Formal specification method for systems automation

Jean-François Pétin, Gérard Morel, Hervé Panetto

EJC file no.: **EJC 039/2004**

Formal specification method for systems automation

Jean-François Pétin, Gérard Morel, Hervé Panetto

Université Henri Poincaré Nancy I
Centre de Recherche en Automatique de Nancy, UMR 7039, CNRS – UHP – INPL
Campus scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex, FRANCE
Tel : +33 (0)3 83 68 44 43, Fax : +33 (0)3 83 68 44 59
{jean-francois.petin}{gerard.morel}{herve.panetto}@cran.uhp-nancy.fr

ABSTRACT. *Currently automatic control deals with the theoretical modelling techniques applied to formally define the behaviour of a control system when the system goals and the process behaviour to be controlled are well defined. Although these approaches are efficient in the design and implementation phases for controlling the dynamics of automatized systems, other tools are also required in the early stages of the process of engineering a system. This paper deals with a specification method aimed at proving that the system goals, as required by the users, are formally refined towards the real target automation system with completeness, consistency, unambiguousness and correctness guarantees. Our specification method is based on the B language to globally verify, from formal constructs, the predicate: Control Systems Requirements $\wedge$ Process Systems Requirements $\Rightarrow$ Production System Requirements. A case study illustrates our approach and opens issues on the way to industrial practice.*

KEY WORDS: *Systems Engineering, Automation Engineering, Formal specification, Requirements analysis, Formal verification*

1. Introduction

Industrial automation systems are embedding Information Technology (IT) intensively to achieve increasingly complex applications. Industrial automation standards, such as the safety-related IEC 61508[1], strongly recommend the use of formal methods to control the complexity of software-intensive applications and their related ease-of-use design techniques [POL 04][MOI 03].

Conceptual and practical approaches have been widely explored by organizations related to computer sciences and automatic control: examples are software verification [CRA 95], symbolic [CLA 00], timed and probabilistic [PAR 04] model checking, or automatic synthesis [CAS 99]. However, in spite of the consensus that early phases of a system definition are the most important in ensuring that the target system will satisfy the user's requirements, most of these models and tools address the design and implementation phases.

Over these later stages, many systems engineering and automation engineering practitioners [SHE 01][JOH 04] consider that the time is ripe to formalize the earlier stages of specification. This means providing a set of guidelines, generic constructs, and formal verification tools to establish a non-ambiguous, correct, consistent and complete model of understanding of what a system has to do according to the users' requirements before designing its behaviour according to the multiple engineers' practices and techniques [LHO 99].

Writing formal specifications for a given system may be based on an *a posteriori approach* using automatic verification techniques, such as model checking [CLA 00], as the mean of checking the link between the resulting specification and the required properties. Writing formal specifications may also be based on an *a priori approach* where the proof is used to progressively refine an abstract specification with simple properties to be checked into more and more tangible and precise models. In such a system *proof-oriented specification*, the models are related by a *refinement relationship* that is able to ensure specification correctness and consistency by preserving properties through the model transformations.

Considering that the specification activity reflects the heuristic capabilities of human intuition, the process of specifying a system should be able to reuse acquired knowledge to facilitate elicitation of requirements and model definition. To this end, a *model-driven specification* of automation systems is based on the definition of reusable patterns and constructs that are generic for well-identified problem classes and whose completeness and non ambiguity have been established once and for all.

[1] IEC 61508, *Functional safety of electrical/electronic/ programmable electronic (E/E/PE) safety-related systems*

This paper combines these two forms of reasoning to propose a formal specification method for automation systems where:

– correctness and consistency of the models are ensured using a *proof-oriented specification* that allows the system goals to be progressively refined and split into process/control sub-systems models while preserving the link between the formal specification and required properties (goals);

– completeness and non ambiguity of these specifications are supported by a *model-driven specification* that promotes the reuse of acquired knowledge in the definition of specific artefacts for the specific automation domain.

Our work is first founded on Fusaoka's automation predicate [FUS 83], which postulates that the design of automation systems consists in defining the (unknown) control rules of the (known) dynamics of a physical system, starting from the behavioural (known) goals to be met:

$$\textit{Control Rules} \wedge \textit{Dynamics} \supset \textit{Goal (1).}$$

This predicate provides formal guidelines that strengthen the proposed formal specification method for automation systems by defining logical relationships between system goals specifications, process specifications, and control specifications [LAM 01] covering the various points of view involved in automation engineering (dynamics, behaviour, information, communication, etc):

$$\textit{Control Specifications} \wedge \textit{Process Specifications} \supset \textit{System Specifications (2).}$$

After a brief overview of models and languages that approximately fulfil these requirements above, Section 2 introduces the B language as an efficient formalism to support *model-driven specification* and *proof-oriented specification*. A formal specification method based on the B language is then presented: B instantiation mechanisms are proposed to reuse formal automation constructs and correspondence between B objects and mechanisms, and mathematical operators of the predicate (2) are given to support the proof-oriented specification. This section ends with the description of a case study that is used to illustrate the method.

Section 3 proposes an illustration of a *model-driven specification* through the formalization of generic constructs for physical processes modelling. Generic rules that help in establishing a complete and non ambiguous process specification are given and applied using the case study.

Section 4 applies, using the same case study, a *proof-oriented specification* to refine the system goals into process/control sub-systems models while preserving correctness and consistency with regards to the initial properties. A global overview of this specification is given in the appendix using UML notation.

The final section provides some conclusions and open issues to be resolved in order to put this method into industrial practice.

2. Formal specification for systems automation based on the B language

Our objective is to propose a formal automation method that combines a proof-oriented specification, based on an incremental reasoning, with a model-driven specification, based on the reuse of generic constructs. To support this approach, we require a formal abstract language that provides support for property verification, proved refinement, specification composition, and instantiation.

2.1. *Candidate models and languages for a formal specification method*

We briefly describe candidate models, methods, or languages that could be more or less able to cover these various requirements. Formal languages and methods are based upon mathematical models that allow a precise formulation of the properties a model has to satisfy and provide proof mechanisms that allow the verification of these properties.

In the area of automatic control, and more precisely as far as the Discrete Events Systems are concerned, several approaches have been explored for:

– the definition of provable models in the design phase, such as analysis techniques for Petri Nets or Finite State Automata [CAS 99], theorem proving and model checking [ABR 91][CLA 00], or control synthesis [RAM 87];

– the verification of PLC programs by applying formal techniques to check the properties satisfied by controllers implemented using IEC 61131-3 programming languages [CRA 95][FEL 99][ROU 02].

Even if these approaches have been proved to be efficient in the design and implementation phases, two common characteristics make them inadequate for the specification phase:

– expression of the underlying mathematical representation is limited to the modelling of system dynamics, and hardly covers the description of other system properties as required for users' requirements and automation constructs modelling;

– most of these formalisms, except Petri nets, do not support incremental modelling, such as formal refinement mechanisms.

At higher level of abstraction, systems engineering approaches or unified languages, such as UML,[2] have proven, in practice, to represent useful views of systems as a result of static, dynamic, and functional diagrams, at different phases of the development cycle. Moreover, the concept of knowledge reuse is included in the UML object oriented approach because it contains extensibility mechanisms, called "UML profiles", which can be used to tailor it to specific domains. A "profile" may be defined as a *"specification that specialises one or several standard UML meta-models, called "reference meta-models"*. On this basis, Model Driven Architecture[3]

[2] UML (2003). *UML 2.0 superstructure specification*, ptc/03608-02, OMG, www.uml.org.

