to-market and quality criteria seriously restrict the range of possibilities. Market competition requires better quality work that involves shorter design time in order to increase the quantity and complexity of developments. As the technology exists, a major effort must concern the reduction of the development cycle time.

Increasingly sophisticated tools are being developed to describe and validate specifications and designs at the highest level as early as possible. Synthesis tools are also becoming a reality to automatically generate a correct solution by construction. Currently, these tools solve only a small part of a system-level problem [16], and even so, some forms of description are to be elaborated as input to tools. Thus, a large part of specification and design work must still be carried out by system designers.

To make the designer's work easier and more efficient, a methodological process must be a precise guide which clearly specifies the sequence of methods to be followed, in addition to the most appropriate solution description models. Here we are specifically interested in developing the process to follow during the specification step. The objective of this step is to translate requirements into a well-defined specification useful for design.

First we consider that requirements and specifications are fundamentally different types of descriptions. *Requirements* are a description of something wanted or needed, a set of needed properties, whereas a *specification* is a description which has those properties [1], [2]. *Requirements definition* is the process of understanding what the needs of all interested parties are and documenting these needs as written definitions and descriptions. The focus is on *what problem* the system has to solve. The primary step is focussing on the world in which the system will operate, not on the system itself. There are no formal languages currently available which are able to fully express requirements. Expertise natural language is used because the expression of requirements forces the discussion of arbitrarily complex problems in many fields of knowledge and with many partners [17], [23].

Requirement specification is a more formal process of translating the description of needs into a more formal structure and model. We consider a *specification* to be a precise description of the system itself which meets its requirements. As such, it should be as independent of the users and of the expected environment as possible. Ideally, a specification document should be complete, consistent, comprehensible, traceable to the requirements, unambiguous, modifiable, writable!!. It should be expressed in as formal a language [33] or notation [11], [16], [15] as possible. A specification should focus precisely on the system itself and provide a complete description of its externally knowable characteristics. The meaning of externally knowable clearly separates those aspects which are functionally visible to the environment in which the system operates from those aspects of the system which reflects its internal structure. Not concerned with the internal organization, a specification is supposed to describe *what* a system must do and not *how* it does it. So far, research in system specification has concentrated on language capabilities and graphical notations necessary to express and verify specifications. Many specification models and methods have been proposed in the available literature. Some are useful for hardware such as ASICs [16], [6], some are appropriate for software [10], [11], [14], [8], [24], [25], [30], and still others are more appropriate to the system level [5], [15], [20], [21], [34].

This paper concerns the specification of real-time hardware/software systems [5], [35]. This development step is becoming a very important issue in system engineering because errors found late during product development are mainly due to a poor understanding of the requirements. To avoid these errors, one would like to remove any ambiguities, inconsistencies and incompleteness in system requirements as early as possible. Going from user requirements to a specification document is not as easy as it looks. To be appropriate and efficient, a recommended specification method must satisfy some properties. The first point is that the specification process has to be based on a model or several inter-related models useful to hierarchically describe the system specification and verify it. The second point is that the specification must be described in a manner simple enough to enable comprehension, analysis and validation. This requires simple description

models, if possible graphical and formal models. The third point is that the specification process must guide analysts and specifiers to find, in as linear a manner as possible, an appropriate specification to correctly represent all the user requirements. The fourth point is that in order to be widely applicable, a specification model and method must not be restricted to some specific classes of systems, nor limited to systems of small or medium complexity. Thus, an appropriate method must allow the specifier to ask himself the right questions for his problem and then efficiently find the appropriate answers.

The proposed answer for the specification step is part of the MCSE methodology (Methodology for Electronic Systems Design) [5] the author has already developed. MCSE is appropriate for the specification, design and implementation of real-time embedded systems.

The contents of this paper is as follows: Section 2 gives an overview of the whole development process. The meaning and the objective of a specification is explained in Section 3. Among many specification models and methods recalled in Section 4, our proposition is located between functional analysis methods such as SART and object-oriented analysis methods such as OMT. To further detail the specification process, Section 5 presents essential modeling concepts and Section 6 describes the recommended model for functional specification which requires three complementary views: object/data modeling (Section 7), activity modeling (Section 8), behavior modeling (Section 9). Regarding non-functional requirements, we explain the specification of performance constraints in Section 10 and technological constraints in Section 11.

The specification process is explained in Section 12 with the emphasis on expressing functional-level specifications and the importance of selecting appropriate modeling viewpoints in Section 13. An example of embedded Hw/Sw CoDesign problem is completely detailed in Section 14 to illustrate both the specification model and the recommended method. In Section 15 we discuss specific features of the model and the benefits of our method. Conclusions are stated in the last section.

2 Overview of the Development Process

For efficiency and quality reasons, every design must be done according to a methodological process based on a hierarchy of description models ranging from the preliminary idea to the final product [28]. The specification model and the corresponding method described hereby are a part of the complete system development process based on the so-called MCSE methodology specifically conceived to design embedded real-time systems [5], and which has been enhanced to support ASIC design [6]. With MCSE, system designers must proceed according to four steps: system specification, functional design, implementation specification, realization.

The purpose of the first *specification step* is to express a purely external view of the system, (WHAT) starting from needs and user requirements. This step requires first a tight knowledge of the system's environment and then a complete modeling of the system behavior. The initial model is a functional specification which can be based on data-flow diagrams (SA and SART methods), FSMs, StateCharts, SDL, Lotos, etc., depending on the nature and the complexity of the problem. Non-functional specifications have to be added to explain constraints such as performance and timing constraints, dependability constraints, cost, implementation and manufacturing constraints, etc.

The purpose of the *functional design step* is to find an appropriate internal architecture (which explains the HOW), but one according to an application-oriented viewpoint. The description based on a functional structure and the behavior of each function has to be technology-independent. The designer uses the functional specifications as an entry point for this step. A first functional

decomposition is carried out based on the search of essential internal variables and events of the system. The design process then consists of successive refinements for each function according to the same process until elementary functions are obtained. These are functions the behavior of which can be expressed by a purely sequential description. The functional description model is appropriate and efficient in order to verify the design quality and to evaluate the system behavior and its performance. For example, VHDL used at a system and behavioral level is one means to validate such an analysis.

The third step called the *implementation specification step*, consists in searching, firstly for the executive support or hardware architecture and, secondly, for the organization of the software on each processor. First, the functional description must be enhanced and detailed to take into account the technological constraints: geographic distribution (if necessary), physical and user interfaces. Timing constraints and performance are then analyzed to determine the hardware/software partitioning. The hardware part is specified by an executive structure. Integration, which includes function allocation, completely describes the implementation of the functional description onto the executive structure. During this step, it is easy to extract the sub-system which concerns the CoDesign process; the rest is more the concern of a software development process. Partitioning between hardware and software is included in this step.

The *implementation step* leads to an operational system. Implementation, which includes testing, debugging and validation, is a bottom-up process since it consists of assembling individual parts, bringing out more and more abstract functionalities. Each level of the implementation is validated by checking for compliance with the specifications of the corresponding level in the top-down design process. Hardware and software implementations are developed simultaneously involving specialists in both domains, thus reducing the total implementation time.

Structuring specifications implies considering the use of each part of them during the next design steps. The design process briefly described above, follows 3 steps:

- a *functional approach,* to identify internal functions and the relations between them which are necessary to describe the system,
- an *operational approach,* to explain the behavior of all internal functions and actions on the system environment and to allocate non-functional requirements such as performances and dependability,
- a *technological approach,* to define the hardware architecture and the allocation of the functional solution on it, based on all technological constraints.

Therefore, to be efficient during the design step, it is interesting to classify all requirements in the specification document in 3 categories according to the functional, operational and technological aspects. Beforehand, the system has to be correctly embedded in its environment. Therefore, the specification process must include the following phases:

- analysis and modeling of the system environment,
- specification of all inputs and outputs of the system,
- description of functionalities and corresponding behaviors of the system,
- description of all non-functional specifications corresponding to operational and technological constraints.

The analysis of the system environment leads to a correct definition of the complete application and all the entities concerned by the system. Modeling concerns every entity and the dependencies between them. The system is defined by its inputs and outputs, its functionalities for the application, its complete and detailed behavior according to an external viewpoint, and all non-functional constraints.

3 Objective, Nature and Quality of a Specification

Based on user requirements, the customer and designers have to carry out initial work together in order to clearly express the objective to be achieved. *Specifications are intermediate between the need and the design*. They are useful to both partners: first to the customer, because he can be sure that his problem has been correctly understood by the designers and that the resulting product will comply with his requirements, and second, to the designers, because in getting a precise idea of the problem, they can efficiently undertake the design and then the implementation work. A better estimate of the development cost is also possible.

To formalize the understanding of the problem and the objective to be achieved, an intermediate document is necessary between the requirements expressing the WHY and the solution developed by the designers expressing the HOW. This intermediate description, called a *specification* (the WHAT), bridges the gap between the customer's partners and designers, and allows a formal verification by the customer of the designers' understanding of the problem. It is used to avoid the fact that what is not specified cannot be verified, and what is not verified may be wrong.

A *specification document* includes functional specifications and non-functional specifications. As a definition, a *functional specification* describes the complete external characteristics of the system to be designed which operates in the environment explained in the requirement document. *Non-functional specifications* are restrictions or constraints imposed to designers. They can be classified into product constraints, process constraints and external constraints. The following list gives an idea of the diversity of non-functional requirements: performance, usability, installation, interoperability, design portability, extensibility, physical and electrical constraints, environment constraints, reliability, safety, security, availability, maintainability, cost, risks, deadline, test and validation procedures, certification, etc.

Concerning the functional specification, the system to be designed is never isolated from the rest of the world. It is necessarily embedded into an environment including objects or entities. Figure 1 shows the normal situation for all systems.

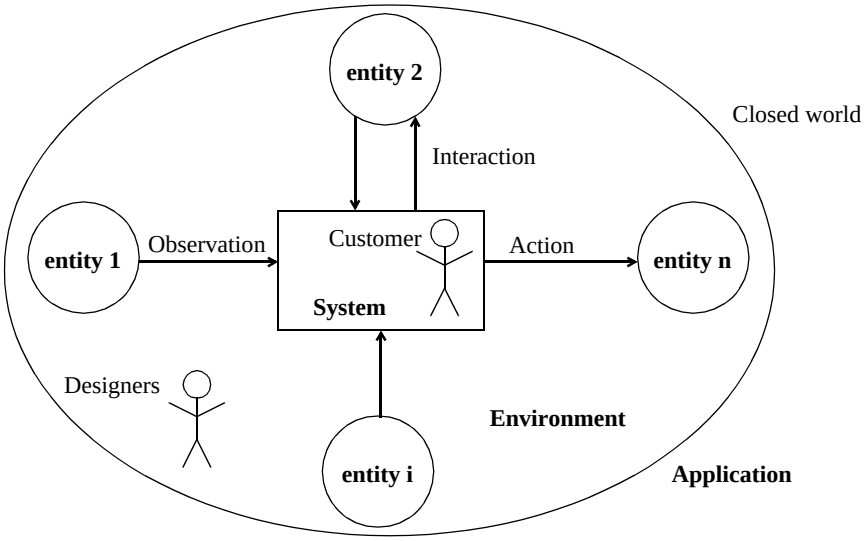


Fig. 1. The system to be specified and its environment.

An *application* in its physical sense is considered here as the closed world having no useful relation with its outside for the problem to be solved. The *environment* is the all application *objects* excluding the system to be developed. These objects, also called *entities*, have their own functionality described by their behaviors and correspond to a functional reality but not necessarily to a physical one. It can be a controlled, reactive, or interactive entity. The *system* has to be linked to all the entities of its environment by its inputs and outputs. Characterizing the system functionalities concerns its environment, its inputs/outputs, its behavior, and its use.

Characterizing an entity or a system is equivalent to modeling its behavior at a given level of detail. Two main model categories are useful:

- *external models*, which use only axioms, relations, algebraic formalisms, and pre-condition and post-condition properties between outputs and inputs. Such models are very formal and are suitable for automatic verifications and for expressing transformations; however, they are not easy to use or to understand for a non-specialist.
- *explicit models*, which use internal characteristic information: state variables, activities or internal operations, events and actions. More simple and understandable by non-specialists, they are commonly used to describe functional specifications.

The nature and level of detail of a model depend on its objective. One model may not be sufficient; consequently, different projections or viewpoints of the object in its description space may be decided: e.g., static description, dynamic description, data description, object description, and activity description.

Concerning *quality*, because a specification is an intermediate document between all customer's partners and designers, it must be verified on the customer side and used by the designers. Therefore, a specification document should be correct; reflect the need; be complete (i.e. contain all characteristics and constraints), be readable and coherent so that it can be verified and used with ease. Moreover, a specification should be executable in order to be validated by tools, supported by a method to guide analysts and should be manageable in order to master complexity and to incorporate later missing items and evolutive specifications. Hence, a specification is first a communication support as an interface between the customer and the designers to reduce distance and is to be used as a basis for verification. It is also a historical reference to the project.

4 Related Models and Methods

Many specification models are well known and currently used: graphical models such as StateChart [19], SDL [3], SpecChart [16][27], SpeedChart, CFSM, etc., and languages such as VHDL, C or C++, HardwareC, synchronous languages [18] - Signal, Lustre, and Esterel -, etc. These models are used mostly to describe the behavioral specification. To design more complex systems, and so to specify at a system level, organizational models are required. Activity diagrams in SADT [29], data-flow diagrams in SA [12] and SART [21][34], Object models [8][24][30][31] are some of these models.

It is more difficult to explain non-functional specifications, and little literature exists on this issue [9][11][17][32]. These are nevertheless essential to describe constraints such as performance, safety, reliability, and cost. These constraints are necessary for selecting a solution during the design an implementation.

In order to describe specifications at a system level, today there is a consensus for system modeling according to three complementary views or dimensions which correspond to three representation spaces: object and data space, state space, and activity space [5][21][25][30]. Each model describes one aspect of the system but contains references to the other models. Consequently, a specification method can be defined by a trajectory within the three-dimensional modeling space.

If many methodologies recommend using the same three-view model, variations exist among methods depending on the order the three views are defined and what components are modeled in each view. It is essentially this aspect which makes the difference between existing specification methods. In this section we focus our attention on two of them: the rather conventional functional method used in SART [20][21][34], and the object-oriented analysis method [24][30][31]. For a few comparisons see [10][13][14][22].

The so-called *functional method* is the result of works done by Yourdon [26], DeMarco [12], then by Ward and Mellor [34], and Hatley and Pirbhai [21]. It is based on the data-flow modeling. Before 1990, the concept of object does not exist in this method. The specification process first recommends a data analysis of the system to model. This analysis starts from the system environment and the system inputs and outputs and leads to the definition of the system context diagram. Data and events are used to describe the essential model by successive refinements as a set of data-flow diagrams (DFD). The main decomposition concept is the activity inside the system. At the end of the refinement, each activity is described by a transformation specification or a mini-specification in a textual form. As the DFD is not precise enough to specify the temporal behavior of all activities, especially for real-time systems, a control-flow diagram (CFD) is added to each DFD to specify the active and inactive states of all activities [21][34].

This analysis technique leads the obtaining of a data dictionary, a hierarchical set of DFD and CFD to specify data transformations, a set of textual specifications for data-flow activities (functional processes), and a set of state-transition diagrams to specify the timeline behavior of each DFD.

The data model is hierarchical. The activity model is also hierarchical and results from the functional viewpoint. Each activity describes transformations applied to data. The data dictionary is progressively enhanced in this way. Refinement is stopped when all activities are described by a sequential textual specification. The behavior modeling concerns the temporal evolution of activities described from their outside viewpoint. If the StateChart is used, the resulting model is also hierarchical [19].

The *object-oriented analysis method* is more and more widespread. It derives its meaning from the increasing systematic use of object-oriented languages like C++ for software implementation. There are different object-oriented ways of thinking, probably because the maturity and stability period have not been reached yet. Here we consider the OMT method recommended by Rumbaugh [30][13]. The method is quite similar to the Shlaer and Mellor method [31].

The specification process is essentially based on the analysis of the application as a set of inter-related objects. These objects are data objects as well as physical, logical and functional components. The object model is the first used to describe the static viewpoint. Concepts of generalization/specialization, class, inheritance, aggregation/decomposition, and relation are used to structure objects and define object inter-dependencies. Each object class is characterized by its attributes and its operations or methods.

Because objects evolve in a dynamic system, the second viewpoint consists of describing the behavior of each object considered important for the dynamic aspect. The so-called dynamic model is recommended for that by using the StateChart. Global diagrams, such as scenarios and use cases [24], are also useful to analyze the behavior of a set of interacting objects.

The third viewpoint concerns the functional modeling dimension, which aims at describing transformation aspects of data by the objects, the functional dependencies and constraints. Functions representing data transformations are invoked as actions in the dynamic model and act as operations on objects in the object model. The functional model uses the data-flow diagram. An activity is described in a textual form (natural language, pseudo-code) or according to a more formal notation. The set of DFD describes the meaning of operations and data constraints in the object model and the meaning of actions in the dynamic model. The specification of these operations leads the enhancement of the object model for the next design step and, later, for the implementation step. Therefore, due to this iterative process, the distinction between analysis and design may seem sometimes arbitrary and confusing.