[3] Model Driven Architecture is an OMG trademark, http://www.omg.org/mda/.

[MEL 04] provides a set of guidelines, generic knowledge, and IT constructs to make the definition of IT systems functionality as independent as possible from any technology platform.
Even if UML provides a wide notation toolbox that covers the requirements in terms of expressiveness equally as well for the system specification as for the modelling and reuse of automation constructs [PAN 05], it suffers from a lack of methodological guidelines and formal semantics. Efforts have been made towards UML formalization [DUP 00][LED 02][LAL 02] and formal extensions, such as RT-UML [SEL 98] or UMsdL[4]. However, methodological rationales, which should underlay the modelling process with UML, such as an incremental reasoning based on the refinement of the models, are still unresolved issues.

2.2. *The B language*

Introduced by Abrial [ABR 96], the B Method is a formal method for the specification, design, and implementation of software applications that supports properties proofs. An abstract B model consists of a section defining the mathematical structures related to the problem to be solved and a section containing elements of state variables, operations, and invariance properties of the model. Proof obligations are generated from the model to ensure that properties are effectively met. A model is assumed to be closed, and this means that every possible change of state variables is defined by operations.
The B language is founded on:

– set theory with classical set operators ($S \cup T$, $S \cap T$, $S \subset T$, $x \in S$, $\#S$), function and relationship ($A \leftrightarrow B \hat{=} \mathbb{P}\, A \times B$);

– first order logic with classical operators of a 2-valued proposal logic ($\neg P$, $P \vee Q$, $P \wedge Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$) and quantifiers ($\forall X \,.\, p$, $\exists X \,.\, p$).

A B model is defined by the structure shown in Figure 1.

```
MACHINE m
SETS s
CONSTANTS c
PROPERTIES p
VARIABLES x
INVARIANT I(x)
INITIALISATION init(x)
OPERATIONS
        O1= Pre P1(x) Then S1(x)
        …
        On = Pre Pn(x) Then Sn(x)
END
```

Figure 1. *B formal model*

[4] UMSDL European ITEA project n° 99028, ITEA office, www.itea-office.org.

A model has a name m; the clause SETS contains definitions of sets of the problem; the clause CONSTANTS allows the designer to introduce information related to the mathematical structure of the problem to be solved; and the clause PROPERTIES contains the effective definitions of the constants. Another point is that sets and constants can be considered similar to parameters.
The second section of the model defines the dynamic aspects of the state variables and properties of variables using the invariant. Operations *O1 … On* are used to describe state variable modifications due to generalized substitutions; *Si(x)* contains the new value of variable *x* after the execution of operation *Oi*. The substitution of a variable is feasible with respect to its *pre-condition* or *guard*. The invariant *I(x)* states that variable x is always in a given set of possible values, which is assumed to be initialized with respect to the initial conditions and which is preserved by any operation of the list of operations. Conditions of verification, called *proof obligations*, are generated from the text of the model, and they express underlying hypotheses required for the preservation of invariant properties:

$$\text{(INV1) } Init(x) \Rightarrow I(x)$$

$$\text{(INV2) } I(x) \wedge P(x) \Rightarrow I(S(x))$$

(INV1) states the initial condition that should establish the invariant. (INV2) should be checked for every operation *Oi* of the model; it states that starting from a situation where the precondition of Operation *Oi* and the invariant are verified, the variable transformation leads to a new value of *x* where the invariant is still preserved. The B proof mechanism is founded on rule-bases using an inference engine supported by the Atelier B tool[5]. Several proof techniques are available, but the proof tool is not able to automatically prove every proof obligation, and interaction with the proof tool is required. Each proven obligation enriches the set of known theories and can be used for other proofs.

The *refinement calculus* is used to relate models at varying levels of abstraction by enriching variables and operation descriptions while preserving the already proven invariants. We summarize the approach as follows:

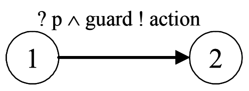
(M1,G1) refined by (M2,G2) refined by …. refined by (Mn,Gn)

Mi is the ith model iteration that satisfies the goal Gi and the goals of lower iteration number. The *refined by* relationship ensures the preservation of goals, but the proof of refinement must be given. This means that if a new model is derived from *(Mn, Gn),* we should prove that the new model refines the previous model and preserves the new properties. Consider a refinement of the machine shown in Figure 1 given by *variable y, invariant J(x,y), operation Oi(y)* $\widehat{=}$ *Pre Qi(y)* then *Ti(y)*, where *J* represents the formal link between the abstract variable *x* and the refined variable *y* and some local properties over *y*. Refinement proof obligations are generated from the following predicates for each operation Oi:

$$I(x) \wedge J(x,y) \wedge Qi(y) \Rightarrow Pi(x) \wedge J(Si(x), Ti(y)).$$

[5] Atelier B is a product of ClearSy, http://www.clearsy.com.

The B language has proved its efficiency for software system development because of its powerful refinement mechanism that supports a complete process of engineering from specification to code, and because of the expressiveness that makes possible the B description of various system views, such as dynamical [LAN 96] (see Figure 2) or informational [LAL 02]. Applying the B language for automation engineering has been addressed by academic [LAN 00][PET 98b] [MOR 04] as well as by industrial trials [LEC 02][AIT 02].



```
MACHINE G1
SETS STATE, T /* sets of states and events*/
VARIABLES st1, st2, guard, p, s
INVARIANT st1 ⊂ STATE ∧ st2 ⊂ STATE
OPERATION t(s,p) =
        PRE guard ∧ s ∈ st1 ∧ p ∈ T
        THEN st1 := st1 – {s} || st2 := st2 ∪{s} || action
END
```

Figure 2. *B formalization of a state-transition diagram [LAN 96]*

2.3. *System model-driven specification using the B language*

The objective of model-driven specification is to formalize generic users' and engineers' expertise to provide a collection of generic constructs for a specific application domain. These constructs include, within a meta-model, syntactic and semantic building blocks derived from theoretical representations, or results from experience, as well as their assembly rules. Specification models can then be obtained (see Figure 3) by instantiating proved constructs, considered as modelling generic primitives, into a model with a well-defined form (syntax) and meaning (semantics). These generic constructs can aid establishing the goals, process, or control specifications of the predicate (2).

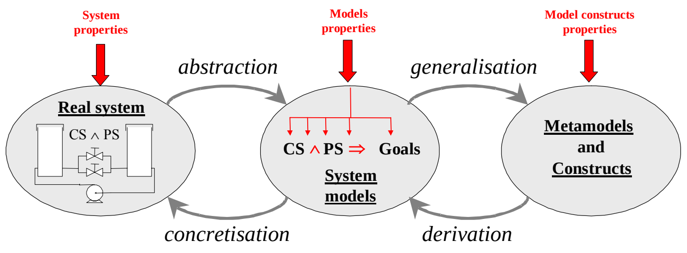


Figure 3. *Model-driven specification process*

Generic guidelines in the area of systems engineering [IUN 03] can be applied to ensure the completeness and non-ambiguity of the goals specification. Indeed, according to these guidelines, an abstract system model can be said to be composed of elementary processors provided with four kinds of information or material flows, stating what the system has to do, when the system has to do it, what it is required to be able to do it, and what the system requires to know how to do it. In this context, generic constructs result from the formalization of elementary processors and their flow typology. Instantiating these constructs requires filling and identifying all connected flows and, consequently, helps in ensuring the completeness of the resulting specification.

In the same way, the specification of process systems should take advantage of using the Paynter's classification [PAY 61] as generic modelling guidelines for the physical variables and rules, to be sure of obtaining a model that is compliant with the physics of a process, expressed in term of causality or material and energy flows balance. Section 3 details the formalization of such expert knowledge.

Finally, as far as the control specification is concerned, methodological expertise and approaches, such as structuring functional models through automation objects covering control, monitoring, and technical mode management issues [LHO 96] or action-based language for controlling cell manufacturing [WRI 88], can be used as inputs for the formalization of control generic constructs that ensures the completeness of the control specification.

Formalization, using the B language, of these constructs is based on the definition of the domain constants and properties through the static part of a B machine, on the modelling of generic behaviour through the dynamical part of a B machine and, finally, on the identification by experts of some invariant of the domain. Note that these invariants' properties can be local to a given construct, but can also define groupings of constructs and rules for valid groupings of constructs.

Reusing the formalized constructs to establish a specification model requires defining an instantiation support for the B language. We propose a two-step mechanism. First, the collection of generic constructs is prefixed with a specific name associated with each instance; this leads to a collection of machines associated with each object involved in the specification to be established. The resulting specification is formalized within a single B machine built as a network of interconnected constructs that call the instantiated constructs using the clause *EXTENDS*. For example, a generic *construct* machine is instantiated into *instance1.construct* machine, which is called by the specification M1 machine through the clause, *EXTENDS instance1.construct*. The consequence of using the *EXTENDS* clause is that all the instantiated machines inherit from the clauses constants, sets, variables, invariants, initialisations, and operations defined in the generic construct machine. More precisely, the instantiated constructs verify the same invariants as those defined in the generic constructs. Moreover, if the constructs' invariants contain the description of grouping rules, the specification model can be presumed to be correct with regards to these. In this sense, formalization of generic constructs efficiently contributes to the completeness and non-ambiguity of the specification.

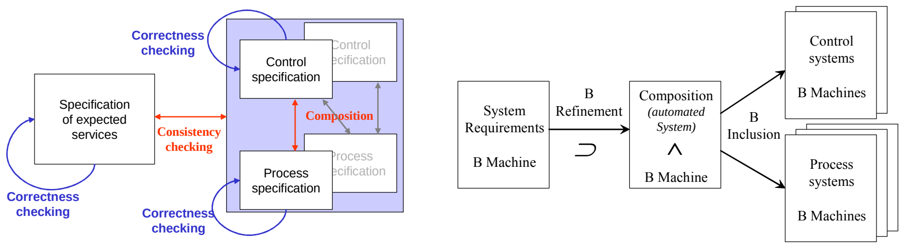
2.4. *System proof-oriented specification using the B language*

Proof-oriented specification for systems automation [MOR 04] consists of giving a formal meaning to the automation predicate operators to verify (see Figure 4a):

– the local properties of each given specification that allows confirmation of their correctness with regards to syntactic or semantics rules;

– the specifications' consistency, especially by proving that the refinement relationship between system goals on one side and target process/control systems on the other side is established.

Models involved in this predicate are formalized through B machines, whereas operators ($\wedge$ and $\supset$ operators) are established with equivalent B structuring mechanisms (see Figure 4b).



a) automation predicate b) B correspondence

Figure 4. *Automation predicate and B correspondence*

The *System requirements* machine represents all the users' requirements in terms of abstract functional behaviour expressed using static information, dynamic information processing, and invariant safety properties of a B machine. A very powerful primitive of the B language is used to support this modelling. Consider the following B operation: *operation* O(v) $\widehat{=}$ *any* w *where* P(w) *then* v: = w, where v and w are machine variables and P a predicate over w. This means that the operation result v is equal to any w if w satisfies the predicate P. In other words, we are able to describe the properties an expected result has to satisfy (what the system has to do), without describing how this result may be obtained (how the system is doing it).

The B *composition* machine formalizes the $\wedge$ operator using the B object oriented features, allowing the operations call and the use of public variables. The primitives are used to interconnect B *control* and *process* machines that describe the functional behaviour of the components involved in automation systems, such as software control, the physical process or electrical wiring

The B refinement mechanism is used to prove that the B *system requirements* machine can be correctly and consistently refined into the *composition* machine by proving that the initial invariants are preserved.

The resulting specification is then the foundation of the design and implementation phases that can be supported by the B language (for the software systems) or by dedicated techniques such as, for example, Matlab[6] [LEC 02] or LUSTRE[7] [AIT 02] which can be used for the design of the reactive control part (see Figure 5).

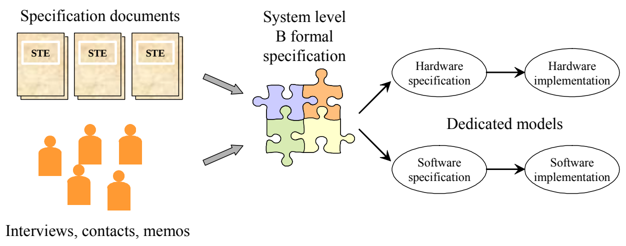


Figure 5. *From B specification towards design and implementation [LEC 02]*

2.5. *Presentation of the case study*

The proposed formal automation method based on the B language is illustrated using the case study (see Figure 6). This case study is part of an industrial demonstrator which has been developed in European projects for evaluating the interoperability of distributed intelligent actuation and measurement field-devices [PET 98a]. The studied sub-system is limited to a tank and a valve. The tank is upstream fed by a water flow and the valve is used to control the level within that tank. Meaning of the variables is the following: Q_0 and P_0 are respectively the flow rate and the pressure of water flow entering the tank, V_1 is the water volume within the tank, Q_1 and P_1 are respectively the flow rate and the pressure of water flow exiting the tank, Q_2 and P_2 are respectively the water flow rate and pressure in the circuit located after the valve and which feed the downstream system.

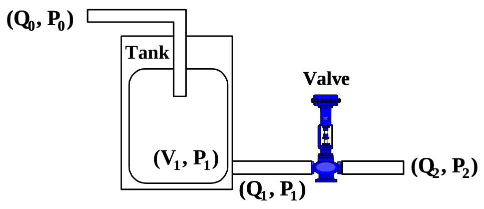


Figure 6. Case study

The model-driven specification is illustrated by formalizing generic constructs with reference to process modelling and more precisely to hydraulic systems. These constructs are instantiated for elaborating the specification model of the case study process. According to the predicate (2) and its B correspondence, a proof-oriented specification is then used to ensure the consistency between the goals' specification on one hand and the process and control specifications on the other hand.

[6] Matlab/Simulink is a product of The Mathworks company.

[7] Lustre is a modelling and programming language for synchronous reactive systems.

3. Model-driven specification of the case study

This section applies a model-driven specification process in the area of physical processes modelling. Recent work in this area [MAT 98] has highlighted the benefit of using object-oriented representations to mix theoretical modelling with identification techniques, when necessary. In this context, a physical system is modelled as a network of interconnected processors acting as white boxes in the case of theoretical modelling or black boxes in the case of identification. To facilitate their reusability, most of these processors are described independently of each other and without any causality constraints. However, when building a physical system representation by connecting these independent processors, the modeller has to take into account:

– the relationships between input and output flows of each elementary processor, which may be constrained by some physical rules of conservation (energy or flow balance, specific features of the transformations, etc.);

– the connection between elementary processors, which are physically limited to some enabled configurations (causality relationships between physical variables).