The *MCSE specification method* is also based on the three orthogonal viewpoints: object/data, behavior, and activities. The order of view modeling is quite similar to OMT, but the components modeled in each view are different. We will show in this paper that the object/data modeling is the first viewpoint to consider but with a different meaning given to the word *Object.* We recommend the modeling of data be different to the modeling of objects; this leads to an important decrease in the number of objects to consider. To avoid confusion, we often prefer to use the word *Entity* instead of *Object.*

The second viewpoint concerns the behavior modeling. The StateChart with some enhancements is recommended for that. What is important here and specific to MCSE is that a behavioral description does not necessarily concern objects. To correctly and efficiently model the dynamic aspect of the system, an appropriate thinking is useful to define which projections are efficient to describe a part of the whole system behavior.

The third viewpoint concerns the activity modeling. The process to follow and the result differ according to the nature and complexity of the problem. For small and medium size applications, the activity diagram is directly the result of the preceding projections. For complex systems, an activity decomposition is first advised in order to decompose the system into sub-systems (Divide to Conquer). This kind of activity is slightly different from the one found in the functional method. There is no difference between a control activity and a data-transformation activity. Each activity integrates both aspects - control and transformation - and satisfies the functional coherence criterion. It results in a minimization of inter-activity couplings.

5 Essential Concepts for Specification

Before describing appropriate description models for system modeling and specification, let us analyze what an entity is and what its main characteristics are that allow for its description. Note that modeling a system is similar to modeling an entity except that, often, the entity exists during the analysis phase and not the system. Five element categories also have to be considered and defined according to the MCSE point of view: object, event/information, data, action, and activity.

An *entity* is a conceptual object of the application. It may have a physical existence such as an operator, a microcomputer, a motor, but it may also be a functional or logical object: for example, a mail producer or an aircraft flight. To facilitate modeling, entities are named and grouped into categories, each having common characteristics. Internal characteristics are called attributes.

An *object* is an abstraction of the problem. It is a dynamic entity owning an internal state and playing a specific role for its environment. It corresponds to a logical entity with a strong functional coherence. An interface specifies the boundary between its inside and its environment. Our definition here is more restrictive than the one commonly used in object-oriented analysis (OOA). The objects we are considering are parts of the environment or of the system. Therefore an entity is an object, it is the same for the system.

An *event* means an occurrence of a significant state change. Two types of events are useful for system modeling:

- a *state change*. The event name directly and completely defines the meaning.
- an *item occurrence*. This type of event is associated with a content. The event defines the moment of the occurrence and so the availability of the associate item. The content (a data value or anything else) enhances the meaning of the event.

To avoid any ambiguity in the terminology when a distinction is necessary, the first type is called *event* and the second *information*. In the notation, an event is only named, while an information is defined by its name and its data type or definition domain.

A *data* is a variable or a set of variables whose existence is considered to be permanent. Its value evolves with time. State variables and attributes of entities and objects are examples. A data can also characterize an interconnection between entities. A data item is defined by its type specifying values domain.

An *action* is considered an atomic operation. An *activity* represents an encapsulation of a set of operations and thus lasts a certain amount of time. The boundary between action and activity is dependent on the model abstraction level. An action can be refined as an activity for a more detailed model.

An object is represented by a rectangular form and links for its inputs and outputs. Inputs and outputs form the interface with its environment. The dynamic character of the object results from its internal activities and actions. An activity is represented by a rounded-corner rectangle. To distinguish between events and permanent data for inputs and outputs, we use a single arrow link to denote events, and so information, and a double arrow link for data, as proposed by Ward [34].

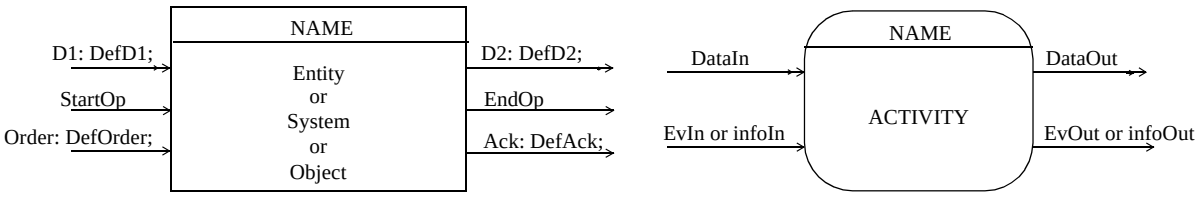


Fig. 2. Notation for an object and an activity.

In Figure 2, D1 and D2 are respectively a data input and a data output. StartOp and EndOp are events, Order and Ack are information items defined by their types.

For an object or activity input, data and states are continuously observable (readable). Events and information are fugitive: this means that an information is significant only on its occurrence. For an output, data and states are modifiable at any time by the object or the activity, whereas events and information imply internal actions at specific times.

6 Model for Functional Specification

To build a system-level model which may be described by a structure from simpler components, it is necessary to add a more precise definition of each component and of dependency relations to the five preceding categories of elements. A more precise definition corresponds to a refinement. Relations between components allow us to represent an organization. Refinement and organization are both useful to express a behavioral description and a structural description.

To be more precise, a data or an information item must be refined to describe its composition except if it is an elementary type. Specifying an activity or an object first implies its refinement, then its contribution to its environment, which means to express temporal and structural dependencies.

Due to the lack of a single model containing all these features, a coherent description of an object, an entity or a system, requires three complementary views shown in Figure 3.

- *Object and data modeling* (WHAT). This describes the static characteristics of each component or item. When it is compound, the structure and the definition domain of each basic item have to be described resulting from the use of the refinement technique.
- *Activity modeling* (HOW). The purpose here is to describe the internal activities of the modeled component and all relations with its environment. This is, again, a structural modeling viewpoint but one which concerns dynamic functional features.
- *Behavior modeling* (WHEN). This involves the describing of temporal dependencies between the occurrence of events and the execution of actions implying data and state modifications. This is equivalent to modeling the temporal evolution of objects, data or/and activities.

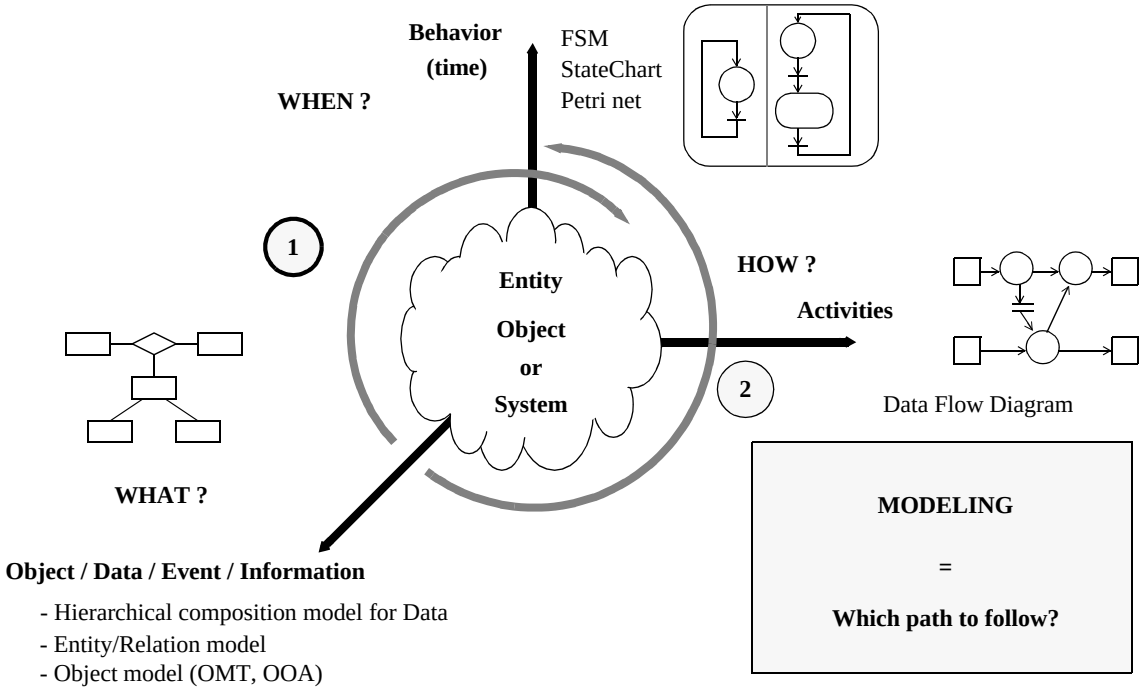


Fig. 3. The three viewpoints for an object, an entity or a system.

This modeling strategy results from the principle of projection according to several viewpoints when the object to be modeled is rather complex. The three projections considered here are not of the same nature: static and structural (WHAT), dynamic and structural (HOW), dynamic and behavioral (WHEN).

These three viewpoints are complementary and undissociated. To characterize an activity, the data used and generated have to be known and, so, defined first. To model an entity or a system, all its internal activities and the reaction of each of them to input stimuli have to be found and described. The global coherence is the result of an appropriate description for each view, thus providing a verification method. As a matter of fact, the three-viewpoint approach implies some redundancy between the evolution of data and of activities, since activities modify data.

The preponderance of object/data on activities has to be recognized [25]. The data attributes of an entity are more stable than the functions and their behaviors. Hence the prime importance given to data in the model.

The result of this analysis is in agreement with modeling procedures recommended by other authors of real-time system design methodologies. Hatley and Pirbhai recommend a two-view model: activities, and behavior to control them. In Appendix C of their book, they describe the third data view [21]. Ward and Mellor consider the three views of the model [34]. Rumbaugh describes a three view model in OMT: object modeling, behavioral modeling, and functional modeling which here means activity modeling [30].

An important question here is, which modeling process should one follow in order to obtain a good model having the qualities of a formal specification? The starting viewpoint is compulsorily the static object and data modeling. But afterwards there are two possibilities: behavior description of components of the first view (path 1 on Figure 3), or description of activities and the behavior of each of them or all together (path 2). Methodologies differ on that point even when they are based on the same three viewpoint model. This interesting observation will be discussed later in a methodology comparison in Section 15.

In the next sections we will describe the model recommended in MCSE for each of the three views in order to be able to describe the corresponding specification process later on.

7 Object and Data Modeling

The object/data modeling results from a static analysis of the component concerned. The word *component* is used as much for an entity of the environment, as for the whole system, or a part of them. The modeling objective of a component is to express its internal components, their definition and/or internal structure, and then structural organization.

For us, two categories have to be differentiated even if the same modeling concepts can be useful for both: objects and data. The word *object* is considered here with its usual meaning in object-oriented modeling; hence, following the definition given by Rumbaugh: “an object is simply something that makes sense in an application context”. From this definition, it emanates the position that every component is an object. For the modeling we are interested in, i.e. system-level modeling and not software modeling as it is the case for most object-oriented methodologies, we limit the use of the word *object* to a component having its correct and true identity and its own functionality. This means that an object is an active component. What we call an object during the specification analysis is in reality similar to the concept of entity or part of it. We are using the meaning of the object-based approach more than that of the object-oriented approach [22], which leads to consider a large amount of objects. We recommend the use of the word *Data* or *Item* or *Information* when the objective is to specify values or any structured collection of values.

In the two following sections, we describe object modeling first and then data modeling.

7.1 Object Modeling

The object modeling approach is usable at every level of complexity and abstraction and may concern all or part of an application. If, for example, we consider a whole application, the object analysis leads to the identification of all active objects of the problem. Naturally, this leads to the identification of the system to be designed and all the entities in the system environment included in the application. When we consider an entity, the object analysis leads to the identification of internal components called “objects” in this case. The object modeling consists of:

- the decomposition or composition of a component,
- the specification of each basic object.

The first aspect concerns a decomposition description in the case of stepwise refinement analysis, or an aggregation description in the case of an abstraction process. The resulting representation for both cases is the same: the use of the aggregation modeling technique. Regarding the second aspect, the objective is to express object properties. For the sake of simplicity we will begin with this second aspect.

-A- Basic Object Description

An object has its own identity or name and its own type or class. An *object class* describes a group of objects with similar properties called attributes. Each object is then an *instance* of its class.

An object class is defined with a set of attributes. An *attribute* is a data value held by the object class. For example, Speed and Position are attributes of the object class Trolley. Each attribute is characterized by its definition domain, also called its type. Elementary-object modeling mainly concerns the identification of internal attributes just useful to solve the problem. It is therefore important to choose the necessary and sufficient appropriate modeling level.

Contrary to considering object-oriented approaches, we do not consider operations or methods of the object class at this stage of the analysis, because object transformations or internal functions are difficult to identify clearly. The specification of operations takes entire sense later during the design and implementation steps, an object being at that time an encapsulation unit.

An object in the sense we consider it here has a functional coherence and plays a specific role for its environment. The coupling-links between an object and its environment form the *object interface*. These links are the medium for data, event and item or information exchanges. Figure 4 gives an appropriate representation of an object with its interface.

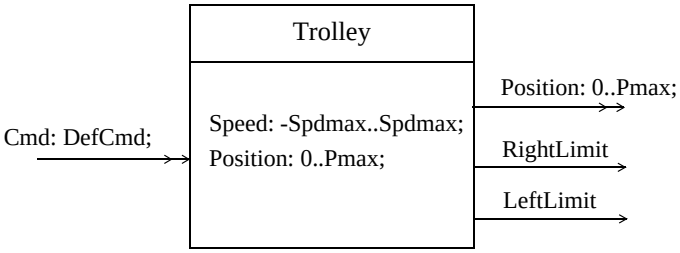


Fig. 4. Representation of an object class with its interface.

Cmd is a data link specifying the control of the motor of the trolley all the time. RightLimit and LeftLimit are events occurring when the trolley goes outside its legal position. Position is an internal attribute of the trolley like Speed, but which becomes observable to its environment through the interface. With this representation, the object characterization is quite similar to that of the object-oriented approach because, internal data (private), public data (accessible from the outside), actions on the object though without specifying operations and methods, are identified.

-B- Generalization/Specialization

Generalization/specialization and inheritance are powerful abstraction concepts used in order to share similarities between object classes while preserving their differences. Generalization expresses a relation between a class and one or more other classes which are refinements concerning properties and/or added attributes. The initial class is called a superclass, the refined class is a subclass. The relation concerns only properties and not organizational dependencies. Figure 5-a describes two types of trolleys which differ in propulsion means. Control inputs are then different.

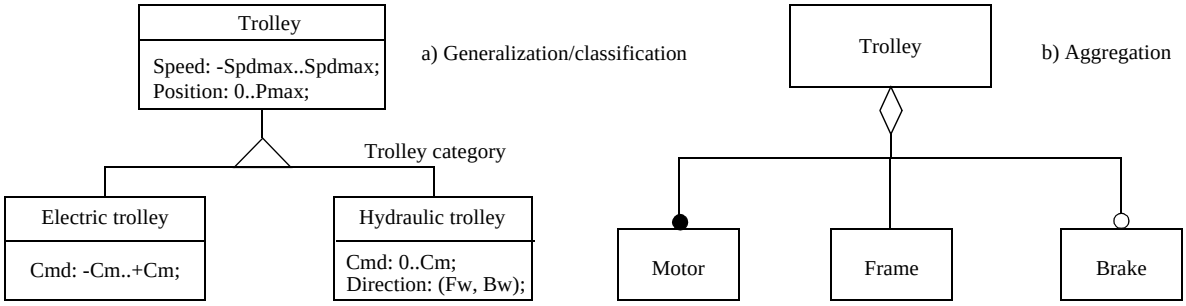


Fig. 5. Class generalization/specialization and aggregation assembly representations.

A triangle represents the generalization concept used to link a superclass to its subclasses. The notation on the right side in Figure 5-a (not obligatory) indicates a discriminant feature to better structure classes. The generalization concept is a powerful means to define more general and so more reusable models.

-C- Aggregation/Decomposition

Another important concept of object modeling concerns the representation of a component with its constituents. This means the use of the *Part of* relationship. Aggregation here is synonymous to assembly. Figure 5-b shows the notation using a diamond to represent the decomposition of a trolley in three parts. Hollow and full dots are used to indicate the object multiplicity, zero or one (hollow), many (full).

-D- Organization

The analysis may also lead to the identification of objects linked together but which are not of aggregation or generalization types. In this case, it is simply an association relationship. See [30] for more details.

More important for system design, when an object, an entity or a system is composed of more elementary objects, the decomposition is shown by an aggregation diagram. But for us, each object owns inputs and outputs. Therefore, it is necessary to add the input/output links between objects to the preceding representations. This is the goal of a structural representation describing the organization and all object interconnections through their interfaces. The *object interconnection diagram* is added for that purpose. Figure 6 gives an example of an application composed of four object classes.

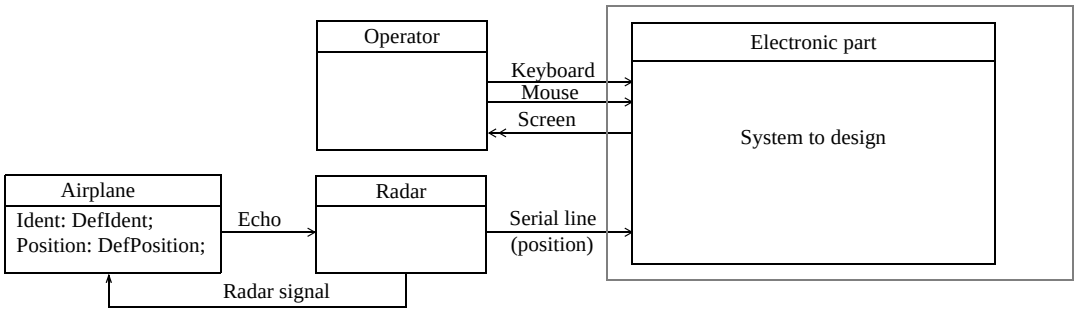


Fig. 6. Object interconnection diagram.