In the context of our study, the key issue we wish to address is correctness checking, with regards to physical laws, of the process structure provided by object-oriented representations. Our approach is based on the formalization, using the B method, of some expert knowledge in the area of physical systems modelling and on proven mechanisms that help in re-using the formalized knowledge. More precisely, our work refers to Paynter's classification of the physical variables [PAY 61] and its application within a systemic approach proposed by Feliot [FEL 96]. We summarize these results before detailing their B formalization.

3.1. *Theoretical foundations*

Physical variables have been classified by Paynter into four basic sets: effort, flow, impulse, and displacement. Applied to hydraulic systems related to our case study, Effort, Flow, Displacement, and Impulse variables correspond respectively to variables of pressure, fluid flow, volume, and moment of pressure. Paynter's tetrahedron of states (see Figure 7) provides the relationships between these variables.

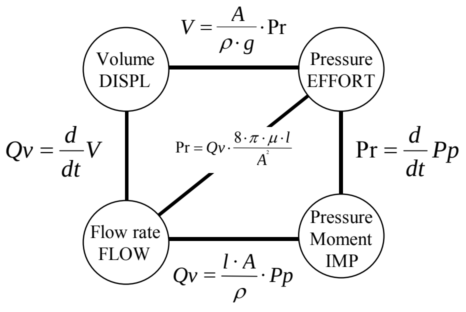

$V = \frac{A}{\rho \cdot g} \cdot \mathrm{Pr}$
Volume
DISPL
Pressure
EFFORT
$Qv = \frac{d}{dt} V$
$\mathrm{Pr} = Qv \cdot \frac{8 \cdot \pi \cdot \mu \cdot l}{A^2}$
$\mathrm{Pr} = \frac{d}{dt} Pp$
Flow rate
FLOW
Pressure
Moment
IMP
$Qv = \frac{l \cdot A}{\rho} \cdot Pp$

Figure 7. *B formal typology of state variables for physical systems*

Considering that some of those physical variables are tightly coupled, Paynter has proposed considering three main couples: (effort, flow), known as power; (effort, displacement), known as potential energy; and (impulse, flow), known as kinetic energy. From these definitions, three main physical processors ensuring the couples' transformation have been identified: from power to power (P/P), with a proportional relationship; from energy to power (E/P), with a derivative relationship, and from power to energy (P/E), with an integral relationship.

Feliot has formally demonstrated [FEL 96] that these three processors are strictly equivalent to the three basic processors proposed by system theory: (P/P), (P/E), and (E/P) processors are respectively equivalent to Shape, Time, and Space systemic operators. From the process modelling point of view, Shape, Time, and Space operators respectively support the physical transformations of a product, its storage, and its transportation. Based on system theory, Feliot has proposed some structuring rules for interconnecting these operators.

Our contribution consists in formalizing these theoretical foundations, using the B method, with two objectives:

– to provide basic constructs for the modelling of process systems and more particularly, in the context of our case study, for the modelling of hydraulic systems;

– to provide a formal framework for the modelling of a particular process system using the instantiation of the basic constructs.

3.2. *Formalisation of basic constructs for process modelling*

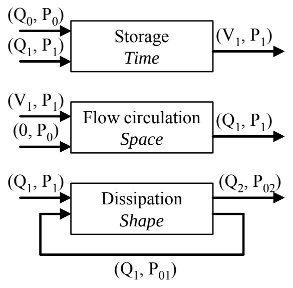
As a first level of abstraction, the systemic Feliot's processors (Figure 8a) are formalized within B machines without taking into account a precise description of the dynamics of the supported transformations. It leads to the definition of three constructs that include:

– a definition of the variable couples involved in the transformation;

– the invariant relationships that link the two variables inside each couple, these relationships are those defined by the Paynter's tetrahedron of states;

– a very abstract description of the transformation by only specifying its inputs and output, which must maintain the previous invariant.

Figure 8b presents the B formalisation of a Time processor in the area of hydraulic systems where *(Q_0, P_0)* and *(Q_1, P_1)* represent respectively the input and output fluid of the storage system with two couples (flow, pressure), whereas *(V_1, P_1)* represents the volume and the pressure within the tank. In the same way, the Shape processor has been formalized with (Flow, Pressure) as input, and another (Flow, Pressure) as the output, by taking into account the problem of energy dissipation, and the Space processor is defined as transforming (Volume, Pressure) and (Flow, Pressure) into (Flow, Pressure). Note that all the relationships described within the invariant and the operation involve some parameters, such as '*A*' (tube or tank section), which should be further tuned when modelling a particular physical system and when technical features of the various actuators, sensors, and transmitters are known.



```
MACHINE Hydraulic_Time_processor
SETS EFFORT, FLOW, DISP, IMPULSE
CONSTANTS A, ρ, g
VARIABLES V1,Q0,P0,Q1,P1
INVARIANT
   V1 ∈ DISP ∧ P1 ∈ EFFORT ∧ ... /* type checking */
∧ (V1 = (A / (ρ*g)) * P1) /* invariant relationships */
OPERATIONS (V1,P1) ← Time ((Q0,P0),(Q1,P1)) =
      PRE (Q0, P0) ∈ FLOW → EFFORT ∧ …
      THEN ANY newV1, newP1
      WHERE  (newV1, newP1) ∈ DISP → EFFORT
                 ∧ (V1 = (A / (ρ*g)) * P1)
      THEN V1 := newV1 || P1 := newP1
END
```

a) Feliot's processors *b) B formalisation of a Time processor*

Figure 8. *B formal typology of physical processors*

As a second level of abstraction, we introduce a qualitative description of the transformation dynamics by detailing the behavioural trends of the transformation (see Figure 9). Our objective is to focus on the early phase of specification by providing a well-structured process model to be used for defining control goals; therefore, this qualitative description can be said to be sufficiently efficient.

Produced Variables	Output Variation	P_0	$\Delta Q = Q_0 - Q_1$
P_1	→	→	▨
	↓	↓	▨
	↑	↑	▨
V_1	→	▨	$\Delta Q = 0$
	↓	▨	$\Delta Q < 0$
	↑	▨	$\Delta Q > 0$

```
REFINEMENT Hydraulic_Time_processor
VARIABLES Vr1, Qr0, Pr0, Qr1, Pr1
INVARIANT
    (Vr1 ∈ DISP) ∧ (Vr1 = V1) ∧
    (Qr1 ∈ FLOW) ∧ (Qr1 = Q1) ∧ ...
OPERATIONS
(Vr1, Pr1) ← Time ((Qr0, Pr0),(Qr1, Pr1 )) =
PRE (Q0, P0) ∈ FLOW → EFFORT ∧ …
THEN
   SELECT (Qr0-Qr1)>0
   THEN ANY newV1, newP1
   WHERE (newV1,newP1) ∈ DISP → EFFORT
    ∧ newV1 > Vr1 ∧ (newV1 = (A / (ρ*g)) * newP1)
   THEN Vr1 := newV1 || Pr1 := newP1
END
```

Figure 9. *B formal qualitative behaviours for physical systems processors*

3.3. *Completeness and non ambiguity checking*

Applying formal constructs to specify the physical process of the case study presented in the appendix consists of first identifying the generic constructs that have to be used. This means that the components involved in the case study process (valve, tank, circuit, etc.) must be associated with a generic processor, as defined by the constructs (hydraulic_time, hydraulic_space, and hydraulic_shape processors). Note that the same process component can be associated with several generic constructs. For example, the valve is involved in water transport as a space processor, but also provokes water flow dissipation as a hydraulic_shape processor. Conversely, a generic construct can be associated with several process components: transport, for example, is performed using valves and circuits.

The second step consists of defining the physical process model as a network of interconnected processors that are instantiated from the generic constructs (see Figure 10, a graphical representation of the network).

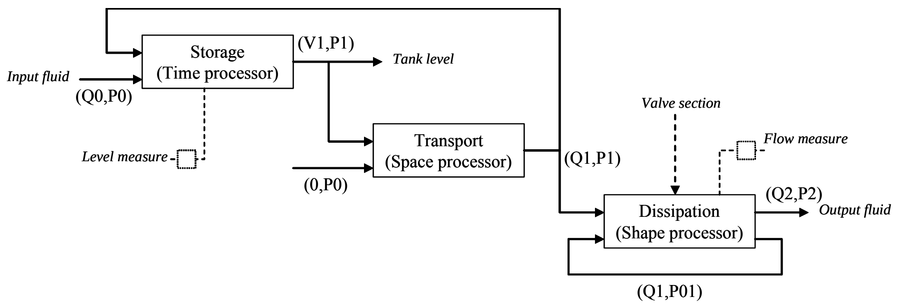
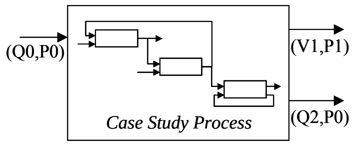


Figure 10. *Network of basic processors*

This network is formalized in the *Case_study_process* B machine (see Figure 11). Hydraulic generic constructs are customized into specific case study processors by renaming constructs' names, variables, and operations according to the instantiation mechanism defined in section 3.3. (*Hydraulic_time_processor* machine is customized into *Tank.Time* machine, *Hydraulic_space_processor* into *Valve.Space* machine, and *Hydraulic_shape_processor* into *Valve.Shape* machine). These machines are then invoked within the *Case_study_process* B machine by the "*extends*" B mechanism. Operations are called with parameters that depend on the edges of the network.



MACHINE Case_study_process
VARIABLES Q_0, P_0, V_1, P_1, Q_1, Q_2, P_{01}
EXTENDS Tank.Time, Valve.Space, Valve.Shape
INVARIANT $(V_1=0 \Rightarrow Q_2=0) \wedge (Q_2=Q_1) \wedge \ldots$
OPERATIONS
$((Q_2, P_0),(V_1, P_1)) \leftarrow$ Case study process (Q_0, P_0) =
$(V_1, P_1) \leftarrow$ Tank.Time $((Q_0, P_0),(Q_1, P_1))$
$\|$ $(Q_1, P_1) \leftarrow$ Valve.Space $((V_1, P_1),(0, P_0))$
$\|$ $((Q_2,P_2),(Q_1,P_{01})) \leftarrow$ Valve.Shape $((Q_1,P_1),(Q_1,P_{01}))$
END

Figure 11. *B formal network of the physical system*

Two levels of model verification can then be performed. The first is directly linked to the instantiation process. Indeed, using the B *extends* clause requires that the invoked operations in the *Case_study_process* machine satisfy the invariant as defined in the generic constructs when the pre-condition (the assumption required to safely execute an operation) associated with the called operation is true. This pre-condition mainly refers to the typology of physical variables and processors represented in the constructs. We illustrate this mechanism using the example of the *Case_study_process* machine shown in Figure 11: the *tank.time* invoked operation produces a couple of variables (V_1, P_1) that is used as input parameters by the *valve.space* invoked operation. The first level of verification consists of verifying that the pre-condition within the *hydraulic_space* construct is maintained true by applying the above input parameters.
The second level of verification concerns new properties that can be locally added to the *Case_study_process* machine. These properties may refer to:

– specific characteristics of the physical system expressed independently of the control or the users' requirements; the invariant of Figure 11 mentions (a) that no output fluid can be observed when the tank is empty, and (b) that the flows out of the tank before (Q1) and after (Q2) the valve are the equal (only the power is changed due to the energy dissipation);

– structural properties of the interconnected processes that represent legal sequences with respect to physical laws.

As an example of the last point, Feliot has demonstrated that only some sequences of processes are enabled with respect to causality principles producing a set of legal sequences, such as Time/Space/Shape/Space/Time. Knowing that the *Time*, *Space,* and *Shape* processors are characterized by their input and output couples, the legal sequence is coded into an invariant that checks that the successive input/output processor flows have the correct type as follows:
INVARIANT … /* see Figure 10 */ $\wedge$
$(Q_0,P_0) \in \text{FLOW} \rightarrow \text{EFFORT} \wedge (Q_1,P_1) \in \text{FLOW} \rightarrow \text{EFFORT} \wedge (V_1,P_1) \in \text{DISPL} \rightarrow$ $\text{EFFORT} \wedge$ ... *the same type checking for all the operations parameters*

To conclude this section, proving a specification of the process system based on instantiated formal constructs means that:

– from a structural point of view, the process system model is well structured with regards the physical rules; the network of interconnected processors is said to be compliant with the basic constructs and their assembly rules;

– from a dynamical point of view, the variables involved in the process system behave as expected by the modeller.

This process specification can then be further integrated, by the use of the B common language, into a more complex representation of the system, including the control model and the requirement model as recommended by the automation predicate.

4. Proof-oriented specification of the case study

Proof-oriented specification is then used to ensure the consistency between the goals specification on one hand and the process and control specifications on the other hand according to the predicate (2) and the B correspondences given in section 2.4. To increase the comprehensibility of the B models, an informal representation of the relationship between these specifications is given in the appendix using an UML notation. B specification of the physical process system is assumed to be given by the *Case_study_process* machine of the previous section.

4.1. *Goals specification*

To illustrate the systemic approach, the requirement specification is proposed to merge classical control problems (such as level control loop and mode management) with information control problem (such as historic management). The *System requirements* machine (see Figure 12) defines these services using three operations: *level_request*, *change_mode*, and *create_historic*.

```
MACHINE     System_requirements
SETS        PRESSURE = NAT; FLOW = NAT; VOLUME = NAT,
            MODE={Automatic, Manual}, MODECHG={M2A, A2M}, HISTORIC;
CONSTANTS   Lmax = 100, Input_flow_max = 100
VARIABLES   request, input_flow, output_flow, tank_level, mode,
            historic, chg_mode_historic, monitoring, level
INVARIANT   (tank_level = Lmax ⇒  Output_flow > 0)  ∧ …                 (I1)
            ∧ historic ⊆ HISTORIC ∧ monitoring ∈ historic → NAT    (I2)
OPERATIONS
Level_control (request) =
    PRE       request ∈ NAT
    ANY       new_output_flow, new_tank_level
    WHERE     new_output_flow ∈ [0, 100] ∧  new_tank_level = request
    THEN      tank_level := new_tank_level || output_flow := new_output_flow END
Change_mode  =
    ANY       newmode
    WHERE     newmode ∈ NAT
              ∧ (newmode = automatic ⇒ tank_level ∈ [request - 10, request + 10]))
    THEN      mode := newmode
Create_historic (h) =
    SELECT    newmode = change_mode(mode) ∧ newmode # newmode$0
    THEN      historic := historic ∪ {h}
              || IF newmode = Automatic and newmode$0 = Manual
              THEN monitoring(h) := A2M ELSIF monitoring(h) := M2A
              || level := request END
END
```