Such a diagram is of course correct only if all the connections respect the link categories - event, information, permanent data - (single or double arrow), and data type compatibility. This diagram normally concerns object classes and sometimes instances, the aggregation diagram indicating the number of instances of each class.

7.2 Data/Information Modeling

Confusion between data/information modeling and object modeling is easily made, since in the object-oriented approach only the last one is used because all things are considered as objects. We recommend maintaining a clear distinction between both. Having first described the object modeling which concerns physical, logical, functional yet active components, the data/ information modeling specifies the data links and identifies internal attributes. Data modeling is very useful in specifying attributes which can be complex data structures.

The data/information modeling objectives is to allow for a complete, formal and understandable description that would be useful for a large category of data from the boolean to relations, and is efficient to describe only the semantics and not the implementation. The method has been well developed in many books [2] [12][25][21][34], but is not often known and used by system designers. Here we only give the interesting result and the common notation.

To model a data, three essential aspects have to be explained: the meaning of the data, its composition, the definition type of each component [34]. The structuration model is based on the three fundamental operators: the cartesian product (composition), the alternative (selection), and the set (multiple similar elements). Figure 7 indicates the syntactic notation and a corresponding graphical notation [5].

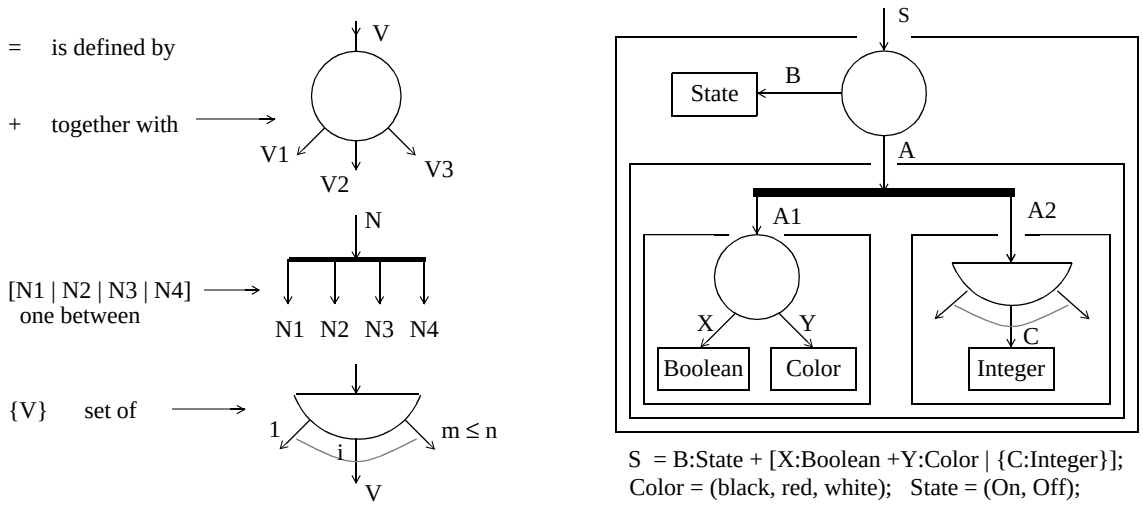


Fig. 7. Graphical and syntactic notations for data modeling.

Beyond individual identified data, relations between data sometimes have to be considered: e.g. database applications. In this case, it is useful to use the entity/relation model or the object model and, specifically, the association relationship concept [30].

8 Activity Modeling

The aim of the activity modeling is to define the relations between data, events, information items, and the internal activities of the object, entity or system.

An activity represents a transformation of input data or information items into output data or information. It is expressed as a set of time-ordered interdependent actions and/or more elementary activities. We are here specifically concerned with the structural aspect. The activity model describes the structural dependencies between activities. Two models are well known: the SADT model [29] and the Data-Flow Diagram (DFD) [12]. Both represent the flow of data or information but do not explain the complete behavior of the whole model.

In the MCSE methodology, we recommend the use of a structuration model that we call the *activity diagram*. The goal of this diagram is to represent only the organization view, even though the data flow diagram is more known to describe a behavior but partially. Figure 8 gives an example of an activity diagram to model an object, an entity or a system.

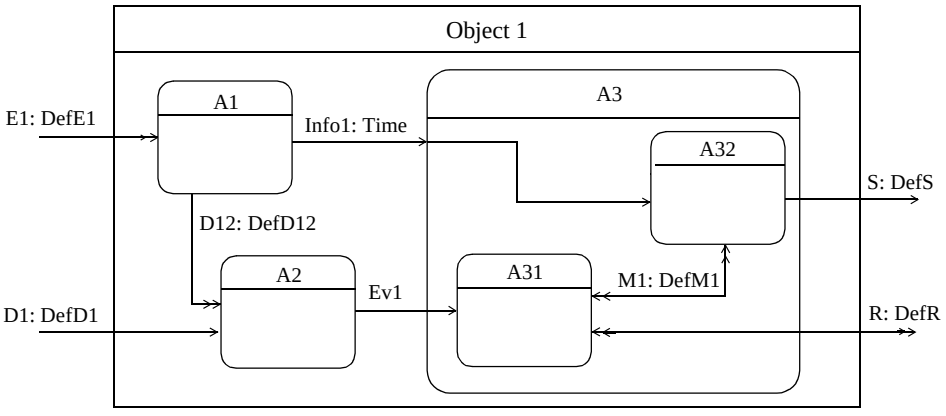


Fig. 8. Activity diagram example.

Note that the single and double arrow links are used to differentiate an event from permanent data. A3 is refined and M1 has to be considered as an internal attribute of A3. This diagram expresses that *S* results from the transformation of *Info1* by *A32* by using the data *M1*. *Info1* results from the transformation of *E1* by *A1* which modifies *D12*. The *Ev1* event produced by *A2* using *D1* modifies the activity *A31* which uses and modifies *M1* and *R*.

Features of this model are:

- it a graphical easy-to-understand and easy-to-use model,
- it is hierarchical, allowing progressive structuration and analysis,
- it describes the activity organization viewpoint of an entity or a system,
- it does not describe the global behavior. To do so, it is necessary to add the behavioral description of each activity.

Remarks: Do not confuse object and activity. The notation seems quite similar except for the shape, and both are encapsulation units. The difference lies in the meaning. An object is a physical or functional unit of the real world and can be refined by objects or internal activities. An activity is a behavior unit which describes an object functionality. An activity cannot include any objects. The meaning of the names is also different: for example, Transmitter for an object, and Transmission for an activity.

It is well-known that the data-flow diagram does not completely describe temporal behavior. This was why, for real-time applications, Ward and Mellor, Hatley and Pirbhai enhanced the DFD model with the CFD (Control-Flow Diagram) in order to specify the evolution intervals of all activities. We do not consider this extension, because our approach consists in adding the behavior description of each activity inside itself instead of describing in one control activity the temporal constraints of the evolution of all the activities of a refined activity or an entity or a system.

9 Behavioral Modeling

Behavioral modeling concerns the dynamic aspect of a component. For an object, an entity, a system and an activity, the objective is to express dependencies between events and actions. According to the behavioral viewpoint, a component is always in a state corresponding to a step or phase of its evolution. A modification on its input(s) and/or internal events can imply a state change.

A variety of behavioral models exists. First of all, we differentiate discrete-time models from time-continuous models. The second category consists of: parametric analytic models - transfer functions, differential equations, recursive equations, etc. - and non-parametric models - waveforms, and frequency bandwidth model, etc.

We are often more concerned with the first category without excluding the use of the second one when necessary. Based on the discretisation of behavior, many models are well-known. Some are graphical models: FSM, SDL [3], the StateChart [19], the SpecChart [27], and Petri nets. Others are general languages: VHDL, ADA, and C. Still, others are synchronous languages: Esterel, Lustre, and Signal [18].

When a discrete-time model is appropriate, we recommend the use of state/transition diagrams in the MCSE methodology and, more specifically, the StateChart, which is a powerful graphical notation. It allows us to graphically represent hierarchical state decomposition and concurrency inside states. The time-out concept is also useful. This model is currently recommended in many methodologies, and computer tools are also available.

9.1 Recommended Notation

Our objective here is not to describe the StateChart model; we assume it is well known. We want to explain some specific enhancements we consider important and usable with all discrete-time models. These enhancements were added to increase the formal feature of the model and also to facilitate the deduction of the solution during the design. Figure 9 is an illustration of the recommended notation.

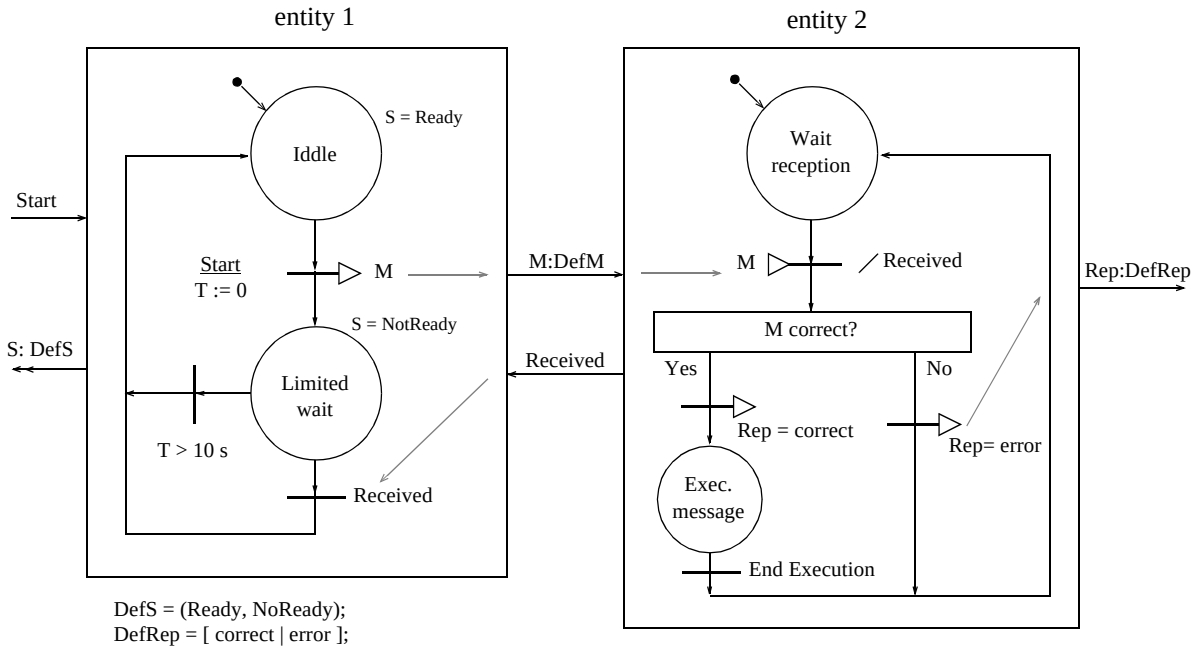


Fig. 9. Recommended notation to represent finite-state diagrams.

First, we recommend the use of both the Moore and Mealy notations. Transitions are explicitly represented by an arrow (bar) on links. Atomic actions (:=) are associated with transitions. Non-instantaneous actions are associated to states (=).

Second, when many states are accessible from a given state, conditions have to be exclusive. Some terms in the conditions are often main conditions. To increase readability, we recommend the use of an extended rectangle with the condition inside and all possible cases as outputs. This simply means a “switch”. The notation is powerful particularly when a message has been received, and the next state is dependent on the content. See the M message in Figure 9.

Third, in the model it is essential to clearly identify events and permanent data (single or double arrow). If the behavior concerns an object, an entity, a system or an activity, we recall that three categories of inputs and outputs are possible: event, data, or information. The link notation allows for this identification.

Also important are the meaning of temporal dependencies between diagrams. Three kinds are to be considered:

1- a *boolean expression* on data: all transitions marked with this condition, for which the preceding states are active, are crossed or fired. A specific data value may be a time variable which evolves naturally at the speed of our world time.

2- an *event* (or a OR combination of many events) to which a boolean expression can be added: the result is similar to the preceding case. The occurrence of the event is broadcasted to all diagrams in the concerned component. The event is not memorized.

3- an *information* (or a combination of some information items) with possibly a specific content or a boolean expression acting as a guard: all validated transitions are crossed and the information is sent to all waiting receivers. Otherwise (receiver(s) not ready), the generation of the information is delayed until a receiver is ready. The transfer of information is in conformity to the rendez-vous protocol.

Hence, the two first dependencies do not imply waiting on the generator side, whereas the third case implies a rendez-vous. This is easy to remember; an event is only an instant in time, while an information has a content to transmit. The rendez-vous guarantees the content transfer. Note also that all couplings are of broadcast type. To distinguish the third case from the two others, we use the triangle symbol in the outside direction for sending and in the transition direction for reception. This notation is similar to the "?" and "!" symbols in the CSP and OCCAM languages.

To correctly understand the example in Figure 9, the transfer of M from Entity 1 to Entity 2 is done after the event Start and when Entity 2 is in the state WaitReception. The instant of rendez-vous is dependent on the execution time of the state Exec.Message in Entity 2.

9.2 Different Modeling Approaches

The behavioral model is useful to describe all types of components such as an entity, an object, a system (or part of it), and an activity. These components are first characterized by their inputs and outputs which are useful to specify conditions and actions. The analyst who has to develop a behavioral model can follow two different approaches:

- *State modeling*. A state modeling allows us to globally describe the behavior of a component by describing the evolution of its outputs conditionally to its inputs and internal events or conditions. This leads to an external modeling of the component. States are deduced from the analysis of data, events, and attributes. Therefore, a preceding static modeling makes the dynamic modeling easier. The process to follow is to search for all successive states first, then transition conditions and finally actions. This process is very easy and efficient for physical entities generally described by an internal state variable.
- *Stimuli/response modeling*. We now consider an entity rich in input stimuli and acting on each one. Rather than searching for internal states, it is easier to describe the successive actions on each occurrence of an input stimulus. This kind of modeling is useful for reactive systems. It is also the case for communication systems, and human/computer interfaces, etc.

10 Specification of Performance Constraints

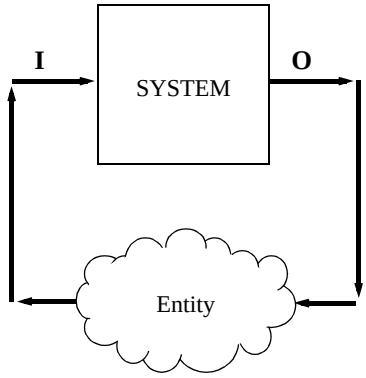
The specification of a system must also include non-functional specifications. In this set of requirements, an important category concerns dynamic performances which we include in the operational specifications. They are added to the functional specification described above from an analysis of non-functional requirements. Though the following is not exhaustive, dynamic performances of a system express:

- timing constraints,
- speed or frequency, throughput or processing ability,
- precision, accuracy, and tolerated error.

In this section we give a classification of dynamic performances in the three previous categories, define each of them and show how to add them to the functional specification.

10.1 Meaning of Performance Constraints

Dynamic performances specify observable reactions of a system or an object to its environment. Figure 10 shows that the interaction of a system with its environment defines four categories of constraints considering all possible input/output pairs: I/I, I/O, O/I, O/O [9][11].



End / Start	I	O
I	between two successive events: for one input : - activation frequency - processing ability for a set of inputs : - processing ability	between one input an one output (stimulus- response) - response time - latency - propagation delay
O	Availability	between two successive events - throughput . for one output . for a set of outputs - reaction speed

Timing constraints

Throughput constraints

Fig. 10. Classification of behavioral performances.

The meaning of each category differs a little if the system is considered a reactive type or a transformational type for its environment. Similarly, the type of input or output modifies the constraint meaning which is called throughput, processing ability or transaction rate in the case of event or information, and reaction or modification speed or rate in the case of data input or output. For example, in the case of I/I, we may consider the maximum number of transactions on all inputs or the minimum and maximum time between two requests on one input. The case I/O is the most common for reactive systems: minimum and maximum response time, maximum latency of an event or message, precision of an output according to a setpoint input, etc. The case O/I is most often forgotten; it concerns the availability of the system to react to, follow or process environment modifications after system controls or actions.

10.2 Specification of Performance Constraints

The preceding analysis shows that dynamic performances may be classified according to three categories: timing constraints, throughput or rate constraints, and precision or accuracy constraints. Each of them is defined with specific attributes described below. Most constraints are related to the system inputs and outputs. Some others, which are more specific, concern the inside of the system; therefore, the use of the functional specification is necessary.

-A- Timing Constraints

A timing constraint (TC) defines a time difference between a resulting event or action and a source event. An event is understood in its wide meaning, i.e. a state change. Considering only system inputs and outputs, the TC notation on Figure 11 shows many different timing constraints. The TC_{IO} type is the most common. For readability of performance constraints, they are represented by arrows from the source event to the dependent one on the system diagram or on an activity diagram.

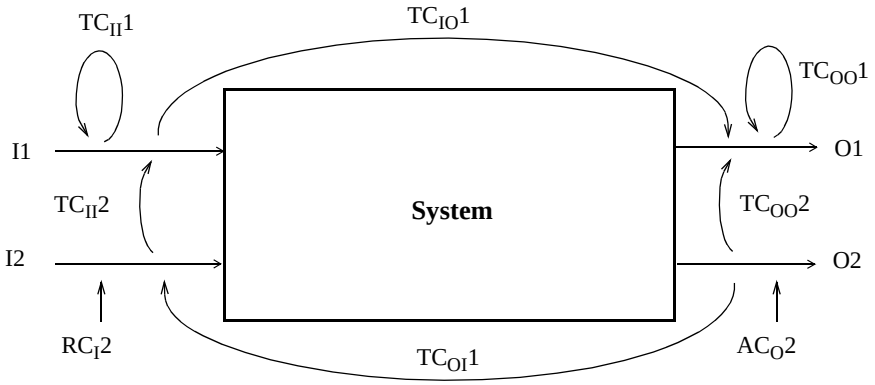


Fig. 11. Different types of performance constraints and notation.

A timing constraint is defined by its name and the following attributes:

Condition:	maximum, minimum, max. and min.,
Limit time(s):	constraint value(s) with its unit,
Start event(s):	name of the input or output, name of the event,
End event(s):	name of the input or output, of the event or action,
Percentage:	rate the constraint must satisfy,
Context:	workload of the system, environment configuration, etc.

For example, the system must react to the alarm event on the input I1 by generating a control action on the outputs O1 and O2 after a delay between 1 ms and 2 ms in all system-load situations.

-B- Frequency, Rate, Throughput, Processing Ability

A frequency constraint FC, a rate or throughput constraint RC, a processing constraint PC concern one or more inputs and one or more outputs. This kind of constraint is defined with the following attributes:

Condition:	maximum, minimum, max. and min.,
Limit value(s):	constraint value(s) with its unit,
Tolerance:	acceptable deviation of limit value(s) with its unit,
Element:	input(s) or output(s) name(s),
Percentage:	rate the constraint must satisfy,
Context:	idem as timing constraints.

Units may be various: number of messages or transactions or packets per second, MIPS or MFLOPS, and frequency or period, etc. For example, the communication system must satisfy a 100 messages/s throughput in 90% of the situations in the degraded mode.

-C- Precision, Accuracy, Error

A precision or accuracy constraint AC and an error constraint EC concern an input or an output as the data medium. The following attributes are important:

Reference element	name of the input or of the reference data or variable,
Measured element	name of the output or controlled variable,
Tolerance:	acceptable deviation with its unit,
Response time	maximum time to satisfy the constraint,
Percentage:	rate the constraint must satisfy,
Context:	idem as timing constraints.

Units may also be various: absolute (3 Rpm) or relative (precision in this case, 0.01%). For example, the speed precision of the elevator according to the reference C does not exceed 5 Rpm in all load situations, the transient phase being lower than 1 s.

10.3 Performance Constraints related to States and Transitions

The specification of performance constraints may not always be defined on system inputs and outputs. As a matter of fact, in some cases, outputs do not allow for the observation of constrained events or data. It is the same for inputs which cannot control specific internal events or data values. In these cases, constraints are expressed on activity diagrams and/or behavioral diagrams. Figure 12 shows two examples and the notation which is the same as above.

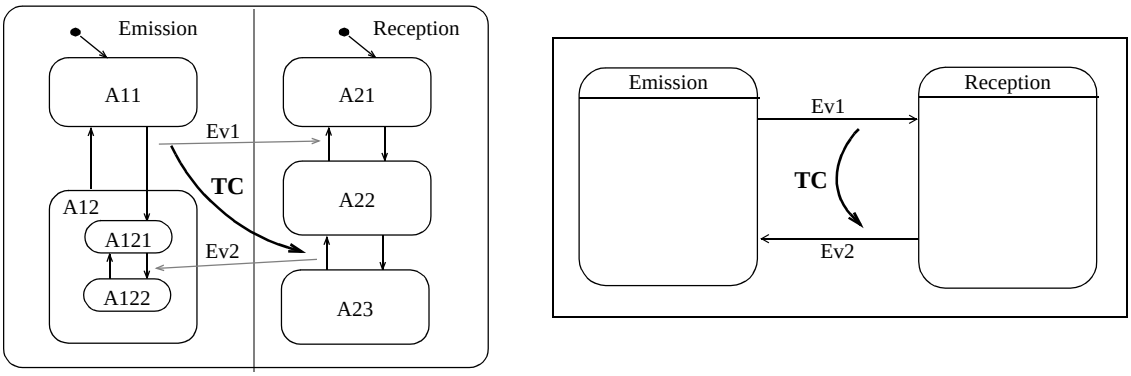


Fig. 12. Expression of internal timing constraints on functional specification.

Note that defining a timing constraint on behavioral diagrams must be an exception. It is better to use an activity diagram and to express the constraint between activity input(s) and output(s) which normally correspond to essential internal data or events in the system.

11 Technological Specifications

This category of specifications is generally vast because it includes all the constraints which are relative to the hardware part, the software implementation, product manufacturing, installation, maintenance, etc. However, these specifications are quite common for designers and relatively easy to describe. Some of them are linked to the physical analysis of the environment. Below, some essential constraints are reminded and classified.

- *Geographic distribution constraints*. When the application is geographically distributed, these constraints describe the implementation topology, subsystem locations, distances, coupling methods.
- *Electrical interface specifications*. These concern the physical input/output requirements.
- *Human/Computer interface specifications*. Most systems are directly accessible by users or operators. These specifications describe, first, the technological components, and second, the user procedure. This last item may be expressed in the form of a document which may be a tedious job, but also in the form of a prototype, a means to simplify the customer verification.
- *Performance specifications of the implementation*. These constraints are not to be confused with the operational specifications. Here, it is necessary to consider the hardware part and all the physical interfaces with the environment. Timing constraints on electrical signals, resource utilization, and storage size, etc. have to be defined.

- *Dependability constraints*. This is a large class of constraints difficult to specify. These constraints also depend on the application domain: aerospace or military systems, automotive domain, telecommunication, and leisure products, etc. This category includes at least: system reliability, availability, fault-tolerant constraints, security, safety, maintenance procedure and constraints, and self-tests, etc.
- *Electrical constraints*: consumption, dissipation, connections, and types of components, etc.
- *Miscellaneous specifications*. This category includes requirements such as constraints related to the context in which the product will be manufactured, environment-related conditions, life cycle constraints, and development process constraints, etc.

12 Specification Method

After having completely described the model and notations that we advise in order to express system specification, we now detail the recommended specification process to efficiently identify functional specifications and performance constraints. First of all, to be able to consider complex systems, it is important to choose the appropriate abstraction level for modeling the system and its environment. This point is explained first. Then, we show why the task of identifying functional specifications is the most difficult one. A good knowledge of the model is not enough. Designers must master an efficient process in order to obtain a good quality specification which is easy to verify and to exploit for design.

12.1 Functional-level versus Physical-level Analysis

Let us first consider the nature of the objects of an application. During the analysis, the easiest way to start specifying consists of first considering entities in the system environment. They may be engines, trolleys, ovens, users, and a nuclear reactor, etc. The concrete vision of these entities naturally leads to their physical modeling. Models are based on a detailed knowledge (point 1 in Figure 13) including technology-dependent information. Specifying the system at the same level (point 2) from this environment modeling implies an undoubtedly complex model of the system due to the amount of information that has to be managed (direct transition from 1 to 2).

Figure 13 shows the benefit of modeling at more abstract levels. The model complexity is greatly reduced and is also technology-independent when modeling objects, entities and the system at the functional level.

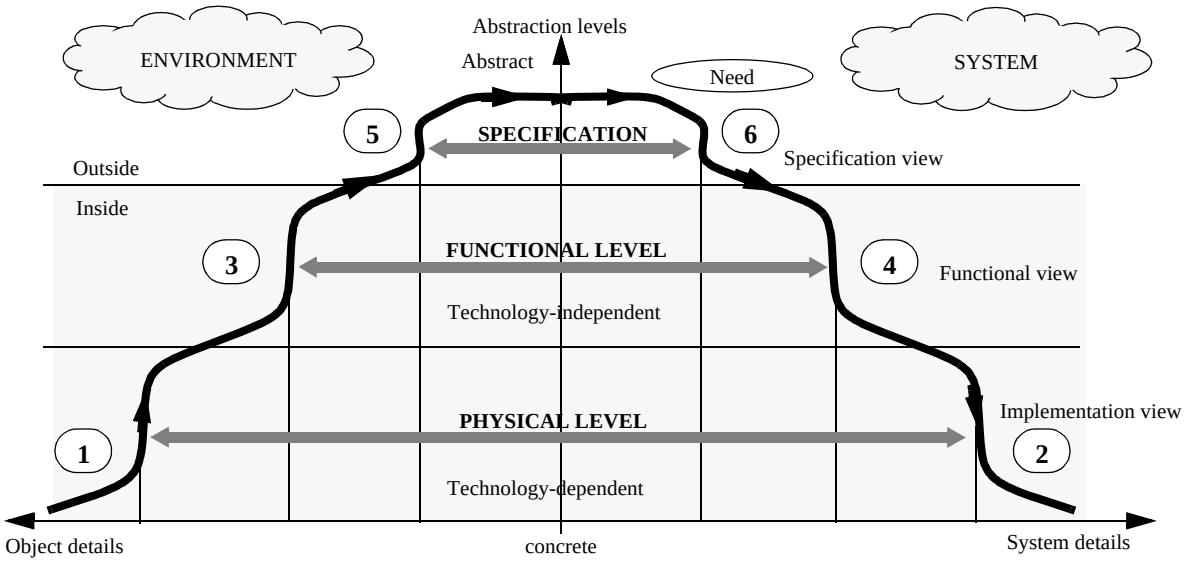


Fig. 13. Object and system complexity related to the abstraction level.

Entities of an application can be described without considering technological characteristics. Called a functional-level modeling, it expresses, on the one hand functional dependencies with its environment by functional inputs and outputs, and, on the other hand, the functional behavior, i.e. its role in relation to its environment. The functional vision implies a great reduction of complexity by first decreasing the number of entities to the only functional ones, next the number of the entity and system inputs and outputs, and, finally, the number of functional attributes. Moreover, a functional model is more stable than a physical model. An appropriate functional modeling of the environment (point 3) makes searching for the functional modeling of the system easier (point 4).

The specification level is located above the functional level which still concerns an internal view. Specifying entities (point 5) and then the system (point 6) is relatively easy after the functional analysis of the environment. Figure 13 clearly shows that the analysis of the environment follows a bottom-up process, the analysis of the system functionalities and constraints follows a top-down process. Correctly mastering the bottom-up procedure is of great benefit to specify more complex systems.

For us, technological characteristics include the following at least: physical and electrical information or constraints, geographic distribution of objects or entities, and physical and human/ computer interfaces. Therefore, one must not confuse interface objects and functional objects. A shaft encoder to measure a speed, a microcomputer with its keyboard and screen, and an Ethernet cable, etc., are examples of interface objects.

12.2 Overview of the Specification Process

In our opinion, this first step in the MCSE methodology whose objective is to produce a complete and verified specification document, is the most difficult. As input data, the specifier makes use of the system requirement document, if there is any. It is his responsibility to ask questions, analyze information and answers, then to synthesize the specification and to have it read and verified by the customer.

Based on the structure of the specification document to be written, Figure 14 summarizes the different phases of this step. The work is decomposed into two major phases:

- environment analysis,
- system specification.

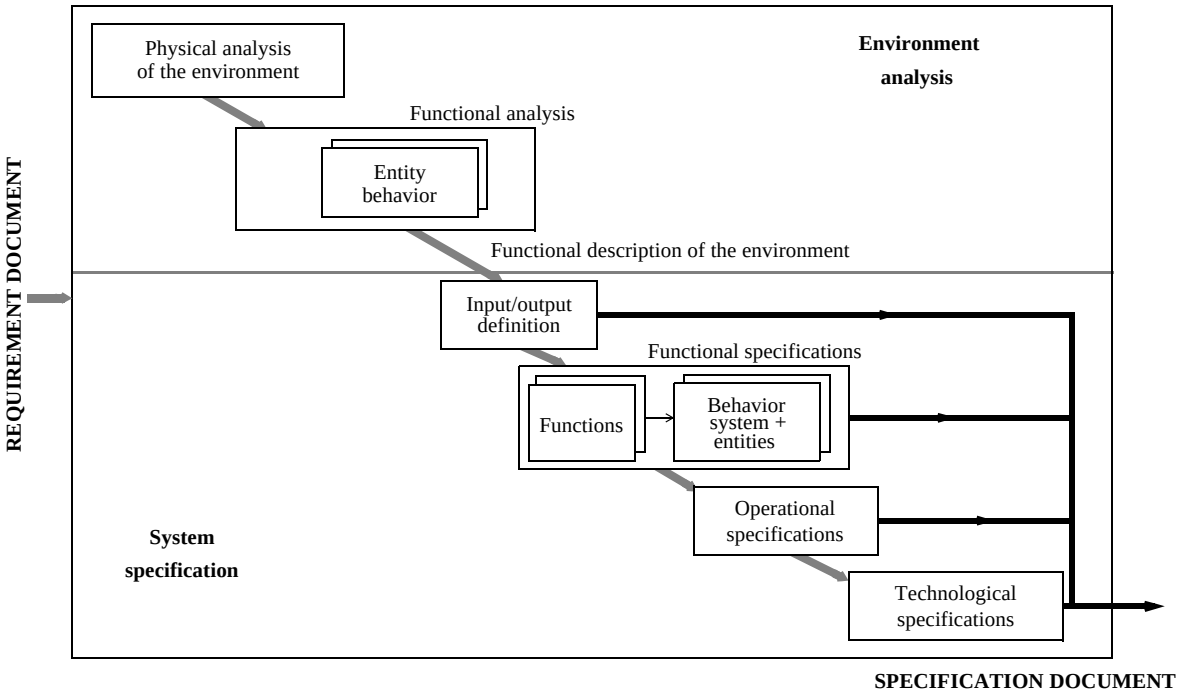


Fig. 14. Description of the specification process.

For the first part, the initial phase concerns the identification of the application objects at the physical level. The next phase leads to a transposition at the functional level keeping only functional objects with their functional inputs and outputs, each object being modeled by a functional behavior. This part is completed by a functional synthesis of the complete environment.

The second part starts by delimiting the system with its functional inputs and outputs. The next phase is essential to describe the functional specifications. Operational and technological specifications are progressively written, together with a complementary user, installation and test documents.

During each phase which consists of modeling real objects or the system to design, the specifier proceeds according to three steps:

- *Analysis*: the objective is to bring out all useful information - data, events, actions - through discussion with the customer and reading documents, by considering scenarios, waveforms, use situations, etc.
- *Synthesis*: all useful information has to be collected and organized in the form of a model as an abstract description.
- *Verification*: this task is necessary and guarantees the validity of previous work. Omissions and errors are then immediately corrected. Being considered as an important task, it must be done with great care.

It is essential to note that the starting point, finding useful entities, is very important since the entire design and implementation will be based on that first analysis. An investment in time for this initial work is essential. This time is not wasted although it may seem to be at first. The whole specification process will be illustrated later in one case study.

13 Expressing Functional Specifications

Functional specifications are essential information needed to clarify the problem to be solved and the objective of the system. Indeed, the design starts from this category of specification. Consequently, a bad specification and any inaccuracy will inevitably cause problems. In this section, we first describe the meaning of functional specifications and then the recommended procedure.

13.1 Nature of Functional Specifications

Functional specifications describe functions that the system must carry out for its environment. The only functions to be considered are service functions (Afnor norm), also known as external functions. Do not confuse them with the internal functions of the system which are the result of the next design step. *Service functions* are the actions expected by the system to answer the customer's needs. It means that such functions are the result of appropriate collaborations between the system and the entities of its environment. These system functions are divided into main and secondary functions.

But it is not sufficient to list these functions. To avoid misunderstanding and interpretation ambiguities, all system functions must be completely explained. The system is considered a black box, and functional specifications are behavioral descriptions from the outside viewpoint of the system. Figure 15 indicates the analyst's situation and what he must specify.

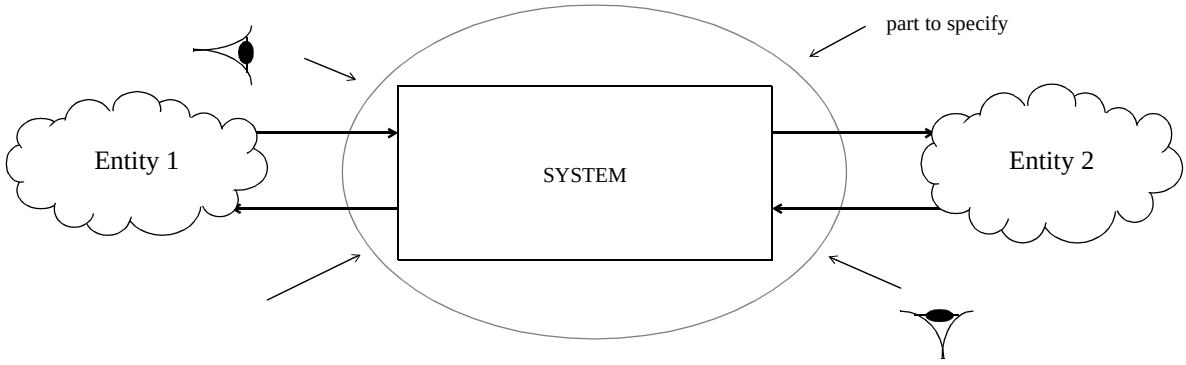


Fig. 15. Example of an application and part of it to model.