Figure 12. *B system requirement*

Production functionality is described using the "*any … where … then …*" primitive described in section 3.4. The post-conditioned operation describes the initial values of variables to be processed (*input_flow*, *request*) and the result to be obtained (*output_flow*, *tank_level*). This simply postulates that we have to modify the output water flow (*output_flow* becomes *new_output_flow*) if we want the new level (*new_tank_level*) to equal the requested level (*request*). In other words, this abstract substitution simply states the input/output expected modification without any other behavioural details on its realisation. Invariant properties of the production activity are given in the invariant (I1), saying that reaching the maximum level of the tank provokes output valve opening, i.e., an output flow other than zero.

Mode management is described using the same primitive, and details the conditions required for changing the production mode. Changing the mode of the system from automatic to manual can be done whatever the physical or control situations are, but changing from manual to automatic requires the system to be near a steady point where a control loop can be applied.

The last operation formalizes the need to monitor and to produce a history of mode commutations. It is done with the help of a:

– historic class having two attributes, the type of commutation (from Manual to Automatic or vice-versa noted M2A and A2M) and the corresponding request when the commutation is operated;

– an invariant (I2) stating the relationships between class and attributes;

– an operation describing when a new occurrence of historic class must be created and the value of its attributes (the $ operator in B is used to make the distinction between a variable value before and after execution of an operation).

Note that, at this level of abstraction, the system variables are defined from an end-user point of view. Only three physical variables are used: the input and output flows, and the level of the tank. We have seen that the physical reality is far from that when having to consider three different variables of flow *(Q_0, Q_1, Q_2)*, two pressure variables *(P_0, P_1)*, and the volume of water *(V1)* inside the tank.

4.2. *Control specification*

The B *Control* machine results, in this example, from an intuitive modelling of the controller, which manages operation modes and the associated control actions (see Figure 13). In automatic mode, the control is supposed to maintain the level of the tank near the value given by the user request. This control is done by increasing or decreasing the percentage of valve opening (*opening*). Note that this abstract specification of control just states what has to be done to maintain the requested level without detailing the algorithm to be used (for example PID). Mode management describes the exact procedure for commutation taking into account the general recommendations given in the *system_requirements* machine.

```
MACHINE      Control
SETS         MODE = {manual, automatic}
VARIABLES    control_mode, level
INVARIANT    control_mode ∈ MODE ∧ request ∈ NAT ∧ opening ∈ NAT
             ∧ level ∈ NAT ∧ …
OPERATIONS
opening ← Control (request, level) =
      SELECT control_mode = automatic ∧ request ∈ NAT ∧ level ∈ NAT
      THEN   IF request > level THEN opening . (opening<opening$0)
             ELSIF request < level THEN opening . (opening>opening$0)
             ELSIF request = level THEN opening . (opening=opening$0)
             END
      WHEN   state = manual
      THEN   IF request = 0  THEN opening := 0
             ELSIF request = 100 THEN opening := 100
             END
      END
mode_management =
      SELECT control_mode = automatic
      THEN   control_mode := manual
      WHEN   control_mode = manual ∧ level > request – 10 ∧ level < request + 10
      THEN   control_mode := automatic
      END
END
```

Figure 13. *B Control Machine*

Note that creation of the mode commutation history remains the same as in the *systems_requirements* machine in the absence of any detail of the database management system that should be used. The B variables, invariant, and operation related to this part are not presented again in the *control* machine.

4.3. *Consistency and correctness checking*

Next step consists in ensuring the refinement relationship between system requirements and a model of the automation system.

4.3.1. Formalization of the automation system

Abstract representation of the automation system is provided by combining the *control* and *Case_study_process* machines using the B structuring *extends* clause that allows invocation of operations (see Figure 14). Variables processed are defined as Q_i for the input flow, Q_o, P_o for the output flow, and V_t, P_t, for the volume and pressure in the tank.

The use of the *extends* clause means that all the invariants described in the process and control machines must be preserved by the *automation_system* machine. This description leads to a failure notification by the theorem proof tool.

MACHINE Automation_System
EXTENDS Case_study_process, Control
VARIABLES User_request, Chg_mode_request, Q_o, P_i, V_t, P_t, Q_i,
OPERATIONS
Level_control (request) =
  $((Q_o, P_i),(V_t, P_t)) \leftarrow$ Case study process (Q_i, P_i) $\|$ $Q_o \leftarrow$ Control (user_request, V_t);
Change_mode =
  SELECT Chg_mode_request $\in$ BOOL
  THEN mode_management */* operation call from Control machine */* ; END

Figure 14. *Formalization of the Automation System machine*

Analysis of the proof obligations, i.e., the underlying hypotheses that have to be satisfied for a proof success, gives some indication as to what should be corrected in our specification:

$Vt \in DISP \Rightarrow Vt \in NAT \wedge Q \in EFFORT \Rightarrow Q \in NAT$ *(extract of proof obligation)*

The problem highlighted is the following: the process model works with physical variables that belong to Paynter sets (effort, flow, displacement, and impulse), whereas the control machine deals with informational variables that belong to the set of natural or real numbers.
Beyond this type checking problem, it appears that the conformance between the variables involved in the process and control models has not been considered. Indeed, the control model deals with an intuitive concept of level, whereas the process model speaks of a volume. In the same way, the control model is supposed to calculate the required opening percentage of the valve, whereas the process model deals with the output flow rate and tube section.
An alternative to this conflict consists in defining new operations within the process model that will be in charge of transforming:

– an opening percentage calculated by the control system into a physical effect understandable by the process model; this can be done by modifying the status of *A* (tube section) within the *Valve.Shape* machine that is now defined as a variable, and by introducing a new operation *Set_parameters (A)* whose function is to initiate the calculation within the *Valve.Shape* machine with the section value that is required by the control;

– a physical volume *($V \in DISP$)* into an informational variable *($L \in NAT$)* that represents the level inside the tank; this can be done by introducing in the *Tank.Time* machine a new operation *$L \leftarrow$ Get_level (V)* whose function is to supply a type transformation, and to calculate the level of the tank from the knowledge of its volume and section (due to a proportional relationship between the two values).

These two operations added respectively to the *Valve.Shape* and *Tank.Time* process machines are in fact systemic operators of Nature (that aim to modify the type of processed variables, from the physical world to the informational domain and vice-versa). They actually represent the abstract behaviour of the Actuation and Measurement system.

4.3.2. Refinement checking

The last stage of our approach consists in verifying that the automation system we have defined satisfies the properties given by the requirements model. This consistency checking is performed using the B refinement mechanism. The properties to be proved are the following:

– the services provided by the automation system are included in those required by the end-user;

– the invariant properties of the automation system are included in those defined by the requirements model.