A functional specification describes the actions of the system upon the environment entities carried out by the system outputs according to data and information from the environment observed by the system inputs. Behavioral models described before are useful to express such specifications.

13.2 Different Approaches to expressing a Functional Specification

As the specification objective is to describe the system from an external system viewpoint, a first difficulty lies in selecting the model type to be used. But since a more complex system carries out several functionalities, it is rarely possible to consider only a single model to express the global system behavior. Different viewpoints or system projections must be concurrently considered. Another main difficulty is to choose the most appropriate viewpoints, first to reduce their number, then to obtain a complete and non-redundant functional specification.

To do so, the specifier first has to consider the three model views: static object/data model, behavioral model, activity model. After having used the first static view, the behavioral modeling results from three non-exclusive approaches corresponding to different viewpoints of the application:

- describing the entire (or part of) system behavior starting from the evolution of its outputs and/or inputs,
- describing the evolution of entities under the system constraints,
- describing the system based on an activity analysis (global approach).

These three approaches may be used concurrently. Hereafter, we describe the first two of them which are mainly recommended in MCSE; the last one corresponds to the Structured Analysis method whose objective is to specify with a data-flow diagram.

-A- Approach from System Input(s) or Output(s)

For the approach from an output, successive states dependent on system inputs have to be expressed using a behavioral model (StateChart for a time-discrete behavior). The modeling may be done for each output separately or for many functionally coherent outputs at the same time.

The modeling may also start from each input using a stimulus/response approach. For each possible data value or information item on an input, the model expresses the system timeline evolution and its actions on the environment by outputs. This approach is efficient for communicating systems and reactive systems.

Such an approach, of course if well used, is the fastest and the most efficient. This is the case when behaviors of the entities are not essential, i.e. they do not give enough details to clearly specify the system behavior. In this case the behavioral model is mostly independent of entities, but mostly based on input and output data definition.

-B- Approach from Entities

When entities are essential to describe the behavior of the system, we recommend an approach from entities. Rather than attempting to describe each system output separately, it is better to express the successive required states of application entities. The important method recommended by MCSE is therefore to characterize the behavior of each entity under the constraint of the system.The states of each entity are then used to determine actions or control outputs generated by the system, leading to correctly specify system output(s). This rule is very important and efficient. In describing the effects of the system and the conditions which produce them, we indirectly specify the system functions. This leads to a really external model of the system (meaning functional specification) since state names are related to entity states and not to internal system states. This approach is made easy by the fact that the first phase of the specification procedure concerns entity modeling, but each entity taken alone.

The advantage of this approach is twofold. First, it remains understandable and therefore verifiable by the customer since the description is based solely on the environment knowledge. Second, if the entity behavior is to be modified, only the part related to this entity is concerned.

-C- Global Approach (activity based)

When the two above approaches do not easily lead to a correct and complete functional specification, a more global approach based on an activity decomposition of system is necessary. This may be the case for complex entities and systems. When the activity refinement seems sufficient, each activity can be modeled again by the two previous approaches.

The main difficulty lies in correctly identifying the appropriate activities. For that, it is advised to search for different viewpoints. This approach is recommended in most “functional” methodologies based on the data-flow diagram. The resulting quality of the specification is strongly dependent on the specifier’s analysis, and often the activity decomposition is carried out too far, therefore mixing specification and design tasks.

13.3 Selection of Viewpoints or Projections

The quality of a functional specification is intimately dependent on the choice of viewpoints and the complete and non-redundant coverage of the whole system description by these viewpoints. Some more explanations are useful. For that, Figure 16-a represents a system linked to three entities of its environment. Two viewpoints are shown with the “eye” symbol. The first one (1) corresponds to the vision of the system behavior depending on stimuli generated by the entity E1. Let us suppose that this viewpoint also leads us to completely describe the system behavior for the entity E3. The second viewpoint (2) describes the behavior of the entity E2 controlled by the system.

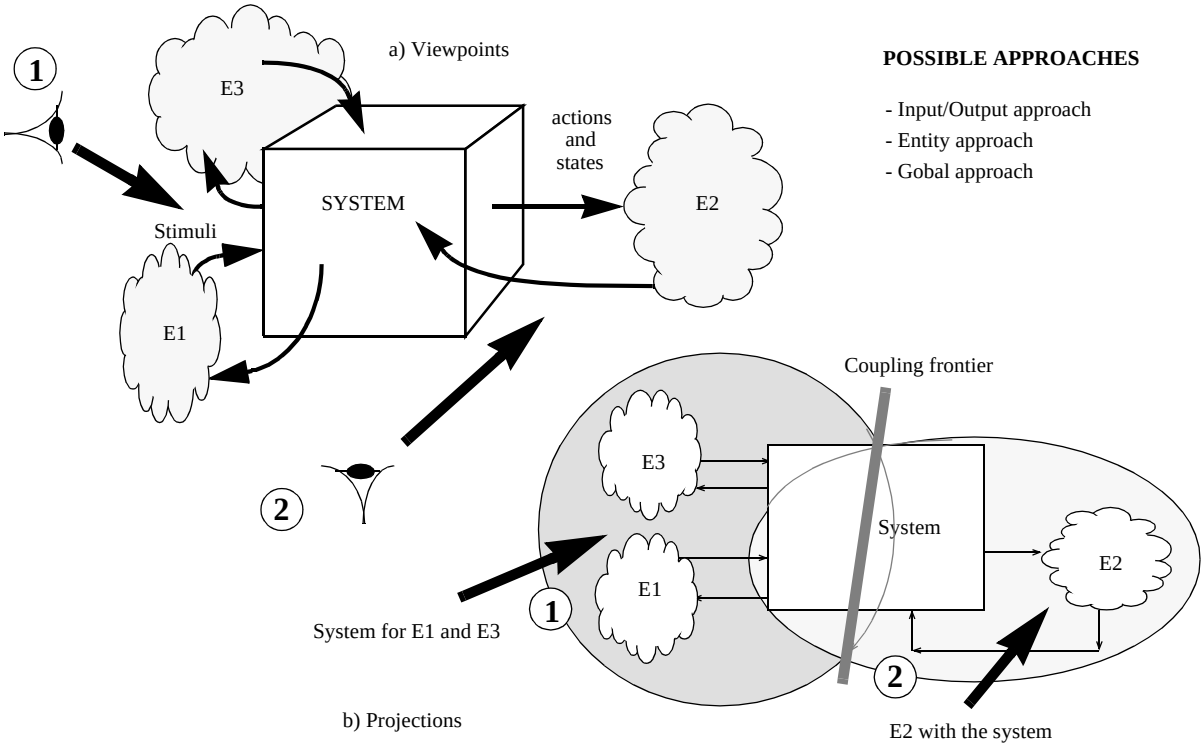


Fig. 16. Importance of viewpoints to express system behavior.

Figure 16-b shows the two resulting projections. Each gray or stripped area represents the coverage of each separate viewpoint modeling. Area (1) shows that the two entities E1 and E3 are completely covered but that the system is partially covered. Area (2) shows a more important coverage of the system except for the upper left corner. Together, the two areas completely cover the system but with redundancy. This is a result of considering the two viewpoints independently. Consequently, redundancies often exist.

To obtain a complete specification of the system without redundancies, all the system must be covered and superpositions must be removed. The solution is to identify a coupling frontier between all zones. For a system, this frontier corresponds to specific data, events and/or information items useful to link all zones. Searching for the most appropriate coupling elements is essential because, usually, these elements are necessary and sufficient to completely and

correctly describe the behavioral system specification. Necessary to express the specification, these elements will also be essential later during the design step to express the functional solution. A criterion is necessary to determine whether a coupling element is relevant or not. This character is decided by considering whether it is possible or not to model the system at the specification level without this element. We agree that such an analysis is not completely obvious the first time. Experience on different examples progressively shows the relevance of this advice.

13.4 Method to express Functional Specifications

Functional specifications include the enumeration of the service functions of the system, then the behavior of the system for these functions.

-A- List of System Functions

This first part of the functional specification phase is useful to list all functions the system must provide for its environment. These functions are normally indicated in the requirement document. Each function is described by a title, a brief description of its role, and the entities of the environment concerned by this function. It is important not to confuse internal and external functions. Experience has shown that it is naturally easier to evoke internal functions, implying a confusion between specification and design. Another common mistake is to consider too low-level functions. For complex systems, it is advised to hierarchically classify and structure functions and distinguish between main and secondary functions.

-B- System Specifications for these Functions

We have previously indicated three useful approaches to express the complete behavior of the system. These three approaches are not mutually exclusive. On the contrary, each of them explains a partial and different view of the system. The main objective of a system specification is to express all its input/output relations. Therefore, whenever possible, the input/output approach is the most efficient.

The entity approach is especially appropriate for real-time control systems. Describing the behavior of each entity in compliance with the system is made easier if its model exists without the system.

When neither one of these two approaches is efficient, it is better to consider a global approach of the application. A data-flow diagram describing all the application activities is expressed to demonstrate the role and therefore the functionalities of the system. A decomposition into subsets deduced from this analysis later leads to the use of the two previous approaches.

To correctly specify, the first step is to define the most appropriate viewpoints to best cover the whole system. Next, for each viewpoint the specifier has to select the appropriate specification model. For discrete-time models, the specification is obtained in an incremental manner by adding states, links, transitions, actions and by successive state refinements.

-C- Activity Diagram

The third dimension of the specification model concerns structuring the system activities. Remember that an activity is a concept to encapsulate a set of operations and actions which are functionally coherent and linked to its environment through a set of inputs and outputs.

Two approaches lead to a system activity diagram:

- the representation of all viewpoints used to express the system behavior,
- the direct search of system activities in the case of a global approach.

The first case consists of a synthesis of the functional specification task. Each behavioral model is considered as the description of an activity. Couplings between activities are directly deduced from coupling elements of the various viewpoints. Figure 17 gives the activity diagram corresponding to Figure 16. The two viewpoints lead to the two activities A1 and A2. The link between both is the coupling used to describe data and time dependencies between behavior diagrams.

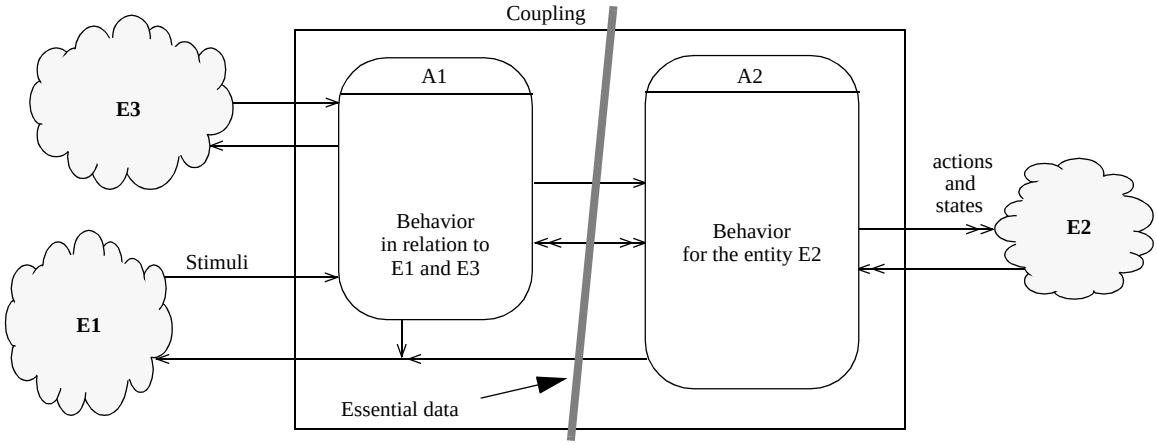


Fig. 17. Example of a simple system activity diagram.

In the second case, the activity diagram is an intermediate model used to make the behavioral modeling easier by reducing the complexity. The decomposition follows a hierarchical stepwise refinement. An activity refinement is stopped when its behavior is expressible. What is most important is not the path followed during the analysis, but that the quality of the resulting specification be understandable and verifiable by the customer.

13.5 Specification Verification and Validation

The objective of the specification step is to obtain a complete and coherent specification document, verified and accepted by the customer.

The first way of verifying is to organize a writer-reader cycle to review all the documents. The readers must correctly represent the participants concerned by the result, namely the purchaser and the users of the system. For reasons of efficiency, at least three reviews have to be scheduled: the first one after analyzing and modeling the environment, the second one after having written the functional specifications, and the last one after the completion of the whole specification in order to correct and eliminate all errors and omissions.

Another complementary means of verification and validation implies the use of specific tools to verify static and dynamic properties. It means that a part of the specifications have to be described in an executable manner. For that, the model described here is formal enough to use computer-aided tools for simulation. A translation into VHDL is one of today's possible solutions.

14 Specification Method applied to an Example

In this section we consider a case study which is quite simple to understand but complex enough to illustrate all the interesting features of the specification model and method. Because the solution to this problem is an appropriate mixture of hardware and software to correctly satisfy performance constraints, the developed specification is an interesting document input to the Hardware/Software CoDesign process [7][35].

14.1 Description of our Case Study

For the chosen example, the required goal is to design and prototype a distributed communication system obtained by assembling many similar boards. On each board, producers have to send short messages or packets (256 bytes max.) to consumers located on the same board or on other boards. Producers and consumers are software tasks. A 20 Mbits/s serial bus called TransBus [4] is used to interconnect boards. The system requirements and the bus specification are shown in Figure 18. The bus includes 4 lines: a DATA line as the data medium (bit protocol similar to transputer link protocol except for acknowledgement), an ACK line for acknowledging each transmitted byte, and 2 lines (TokenIn and TokenOut) to manage the bus access according to a hardware token ring. The protocol at the byte level is quite similar to the transputer link protocol. Each message includes the address of the consumer, the length of the data part and then the data.

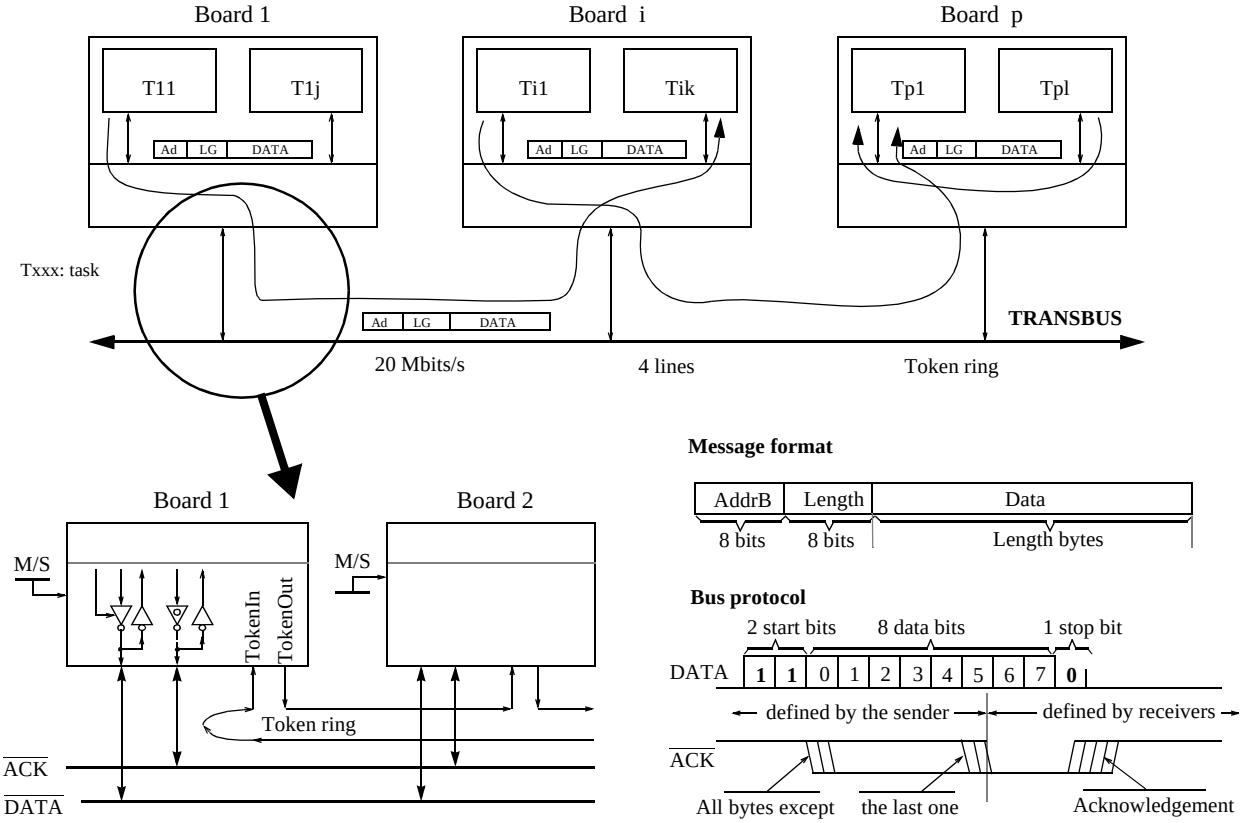


Fig. 18. Requirements of the communication system.

At any moment, only one board must own the token. The token owner can then send a message, and only one if it has requested an exchange. If there is no request or if the transmission is complete, the token is given to its neighbor which, in this case, is able to send one message. Without messages to send, the token moves very quickly (100 ns per board). The token is implemented as a boolean signal and all the boards are wired as a circular shift register. To initialize the token ring correctly with all identical boards, one of them (the first) is wired as the master, the others as slaves (meaning of the M/S input).

The address field is used to designate the corresponding board. A board receives only messages bearing its address. The specific address 0 is used to broadcast the message to all boards. The ACK line is essential to enslave the sender to all other boards which are in the receiving state. To achieve that, each receiving board must acknowledge each byte. The ACK wire is driven by open collector outputs, so as to allow an AND function on acknowledgements of all concerned receivers. The

acknowledgement specification is a four-phase protocol. Since the length of the message is variable, a board has to recognize the end of each message in order to consider the next byte as the address field of the next message. To do so, the sender marks the last byte by signaling it on the ACK line only during bit 5 instead of bit 0.

The TransBus is a simple but efficient bus developed in our laboratory initially to interconnect up to 255 transputers on it (case study developed in [6]). Currently, we are using it to implement hard real-time systems and efficient monitoring techniques to collect event traces for analyzing real-time multiprocessor architectures.

For this case study, the objective here is to define the specification of the system corresponding to one board. Performance constraints also have to be specified for message throughput and producer-consumer transmission latency.

Another constraint is imposed at the application level. It concerns the protocol for each message transfer which has to implement a mailbox of fixed capacity (constant parameter N) on the consumer side so as to satisfy an asynchronous communication protocol. Message-flow control is necessary to avoid deadlocks of the whole system with the bus in the busy state.

The first step of the MCSE methodology is to write the complete specification of one board of the communication system. Most of the technological constraints are given in Figure 18. To correctly design the board at a system level and not at a physical level, it is necessary to describe the functional specification which we consider as specifications at the system level. Such specifications do not include information about the bus characteristics.

One criterion to measure that the resulting specification document is correct and complete is to consider whether it could be transmitted to a team of designers who would be able to develop the product without discussing it with specifiers.

14.2 Analysis of the System Environment

The first phase of the specification step consists in analyzing the problem at a physical level and then translating this view at a functional level.

-A- Physical Analysis

The objective is to build a system by interconnecting similar boards on which producer and consumer tasks run (Figure 18). A producer can send a message to any consumer of the system through a port which acts as a fifo buffer. A port is linked to each consumer. Message transfers can take place inside a board or between boards. In this case, the TransBus serial 20 Mbits/s bus has to be used. The number of tasks on each board is undefined but limited here to 255. Generally speaking, a task may be a producer and a consumer relating to the other tasks.

The physical analysis identifies the following objects: the whole system; the board, including each producer and/or consumer task as part of the user's application, in addition to the hardware and software parts needed to manage these tasks; and the Transbus. Figure 19 represents the aggregation and relationship model of physical components and identifies the part to specify.

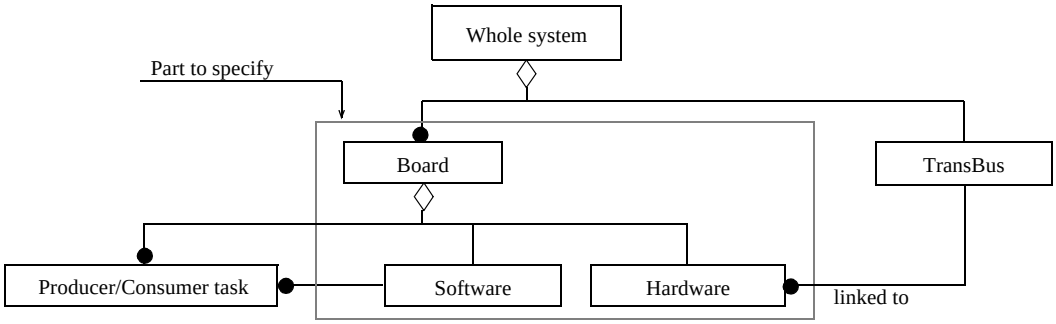


Fig. 19. Object decomposition of the problem.

Objects are necessarily interconnected. Figure 20 shows each object with its inputs and outputs and object interconnections. BoardSupport is an object including the hardware and the software excluding the application tasks. Object inputs and outputs are also specified by the data model notation. Therefore, BusInOut is a bidirectional link of permanent data type (double arrow) which symbolizes the state of the four wires (Data, Ack, TokenIn, TokenOut) of the TransBus. MessProd and MessCons are information links (single arrow) whose data structure is defined by DefMess. Each message imposed by the transbus protocol includes the board address, the length of the data part (and in this data part for each board the task address), and the message value.

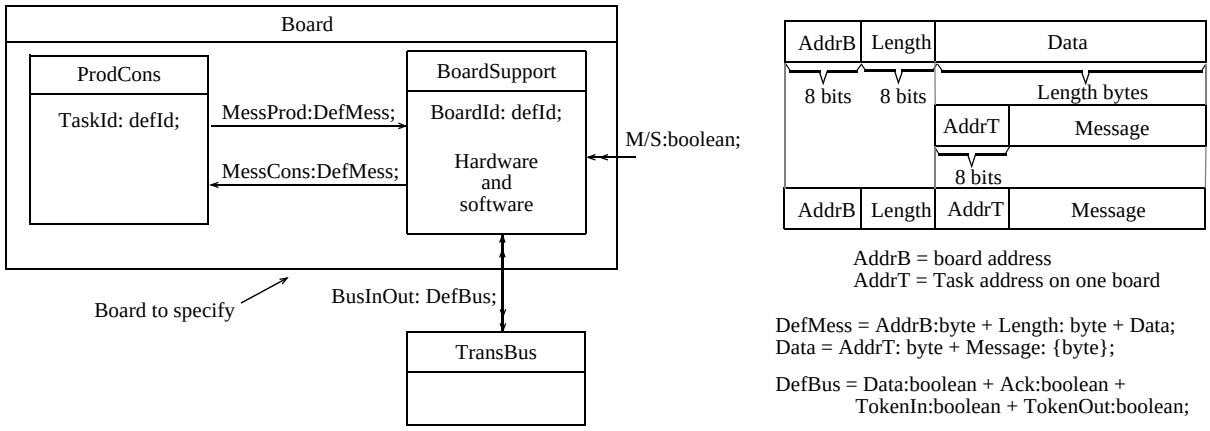


Fig. 20. Interconnection diagram of physical objects.

Notice that the number of ProdCons tasks is arbitrary. The notation in Figure 20 concerns object classes and not object instances. The objects ProdCons and BoardSupport are respectively characterized by the attributes TaskId and BoardId.

-B- Functional Analysis

The functional view of the application shown in Figure 21 is derived from the previous analysis by removing all technological characteristics. The TransBus link is replaced by the two functional links ComOut and ComIn. The definition of DefMess is changed to keep only a functional meaning: TaskId identifies the addressee task and Mess is the user data. TaskId is the task identification among all the application tasks. The input M/S is removed because it is useful only to initialize the bus token.

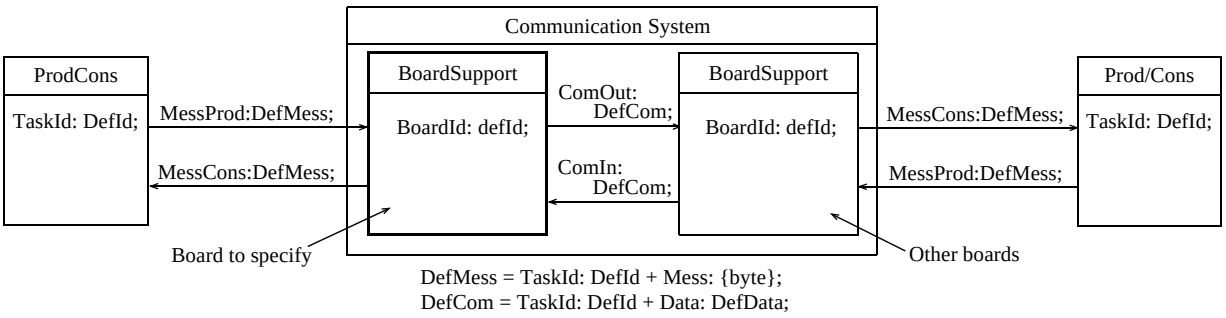


Fig. 21. Interconnection diagram of functional objects.

Only one kind of entity or object class exists in the environment of the part to specify. Figure 22 represents its behavior. the entity may wish to send or receive a message at any time but not both concurrently.

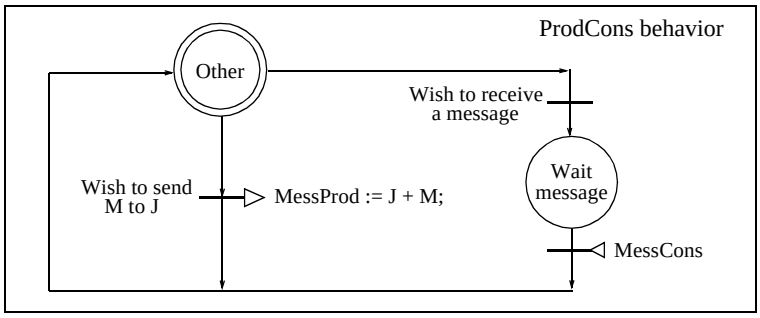


Fig. 22. Behavior of the ProdCons object or entity.

14.3 Delimitation of the System and its Inputs and Outputs

The appropriate delimitation of the functional system to specify with its inputs and outputs is deduced from the functional analysis. Here all objects (instances) or entities are represented to correctly identify the category of each input and output: single link or link array. In Figure 23, N entities ProdCons are considered. On the other side, the system (board to design) is considered directly connected to P other boards. The functional interconnection of all boards is chosen to be the most general because each board is directly connected to all the others. In this way, every technological solution, including the TransBus, is potentially possible.

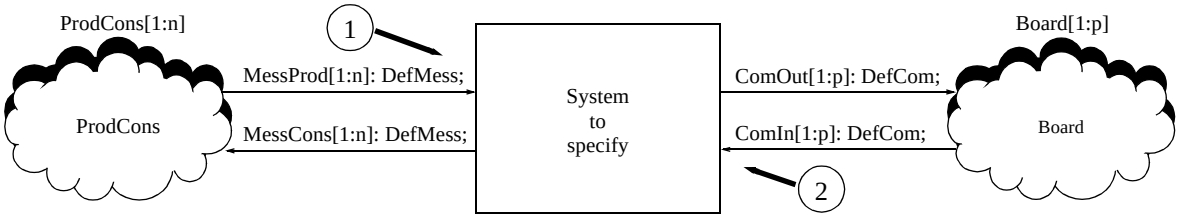


Fig. 23. Functional delimitation of the system and its inputs and outputs.

14.4 Functional Specifications

The only service function of the system is to send each message of MessProd[1:n] to the addressee task. This function can be globally described by:

MessCons[MessProd[i].TaskId] := MessProd[i]; ∀ i in 1..n.

This high-level description appropriate for ProdCons entities is not enough to specify the functional behavior of the system for at least two reasons: the inputs ComIn and outputs ComOut are not used, and there is a constraint on the space capacity of the input port MessCons[i] of each entity ProdCons. This second reason needs a deeper analysis of the system behavior.

The specification method we explained before consists of searching for the appropriate viewpoints in order to completely describe the behavior of the system at the functional level. Two viewpoints are quite obvious here (see Figure 23):

- the behavior of the system in relation to each ProdCons providing a message,
- the behavior of the system in relation to other boards providing a message.

These two viewpoints correspond to an input approach of the system. The entity approach is not useful here because the ProdCons behavior is known and the behavior of other boards is also what we have to specify.

-A- System Behavior for each Input MessProd[]

The modeling is of stimulus/response type. We have to explain what the system must do for each input message coming from each task ProdCons. The notation [] designates the current or considered element in the array [1:n], and its identity is Me. [:] is used for a complete generic or unconstrained array.

Two different cases are to be considered. The first situation corresponds to the case of an available space in the input port MessCons[] of the addressee task; the message can be sent. The other situation (no available space) implies that the producer must wait for a space before sending the message. Yet the destination port may be on the same board or on another board. In this last case, in order to know if a space is available, the board needs to send a message with the space request question to the destination board and must then wait for the answer. A more complicated situation appears when many tasks may want to send messages to the same consumer at the same time. To avoid errors, i.e. allocation of the same space in the port, the question message will mean the allocation request of a space in the consumer's port and the answer will indicate the allocation effectiveness.

Figure 25 describes the behavior of the system for each entity ProdCons.

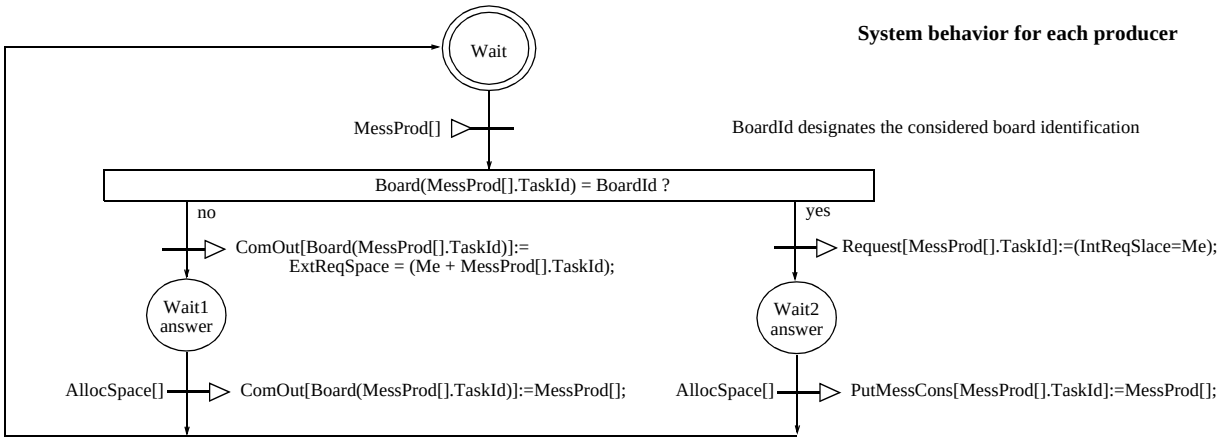


Fig. 24. System behavior for each producer.

This specification needs to use the event AllocSpace[] and the information item Request. The type definition of information item ComOut has to be enhanced with the ExtReqSpace message. *Me* is a specific value indicating the current producer identity or index, the value necessary to receive the allocation answer AllocSpace[]. Board(TaskId) is a function able to provide the

designation BoardId of the board running the consumer task TaskId. This implies a routing table or a specific encoding of TaskId. Here we will use the physical coding of the messages given in Figure 20. PutMessCons is used to release a message in the port MessCons.

-B- System Behavior for each Input ComIn[]

There are two different kinds of incoming messages. The first one includes a data message to a specific consumer, the second one is a flow-control message to request a space in a consumer port or to answer that a space has been allocated.

Figure 25 describes the behavior of the system for each message input ComIn[].

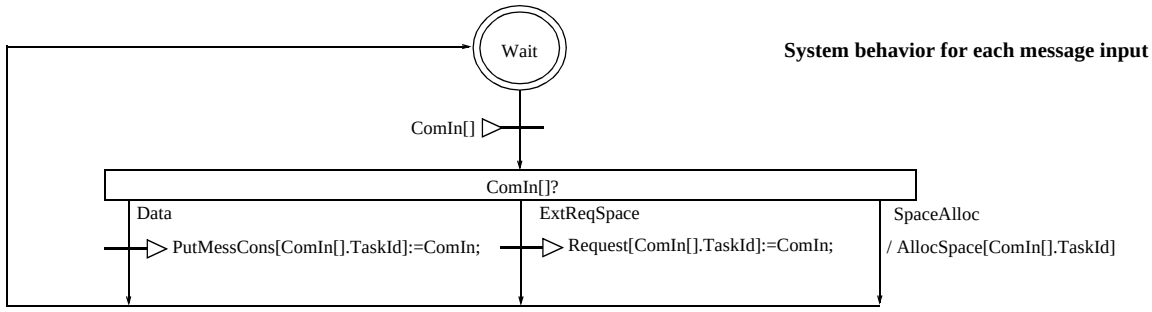


Fig. 25. System behavior for each message input.

In the case of ExtReqSpace, ComIn[].Me is the identity of the producer asking for a space.

These two viewpoints are not enough because some events and information items are not completely defined. The management of each port MessCons[i] is also missing. This corresponds to a modeling of the system for each output MessCons[].

-C- System Behavior for each Output MessCons[]

Two different cases need to be considered. The first case corresponds to available space in the port; an answer is directly sent back. In the other case, the caller has to wait for a space.

Figure 25 describes the behavior of the system for each port MessCons[].