These assumptions comply with the definition of the B refinement. What has to be proved is that the *Automation_System* machine actually constitutes a refinement of the *System_requirements* machine. We postulate that this refinement relationship exists, and we describe, using the B formalism, the logical links between the abstract variables defined in the requirements *(input_flow, output_flow and tank_level)* and the more tangible requirements *(Q_o Q_i,V_t, P_i,P_t)* processed in the *Automation_system* machine. This can easily be done by enriching the invariant as follows (see Figure 15). The B theorem prover is used to demonstrate that the assumed refinement relationship is preserved by the machines involved in our specification.

REFINEMENT Automation_System
REFINES System_requirements
CONSTANTS atmospheric_pressure
INVARIANT
/ existing variables in both abstract and refined machines */*
Q_o = input_flow $\wedge$ Q_i = output_flow $\wedge$ V_t = tanl_level
/ not existing in the abstract machine */*
$\wedge$ P_i = atmospheric_pressure $\wedge$ P_t = k * tank_level
OPERATIONS
/ same operations as those defined in Figure 14 */*

Figure 15. *Automation_system as a refinement of System_requirements*

A proof success implies that the combined operations included in the B refined machine (i.e., operations of the *Automation_system* machine and consequently the called operations supplied by the *Control and Process* machines) satisfy the invariant and the post-conditions (i.e., produce the same results) of the operation related to *System_requirements* machine.

In other words, proving the refinement ensures that the actual operations performed by the control and process systems are compliant with the functional operations expected within the system requirements.

5. Conclusion

There is a growing interest in methods and tools that facilitate the validation of software-intensive automation systems. This interest becomes a legal requirement when dealing with safety-critical systems; the IEC 61508 safety-related standard strongly recommends the use of verification methods to be applied in the certification process by the suppliers, integrators, or independent external authorities, but without defining how they can be applied. Among many techniques enabling improvement in quality, formal approaches appear to be suitable for checking, from the early phase of a project life cycle, the completeness, the non-ambiguity, the consistency, and the correctness of all the various specifications a system is intended to meet.
Most formal methods aim to check *a posteriori* the correctness of a designed system with regard to required goals. In this case, the formal statement is very sensitive to a host of implicit relationships inferred by the requirement process, itself based on natural or skill-oriented languages. This paper presents an alternative system strategy aimed at making a more robust automation by checking *a priori* proved properties from the earlier phases of specification. Our method combines *model-driven* and *proof-oriented* specifications based on the B language and its refinement mechanism as a unique and integrating framework.

Although these interdisciplinary exchanges between computer science and automation-related approaches demonstrate that this *a priori* formal method of modelling allows one to verify the highest levels of safety integrity of automation systems, common experiments on laboratory-scale and industrial-scale case-studies emphasize the effort that still must be employed to make the proposed engineering framework effective in practice.
An important limitation for the acceptance of the proposed approach into a well-established automation engineering process is the mathematically oriented notation of the B language. A means of bridging this gap at the specification level is to propose equivalent representations of a B specification using more accepted formalisms. To this end, translations of UML diagrams into B machines have been proposed [LED 02][LAL 02], as well as paraphrasing of the B model into a natural language text to facilitate its understanding [DIA 98]. As far as the design and implementation levels are concerned, formal refinements of B models towards skill-oriented formalisms remain unresolved issues, although identified as crucial by industrial practitioners [AIT 02][MOI 03]. Some partial results are available for translating B into continuous [LEC 02], event-oriented [BER 03], reliability-oriented [MOR 04], or timed [ABR 98] models, as well as into programming languages such as C code [BER 00].

6. References

[ABR 91] ABRIAL J.R., BORGER E., LANGMAACK editors, Formal methods for industrial applications: specifying and programming the steam boiler control, *Lecture Notes in Computer Science*, State of the art-Survey, ISBN 3-540-61929-1.

[ABR 96] ABRIAL J.R., *The B Book: Assigning Programs to Meanings*. Cambridge Univ. Press, ISBN 0-521-49619-5.

[ABR 98] ABRIAL J.R., MUSSAT L., Introducing Dynamic Constraints in B, Proceedings of the Second International B Conference, Montpellier, France, April 22-24, 1998, *Lecture Notes in Computer Science* 1393, Springer, ISBN 3-540-64405-9.

[AIT 02] AIT-AMEUR Y., D'AUSBOURG B., BONIOL F., DELMAS R., WIELS V., A component based methodology for description of complex systems: an application to avionics systems, *European Systems Engineering Conference,* 21-24/2002, Toulouse, France.

[BER 00] BERT D., CAVE F., Construction of Finite Labelled Transition Systems from B Abstract Systems. In Integrated Formal Methods, IFM2000, *LNCS* 1945, pages 235-254. Springer-Verlag, November 2000.

[BER 03] BERT D., BOULMÉ S., POTET M.L., REQUET A., VOISIN L., Adaptable Translator of B Specifications to Embedded C Programs, In Proc. of *FME 2003: Formal Methods*, LNCS 2805, pp. 94-113, Pise, Sept. 2003.

[CAS 99] CASSANDRAS C.G., LAFORTUNE S., *Introduction to Discrete Event Systems*. Kluwer Academic Publisher, ISBN 0-7923-8609-4.

[CLA 00] CLARKE E.M., GRUNBERG O., PELED D.A., *Model Checking*, The MIT Press, 2000

[CRA 95] CRAIGEN D., GERHART S., RALSTON T., Formal methods technology transfer : implements and innovation, Formal Methods by Hinchey M.G., Bowen J.P., pp. 399-419, Prentice Hall International, 1995.

[DIA 98] DIALLO D., VAILLY A., Paraphrasing of specification written in B, *Networking and Information Systems Journal (NISJ)*, Vol. 1 n°2-3/1998.

[DUP 00] DUPUY S., BOUSQUET L., Validation of UML models thanks to Z and Lustre. In *Formal Methods Europe*, Berlin, Germany, March 2000.

[FEL 96] FÉLIOT C., CASSAR J. Ph., STAROSWIECKI M., Towards a Language of Physical Systems, *IEEE SMC Conference*, Beijing, Oct. 4-10, 1996.

[FEL 99] FELDMANN K., COLOMBO A.W., SCHNUR C., STÖCKEL T., Specification, Design, and Implementation of Logic Controllers based on Coloured Petri Net Models and the Standard IEC 1131, *IEEE Trans. on Control Systems Technology*, Vol. 7, n°6, 1999.

[FUS 83] FUSAOKA A., SEKI H., TAKAHASHI K, A description and reasoning of plant controllers in temporal logic. *Proceedings of the International Joint Conference on Artificial Intelligence,* pp 405-408. Karlsruhe, Germany, 8-12 Aug. 1983.

[IUN 03] IUNG B., MOREL G., LEGER J.B., Proactive maintenance strategy for harbour crane operation improvement. *Robotica*, vol. 21, pp. 313-324, ISSN 0263-5747, March 2003.

[JOH 04] JOHNSON T. L., Improving automation software dependability: a role for formal methods ? In proceedings of 11th IFAC *International Conference on Information Control Problems*, Salvador-Bahia, Brazil, April 4-7 2004.

[LAL 02] LALEAU R, POLLACK F., Coming and going from UML to B : a proposal to support traceability in rigorous IS development. In *Intl. Conf. Formal Specification and Development in Z and B (ZB2002)*, volume 2272 of *Lecture Notes in Computer Science*, pages 517–534, Grenoble, France, January 2002. Springer Verlag.