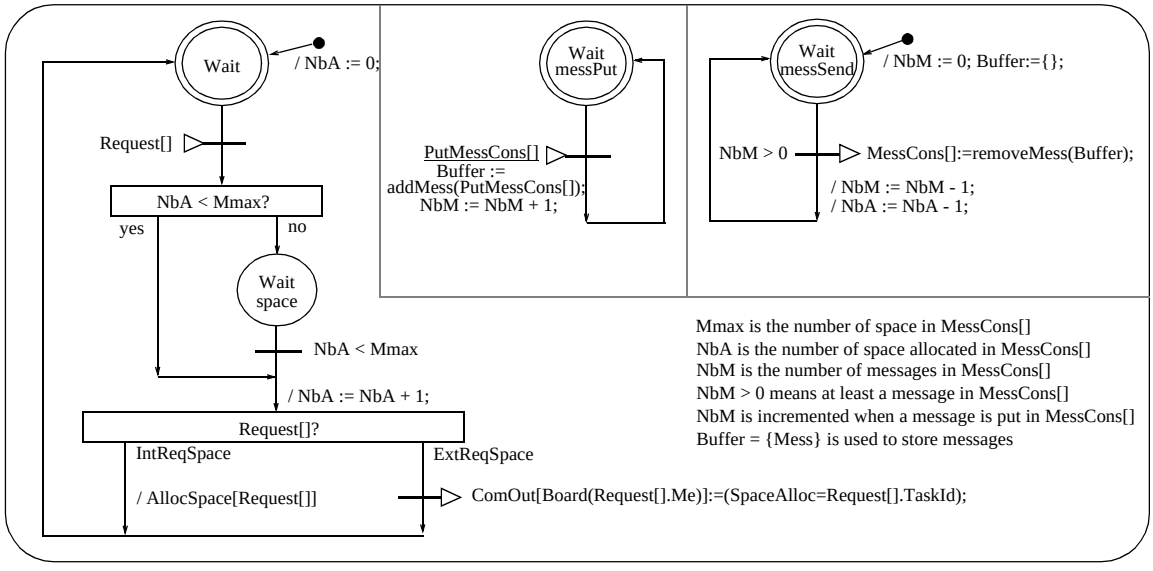
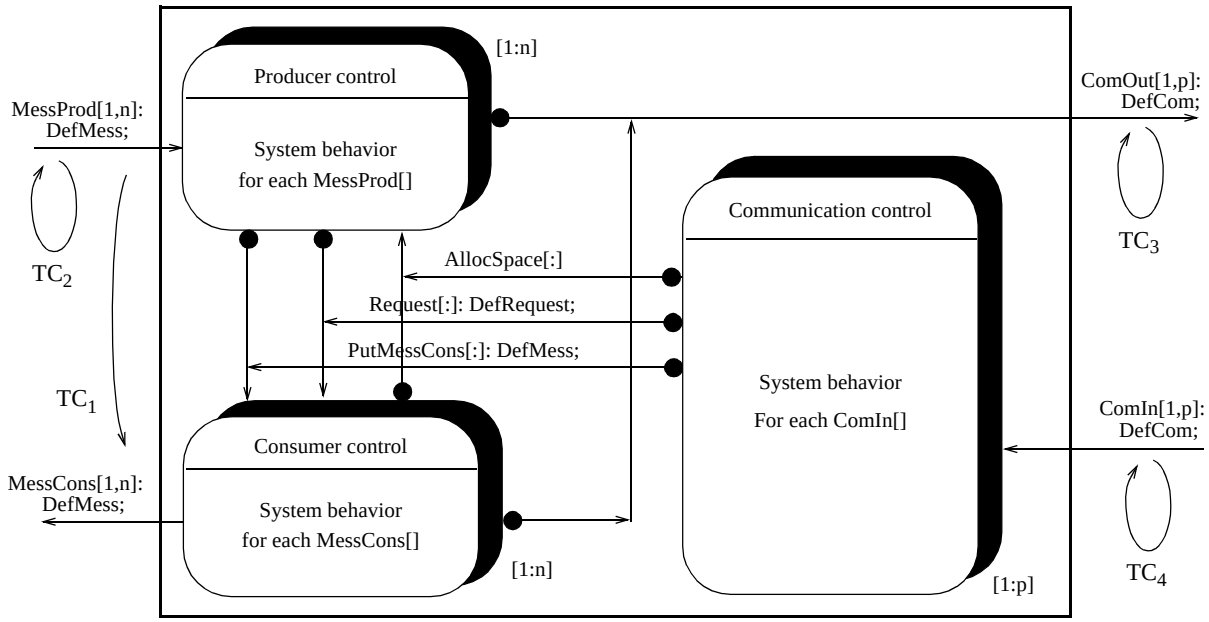


Fig. 26. System behavior for each port MessCons[].

The right-hand part of the statechart explains the sending of each message to ProdCons[] according to the rendez-vous protocol. The left-hand part explains the space allocation for each request. The middle part concerns the memorization of each message in the variable Buffer for which a space was allocated before.

-D- Synthesis: Activity Diagram

The synthesis of the three viewpoints is expressed by the means of an activity diagram shown in Figure 27. The last definition of each data type is also given.



DefMess = TaskId: DefId + Mess: {byte};
DefCom = [Mess | ExtReqSpace | SpaceAlloc];
Mess = TaskId: DefId + Data: DefData;
ExtReqSpace = Me: DefId + TaskId: DefId;
SpaceAlloc = TaskId: DefId;

DefRequest = [ExtReqSpace | IntReqSpace] & unconstraint capacity
IntReqSpace = Me: DefId;
ExtReqSpace = Me: DefId + TaskId: DefId;

Fig. 27. Activity diagram for the functional specification.

The full dot at the beginning of a link means that each source activity has access to all activity inputs pointed out by the link (useful for link arrays).

A verification has to be done at this stage. It concerns first a static check: all names are defined, all conditions and actions of a behavioral diagram are inputs and outputs of the corresponding activity, etc. More important is the verification of the dynamic aspect. Especially here, remember the meaning of information transfer with the output triangle and the input triangle. This transfer is of rendez-vous type. It means that the first activity or object ready for the transfer has to wait for the other.

A specific situation must be noticed in Figure 25. On a Request message, when no space is available, the activity has to wait in the WaitSpace state. In that case, it cannot receive a next Request message, which may suspend the evolution of the Communication Control activity therefore blocking the reception of all messages from other boards. In this specific case, it is useful to release the rendez-vous protocol for the Request link. Therefore, Request is specifically defined as an unconstrained message capacity link.

14.5 Operational Specifications

Operational specifications define constant values, definition domains of data and performance constraints. For constant values, the task number on each board and the number of boards are limited to 255. The message space Mmax of each consumer port is modifiable from 1 up to 255.

Performance constraints are easily defined on the activity diagram. From the application viewpoint, i.e. the message transfer functionality, some requirements may be important. We may want to limit the maximum time for each message to go from the producer to the consumer (constraint TC1 in Figure 27). One may be interested in the producer throughput (constraint TC2). These constraints are significant only when considering that the consumer task is very fast in order to always have a free space in the consumer port.

One may also be concerned by the message throughput between boards: maximize TC3 + TC4.

The verification of these constraints entails some difficulties. The context of verification has to be clearly defined with the customer. For example, we may consider 6 boards with only one producer task and one consumer task on each board. A consumer task only receives messages. All producers permanently send messages of 10 byte-length alternatively to each consumer.

14.6 Technological Specifications

Technological constraints for this problem have been partly described in the presentation of the example. We analyze some of them in this section.

-A- Geographic Distribution

The whole communication solution is a distributed system. But the problem here is the design of one board. Therefore, each board does not have geographic distribution constraints. If the problem were considered at a higher level and for the whole system, such a constraint should have been imposed.

-B- Physical Interface

The use of the TransBus to interconnect all boards is imposed. TransBus is a bus concept as well as a specific physical interface with electrical constraints (open-collector and tristate outputs) and timing constraints (duration accuracy of each bit, waveform of the Ack wire). A lot of information is given in Figure 18. In order to avoid some misinterpretations, it is necessary here to completely specify the bus protocol. This is also an argument to show that the recommended specification models and the method are useful even at the technological level.

Three parts of the TransBus interface must be specified:

- the bus arbitration and allocation with the token bus technique,
- the message transfer and so each byte transfer on the bus,
- the message format on the bus.

We refer to the physical objects of Figure 20. The first analysis concerns the bus allocation which leads to the specification of the behavior of TokenOut in relation to TokenIn and M/S. Remember that M/S is used to correctly initialize the bus by allocating the token to a first board. Figure 28 shows the board interconnection and the behavior of each board for the token. This behavioral model results from an output approach (TokenOut). Token defines the bus allocation state.

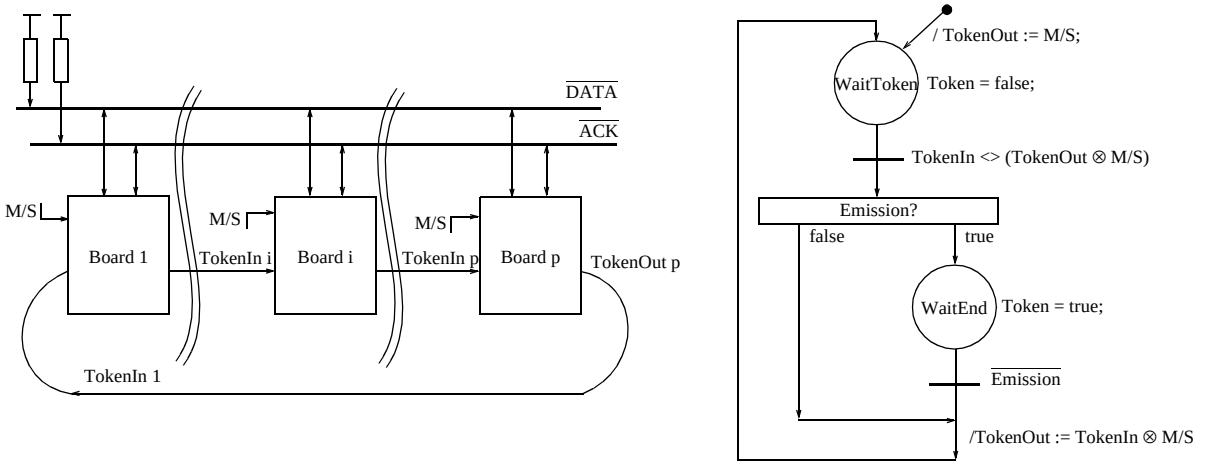


Fig. 28. Behavior for the bus allocation.

For a board, TokenIn and TokenOut are respectively the input and the output. When a board has no message to send (boolean variable Emission), the token is transmitted to its neighbor; otherwise, the token is kept until the end of the message transmission.

The next part to specify concerns a message transmission on the bus. This second viewpoint must specify the evolution of Data and Ack and must explain the variables Emission and Token. When a message ComOut is to be transmitted, Emission becomes true and the board must wait for the variable Token. Then the message is sent byte after byte. The message length is ComOut[2]. During the transmission of each byte, the Ack wire has to be correctly controlled.

Figure 29 describes the board behavior during a transmission. To be correctly specified, the specification is given synchronous to an event Clk20 corresponding to a 50 ns period, the duration of each bit. Ack = null means an inactive open-collector level. Data = null means a high-impedance output. This specification needs to be complete, and so is a formal transcription of the waveform given in Figure 18.

The reception of a message, and therefore of each byte, could be described in the same manner. However, we believe that this is not necessary here because, with the specification of the message transmission, the specification of the environment for the reception is well defined; hence, we only give the behavior in order to receive a complete message in Figure 30.

A message can be received only if its address field AddrB (first byte) is 0 or equal to the board identification BoardId and if the message does not come from the board (Token=false). The macro-state ReceptionByte provides the received byte (Byte) on its output and a boolean variable indicates whether it is the last byte of the message or not. This indication results from the observation of the Ack line during the bit 3 reception. A received message is transmitted on the ComIn link which is the input of the functional level.

The synthesis of this specification for the bus interface is given in Figure 31. If another kind of interface has to be considered - Ethernet cable, shared memory, I^2C bus, etc. - only this part of the solution has to be changed. This a proof that the functional specification is technology-independent.

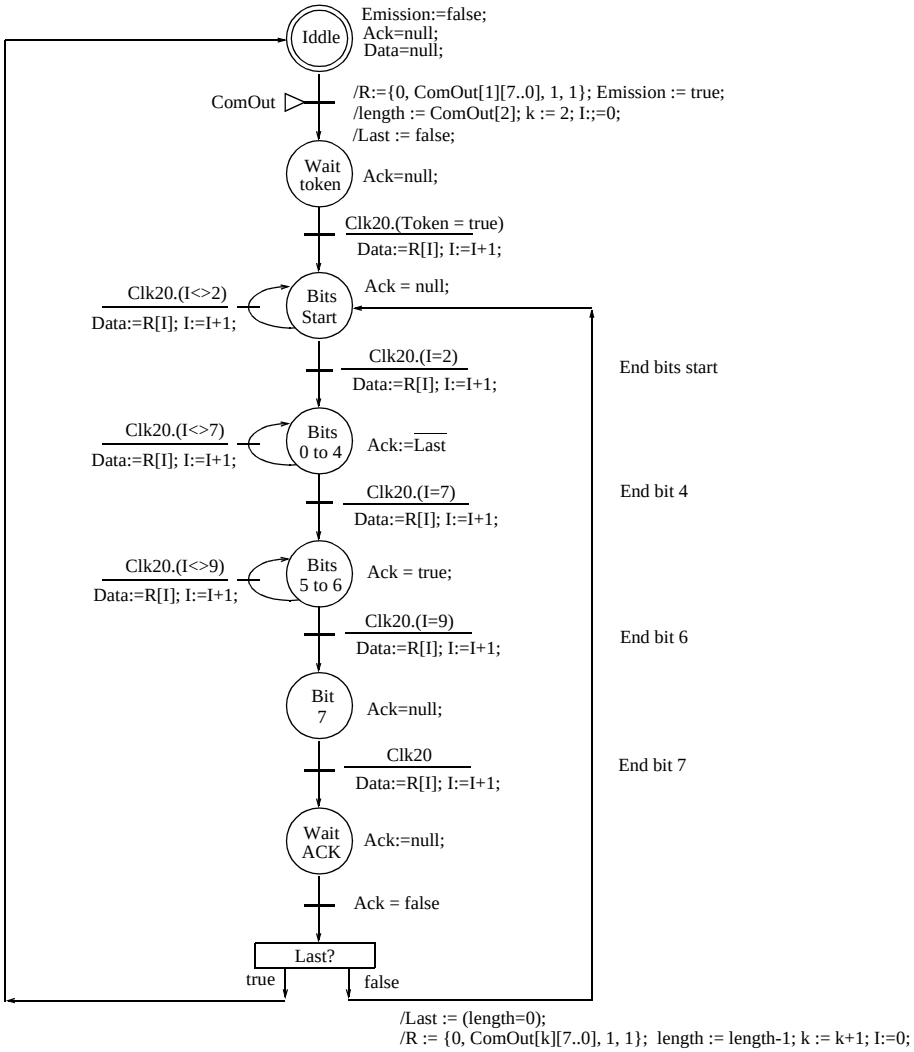


Fig. 29. Behavior for the transmission on the bus.

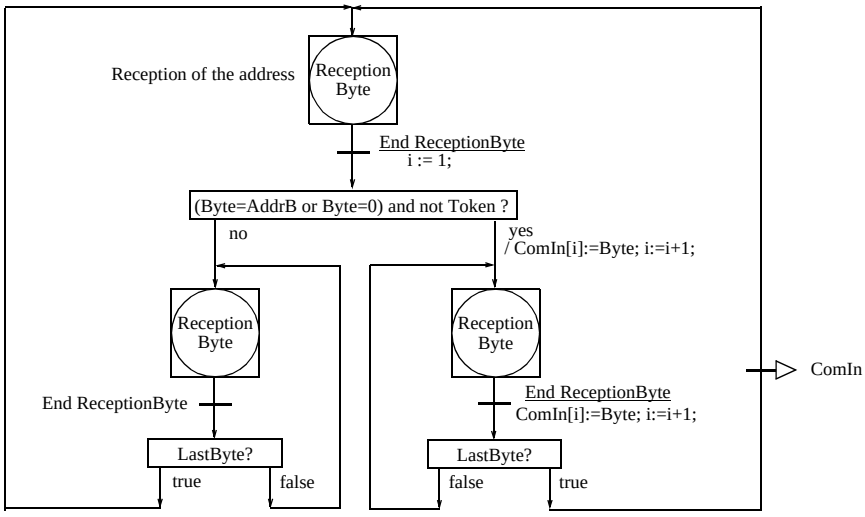


Fig. 30. Behavior for a message reception.

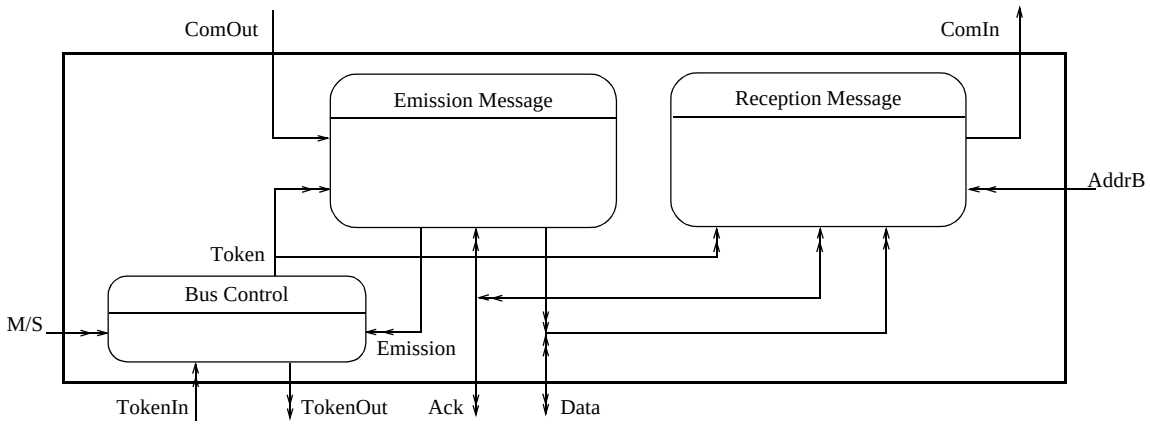


Fig. 31. Activity diagram for the TransBus specification.

The last part of the specification is the message format on the bus. It is a translation of the functional types of messages (DefCom) into a physical frame. This is an easy task not resolved here.

Concerning performance constraints at the physical level, it may be requested to maximize the bus utilization. This implies two requirements. First, during a message transmission, waiting for the acknowledgement must be avoided (only one stop bit between two successive bytes). Second, the delay between two following messages must be the smallest possible. Timing constraints must also be added to specify more precisely the bus signals (rise time, fall time, and Clk20 period tolerance, etc.).

-C- Implementation Specifications

Different kinds of constraints can be added to the previous specifications. Here we give some of them.

For example, the hardware solution is imposed: the use of a 68332 microprocessor from Motorola to run the software and the use of a FPGA or EPLD for the connection to the TransBus. The constraint is to reduce the cost of the board. But high performance also has to be met. Therefore, the designer has to find the best solution.

The user of the board will write his application tasks in the C or C++ language. To send and receive messages, two procedures must be provided to link the user's application to the communication software implemented into an EPROM. One Mbytes of RAM will be available for the user.

A test mode is to be implemented on the board. This means that only one board must be fully tested. To do so, TokenIn is connected to TokenOut and the board must receive the message it sends. The software program for testing must be supplied.

14.7 Remarks on this Example

This example was given to demonstrate the use of the specification model and the recommended method to lead to a complete specification document. The reader may think that such a detailed specification is already a part of the design. It is not so. The given specification is only a behavioral outside view of the system to design. Activity diagrams are only results and are useful to represent the behavior of different outside viewpoints in order to verify the completeness of the specification.

To persuade oneself, one can try to define the internal solution. A correct and efficient design for this problem is not easy. The first problem is how to represent the design solution; the second one is what design process to follow in order to find an appropriate technology-independent solution. The MCSE methodology, and specifically its second functional design step, is suitable to answer these questions: the use of the functional model to describe the solution, and how to start the functional decomposition based on essential internal variables or items given in the activity diagrams. It is not the objective of this paper to show the design deduction.

One interesting aspect of the proposed specification model is that it is an executable model. This means that if a computer-aided tool exists to capture this specification, the semantics of the model is complete in order to enable its execution. In comparison with existing tools for specification (Statemate [20], SpeedChart, etc.) only some specific features have to be added: the difference between permanent data and events, the addition of data structure specification and of objects with an interface, the addition of specific synchronization meaning between behavioral diagrams, and the use of the activity diagram only as a structural model.

15 Specific Features of the Specification Model

The following analysis argues from the specific features of the specification model recommended in the MCSE methodology. At first glance, the reader may think that the recommended model is well known because it is the same as those described in functional methods (Ward, Hatley) and also in object-oriented methods, specifically OMT (Rumbaugh). It is true that the three models, one for each viewpoint, are the same: object and data model, data-flow diagram, and StateChart. Differences in meaning and so in the semantics of these models exist in our proposition to enhance the quality of the resulting specification and to make the search for such a specification and then for the design solution easier. These differences are highlighted hereafter.

-A- Importance of Data or Item Classification

The analysis of an application begins with the identification of objects, data variables, items, and attributes. Each of them has a name and a definition type. We consider it essential to clearly differentiate a data with a permanent meaning and the occurrence of an event or an information item from the beginning. Their use would later imply major differences. A permanent data always owns a value; therefore it may be read and modified at any time. This is not the case for an event which is available only on its occurrence implying time dependencies.

This difference is not considered in other methods except in [34] in which the two kinds of links (single or double arrow) are recommended in the DFD model. We also advise distinguishing an event from an information item. The first has only one meaning; the second carries a data value or, more generally, a content. This difference is useful to deeply express object or entity or activity behaviors.

-B- Difference in Object Meaning

The word *object* is often used but with different meanings not always clearly defined. In object-oriented methods, every component is an object. We do not use this meaning in our specification method. Objects are only active objects of the problem and located in the system environment. Other components found during the analysis are data, items, attributes, and inputs or outputs of active objects. The difference is observed by the number of objects which is much lower than in an object-oriented analysis. This method is often named object-based analysis [22]. To avoid any confusion, we prefer to use the word *entity* instead of *object*.

With such a conceptual difference, we do not give up the object-oriented qualities. Therefore, the generalization/specialization concept (and so the class and inheritance concepts) and the aggregation/decomposition concept are strongly recommended. But these concepts are not used for data modeling. A more structural and syntactic model leads to better readability and simplicity.

Another important difference lies in the fact that an object for us and for the specification only, is not defined as an encapsulation unit including methods and operations available for its environment. This does not mean that these operations do not exist inside. The full meaning of an object with its methods is correct only for design and implementation. Contrary to the object-oriented model, we find essential to consider object inputs and outputs. It is rather surprising to note that input and output definition is not so explicit. We refuse to consider that the method or operation definition for objects is similar to our object input/output definition and we consider that such a definition is too early and too specific during the specification step. If this point of view is acceptable for software, it is not the case for system specification. Object inputs and outputs are the means of data and item exchanges between objects for us. Because of the active nature of an object, it may be specified by one or more behavioral models.

-C- Different Meaning of the Activity Diagram

The activity diagram used in this paper has the same meaning as the data-flow diagram but with a slight difference in notation. The permanent data link (double arrow) always carries a value; so, it is equivalent to the file or repository concept in the DFD notation.

Contrary to the SART model, we do not add the control-flow diagram to specify the time evolution of activities. We assume that all activities viewed from the outside are always potentially active. It is the inside view when described with a behavioral model which expresses the active and inactive states of each activity. The modeling process is therefore clearly different from the SART process recommended by Ward and Hatley. In the latter case, the behavior of one activity is partly described inside (the activity mini-spec.) and partly outside (the CFD).

-D- Different Meaning of temporal Dependencies

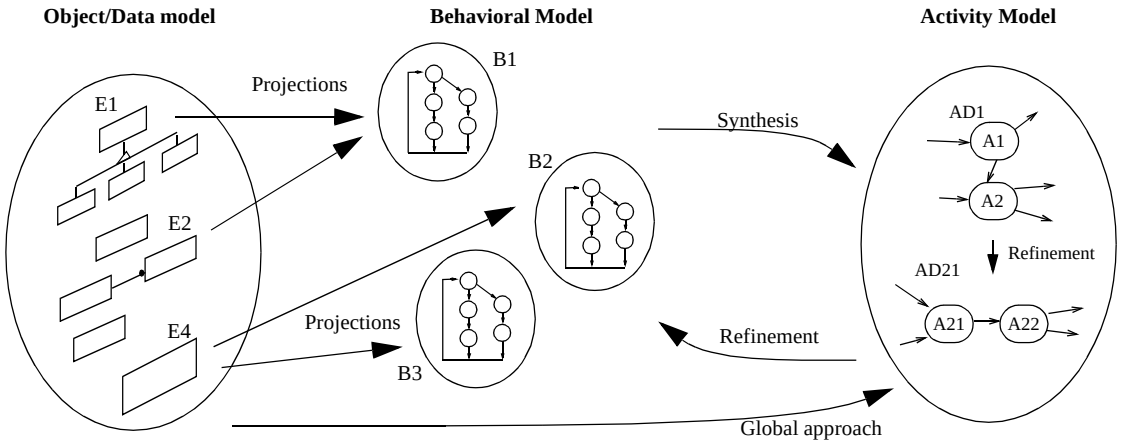
The behavior is expressed according to the StateChart model for its main qualities: hierarchical description and concurrency. For this model, temporal dependencies are of a synchronous and broadcast type. This means, on the one hand, that a transition firing, actions on that transition and actions of the new state are executed at the same time (hypothesis of atomic and instantaneous execution); on the other hand, it means the result of an action is broadcast to all conditions on transition.

All state/transition-based models do not have the same meaning. For example, in the SDL model [3], temporal dependencies between diagrams are of asynchronous and point-to-point type. So, information items and events are memorized in an implicit file of the signal link waiting for the transition firing condition.

In our notation, and specifically because we clearly distinguish between the occurrence of an event and the occurrence of an information item, the temporal dependency differs. Both are of broadcast type, but the event type dependency is without memorization, and no rendez-vous and the information type dependency implies a rendez-vous between the source and all receivers. Dependencies on permanent data are the same as for the StateChart. In comparison with the StateChart, we limit the association of actions only to transitions and to states (not on Entry and Exit of a state) for readability.

-E- Importance of the Specification Process on the Result

Formalizing a description model for functional and non-functional specifications is necessary but not sufficient. In fact the specifier's problem is to efficiently translate user requirements into a complete specification. Added to the description model(s), a method is essential. But a method is necessarily linked to the underlined model characteristics or properties. The model we propose here is efficient in order to describe the result, but also to obtain it because the model corresponds to the natural manner specifiers and designers view the environment components, the system itself and the whole application. The model is also useful to derive an internal functional solution of the system later during the design step. Figure 32 represents the recommended process.



Object/Data Model = Data dictionary + Objet model diagrams + Object interconnection diagrams
Behavioral Model = State/transition diagrams
Activity Model = Data and Event-flow diagrams + non-functional constraints

Fig. 32. Representation of the MCSE specification process.

-F- Ability to express Performance Constraints

Writing functional specifications is not enough. Non-functional requirements exist for every system to be designed. In all these requirements, which are not easy to specify, performance constraints play an important role for reactive, real-time and embedded systems. Correctly defining performance specifications and then their verification and validation is often a real problem because such constraints are written in a textual form. We have shown that the recommended model can be enhanced with more formal performance constraints and that the model is a good basis for an exhaustive analysis of all possible constraints by considering all component inputs and outputs of the specification both at the functional and the physical levels.

16 Conclusions

In this paper we have described the specification model and method recommended in the MCSE system-level specification and design methodology. A complete specification includes functional specifications and non-functional ones which are operational and technological specifications. An example has been developed to illustrate the models and to help readers understand the recommended process. This example is an interesting case study for CoDesign. It may be used to evaluate the quality of specification models and methods and also CoDesign methods as well as CASHE tools.

The specification model is based on the well-known three-dimensional modeling space. Compared to models recommended in the functional analysis method and the object-oriented analysis method, few differences exist. Probably the main one is between an event flow and permanent data flow. We have also enhanced the model to express performance constraints.

The main difference concerns the specification process or method and the quality of the resulting specification. First of all, we agree with the usefulness of the object-based modeling to begin the analysis of the whole application. The representation of objects with their inputs and outputs and their interconnections are of valuable help at the system level. Differentiating modeling at the physical level and at the functional level is essential to increase the ability to specify complex systems and later to efficiently design them.

For small and medium size problems, the MCSE approach leads the simplification the modeling because it avoids data-flow diagram searching. This results in at least two advantages. First, a stop criterion for DFD decomposition is not needed because the DFD is a synthesis result of the behavioral modeling. The functional approach shows very often a too detailed refinement of the specification and a confusion between specification and design. Second, the search for a solution during design is easier and the resulting solution is more efficient. For complex problems, a global and hierarchical approach is indispensable, leading to the use of the aggregation/ decomposition technique. At some level of decomposition, each sub-system may be modeled according to the behavioral viewpoint.

By using the functional approach, the behavior of a component is distributed on different levels of DFD and in different data-transformation and control-flow activities. There results a poor readability of the specification. In the MCSE approach, the activity diagram is only a structural model to encapsulate behaviors. Each component can be considered separately.

Relative to the object-oriented approach, our approach leads to a reduction of the complexity of the set of models. The data model is simple and the number of objects in an application is limited to the active components of the problem. The behavior modeling with MCSE requires more reflection but limits the number of behaviors.

Very often, specification methods describe how to represent and find functional specifications. Only of few of them describe what non-functional specifications are and how to find and formalize them. In this paper, we have formally defined performance constraints and have linked them to the functional specification. All other constraints such as dependability constraints, physical and electrical constraints, and human/computer interfaces, etc. are included in technological specifications.

The quality of a specification can be evaluated using different criteria. The first one is the quality of the document at the end of the specification step because it is the main communication interface between the designers and the customer. The method described above gives the structure of such a document and makes its writing and reading easier. Other qualities are the completeness, the accuracy and the exactness of the specification. Therefore, a specification must be verified. One necessary means is to use a writer/reader cycle process. It is important to add a more efficient way which consist in simulating the specification. This is possible because the model is formal enough to be executable which allows those involved to test it very easily. The recommended model and method can also be supported by efficient tools. Last but not least is the ability of a specification to be translated into an efficient functional design and then an implementation. The next step of the MCSE methodology is precisely provided in order to deduce an internal functional organization of the system to be designed from the activity diagrams.

Acknowledgements

The author would like to thank C. Galais and O. Pasquier for their constructive comments on this paper.

References

[1] R.J. Abbott, D.K. Moorhead, Software requirements and specifications: A survey of needs and languages, The Journal of Systems and Software, No 2 1981, pp 297-316

[2] R.J. Abbott, An Integrated Approach to Software Development, WILEY Interscience Publication, John Wiley&Sons 1986

[3] F. Belina, D. Hogrefe, The CCITT-Specification and Description Language SDL, Computer Networks and ISDN Systems, No 16 1988/89, pp 311-341

[4] J.P. Calvez, O. Pasquier, A TRANSputer interconnection BUS for real-time systems, TRANSPUTERS'92 , Arc-et-Senans, France, May 20-22 1992, M. Becker et al., Ed. IOS Press, pp 273-283

[5] J.P. Calvez, Embedded Real-Time Systems. A Specification and Design Methodology, John Wiley 1993

[6] J.P. Calvez, ASICs Specification and Design, Chapman&Hall, March 1996

[7] J.P. Calvez, A CoDesign Case Study with the MCSE Methodology, to appear in "Design Automation of Embedded Systems", Special issue on "Embedded Systems Case Studies", Kluwer Publisher, 1996

[8] P. Coad, E. Yourdon, Object-Oriented Analysis, Yourdon Press, Computing Series 1990

[9] B. Dasarathy, Timing constraints of real-time systems: constructs for expressing them, methods of validating them, IEEE transactions on Software Engineering, Vol SE-11, No 1 Jan 1985, pp 80-86

[10] A.M. Davis, A comparison of techniques for the specification of external system behaviour, Communications of the ACM, Vol 31 No 9, Sept. 1988, pp 1098-1115

[11] A. M. Davis, Software Requirements. Analysis & Specification, Prentice-Hall 1990

[12] T. DeMarco, Structured Analysis and System Specification, Yourdon Computing Series, Yourdon Press, Prentice-Hall 1979

[13] M.E. Fayad, W-T. Tsai, R.L. Anthony, M.L. Fulghum, Object Modeling Technique (OMT): experience report, Journal of Object-Oriented Programming, Nov-Dec 1994, pp 46-58

[14] R.G. Fichman, C.F. Kemerer, Object-oriented and conventional analysis and design methodologies, Computer, Oct. 1992, pp 22-39

[15] U. Furbach, Formal specification methods for reactive systems, Journal of Systems Software, Vol 21 1992, pp 129-139

[16] D.P. Gajski, F. Vahid, S. Narayan, J. Gong, Specification and Design of Embedded Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1994

[17] J. O. Grady, System Requirements Analysis, McGraw-Hill, 1993

[18] N. Halbwachs, Synchronous Programming of Reactive Systems, Kluwer Academic Publishers, 1993

[19] D. Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming North Holland, Vol 8, 1987, pp 231-274

[20] D. Harel, H. Lachover, et al., Statemate: A working Environment for the development of complex reactive systems, Proceedings of 10th International Conference on Software Engineering, Singapore 11-15 April 1988, pp 122-129

[21] D.J. Hatley, I.A. Pirbhai, Strategies for Real-time System Specification, Dorset House Publishing New York 1987

[22] W. Haythorn, What is object-oriented design, Journal of Object-Oriented Programming, March-April 1994, pp 67-78

[23] K. J. Hughes, R. M. Rankin, C. T. Sennett, Taxonomy for requirements analysis, Proceedings ICRE, first International Conference on Requirements Engineering, IEEE Computer Society Press, 1993, pp 176-179

[24] I. Jacobson & al, Object-Oriented Software Engineering. A Use Case Driven Approach, Addison Wesley 1992

[25] M. Jackson, System Development, C.A.R Hoare series, Prentice-Hall 1983

[26] R.W. Jensen, C.C. Tonies, Software Engineering, Prentice-Hall International Editions 1979

[27] S. Narayan, F. Vahid, D.D. Gajski, System specification with the SpecCharts language, IEEE Design&Test of Computers, Dec 1992, pp 6-13

[28] I. Pyle, P. Hruschka, M. Lissandre, K. Jackson, Real-Time Systems. Investigating Industrial Practice, John Wiley, 1993

[29] D.T. Ross, Applications and extensions of SADT, Computer April 1985, pp 25-34

[30] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall 1991

[31] S. Shlaer, S.J. Mellor, Object lifecycles, Modeling the word in States, Yourdon Press 1992

[32] B. Thomé (Editor), Systems Engineering. Principles and Practice of Computer-based Systems Engineering, John Wiley, 1993

[33] T.H. Tse, L. Pong, An examination of requirements specification languages, The Computer Journal, Vol 34, No 2, 1991, pp 143-152

[34] P.T. Ward, S.J. Mellor, Structured Development for real-time systems, Vol.1: Introduction and Tools, Vol.2: Essential Modeling Techniques, Yourdon Computing series, Yourdon Press, Prentice-Hall 1985

[35] W.H. Wolf, Hardware-software Co-Design of embedded systems, Proceedings of the IEEE, Vol 82, No 7, July 1994, pp 967-989