[LAM 01] LAMBOLEY P., Production Systems Automation Formal Method Proposal, PhD Thesis, University Henri Poincaré, Nancy 1, 2001 (in French).

[LAN 00] LANO K., BICARREGUI J., KAN P., Experiences of using formal methods for chemical process control specification, *Control Engineering Practice*, Vol. 8, n°1, 2000.

[LAN 96] LANO K., *The B language and method, a guide to practical formal development*. Springer Verlag, ISBN 3-540-76033-4, 1996.

[LEC 02] LECOMTE T., Event Driven B: language, tool support and experiments, in Proceedings of *RCS'02 International Workshop on Refinement of Critical Systems : Methods, Tools and Experience*, in conjunction with the 2nd Conference of B and Z Users (ZB 2002), January 23 - 25, 2002, Grenoble, France.

[LED 02] LEDANG H., Automatic translation from UML specifications to B, in proceedings *of the Int. Workshop on Refinement of Critical Systems: Methods, Tools and Experience,* in conjunction with the 2nd Conf. on B and Z, 23-25/01/2002, Grenoble, France.

[LHO 96] LHOSTE P., MOREL G., From discrete event behavioural modelling to intelligent actuation and measurement modelling, *In proceedings of the ASI annual Conference of the ICIMS-NOE*, Toulouse, 1996.

[LHO 99] LHOTE F., CHAZELET PH., DULMET M. (1999). The extension of principles of cybernetics towards engineering and manufacturing. *IFAC Annual Reviews in Control.* Volume 23 (1), pp 1-11.

[MAT 98] MATTSSON S.E., ELMQVIST H., OTTER M., Physical system modelling with Modelica, *Control Engineering Practice*, vol. 6, pp. 501-510, 1998.

[MEL 04] MELLOR S.J., KENDALL S., UHL A., WEISE D., *Model Driven Architecture*, Addison-Wesley Pub Co, March 3, 2004, ISBN: 0201788918.

[MOI 03] MOIK A., Engineering-related formal method for the development of safe industrial automation systems, *Automation Technology in Practice International journal*, Vol. 1, 2003, pp 45-53.

[MOR 04] MOREL G., MERY D., LEGER J.B., LECOMTE T., Proof-oriented fault-tolerant systems engineering: rationales, experiments and open issues, *7th IFAC Symposium on Cost Oriented Automation - COA'2004*, Gatineau, Québec, Canada, June 2004.

[PAN 05] PANETTO H., PÉTIN J.F., Metamodelling of production systems process models using UML stereotypes. *International Journal of Internet and Enterprise Management,* Inderscience publishers, 2005 - Vol. 3, No.2 pp. 155 - 169, ISSN 1476-1300.

[PAR 04] PARKER D., KWIATKOWSKA M., NORMAN G., Controller Dependability Analysis By Probabilistic Model Checking, IFAC/INCOM2004 symposium, Salvador-Bahia, Brazil, April 4-7 2004.

[PAY 61] PAYNTER M., *Analysis and design of engineering systems*, M.I.T. Press, Cambridge, ISBN 0-262-16004-8.

[PET 98a] PETIN J.F., IUNG B., MOREL G., Distributed Intelligent Actuation and Measurement system within an integrated shop-floor organisation, *Computers In Industry Journal,* Vol. 37, pp 197-211, Elsevier Sciences Pub, ISSN 0166-3615.

[PET 98b] PÉTIN J.F., MOREL G., MÉRY D., LAMBOLEY P.; Process control engineering: contribution to a formal structuring framework with the B method, *Lectures notes in Computer Science*, vol.1393, p. 198-209, 1998.

[POL 04] POLZER K., Ease of use in engineering – availability and safety during runtime, *Automation Technology in Practice International journal*, Vol. 1, 2004, pp49-60.

[RAM 87] RAMADGE P.J., WOHNHAM W.M., Supervisory control of a class of discrete event processes, *SIAM Journal on Control and Optimization*, Vol. 25(25), 1987, pp 206-230.

[ROU 02] ROUSSEL J.-M., FAURE J.-M., An algebraic approach for PLC programs verification, *6th IFAC / WODES*, Zaragoza, Spain, 2-4 October, 2002, pp. 303-308.

[SEL 98] SELIC B., Using UML for modelling complex real-time systems, *Lectures Notes in Computer Science*, 1474, pp. 250-262, 1998, ISSN: 0302-9743.

[SHE 01] SHELL T., Systems functions implementation and behavioural modelling: system theoretic approach, *International Journal of Systems Engineering*, 4/1, 2001.

[WRI 88] WRIGHT P.K., BOURNE D.A., *Manufacturing Intelligence*, Addison-Wesley, ISBN 0-201-13576-0, 1988.

APPENDIX

All UML graphical notations are very helpful in improving the readability of the B machines presented in this paper. The best practices found in the computer science literature start a UML study by defining the main objects classes involved in the application considered. This can be done using "Use case" diagrams, or more precisely, using class diagrams.
Applying Fusaoka's predicate to our case study leads to the following class diagram (see Figure 16), where the *Automation_System* class is composed of a *Process_System* class and a *Control_System* class. Associated operations represent the services provided by these two classes (physical transformations for the first and informational processing for the second). The dependency association between the *Automation_System* and *Requirements* classes represents the refinement relationship that must exist between these two classes. The Object Constraint Language (OCL) enables the expression of constraints applied on objects or associations of objects, and is used in the case study to express the invariant properties of the system.

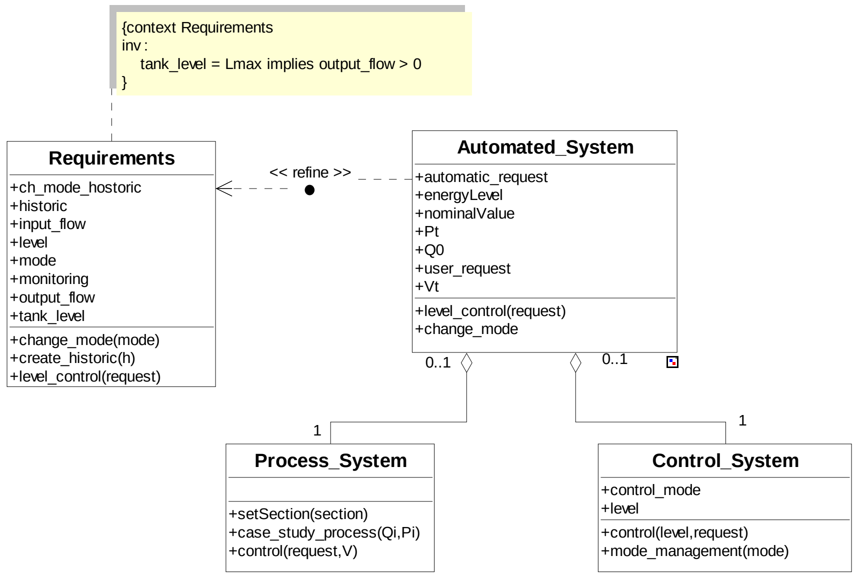


Figure 16. *UML class diagram of specifications according to Fusaoka's predicate*

A further specification should enrich this static representation with additional details related to the dynamic behaviour of the objects. Unfortunately, even if many formalisms are available (information exchanges between objects can then be described using a collaboration diagram, scheduling of operations within a class can be described by a sequence diagram, control operation behaviour can be described through a State-Transition diagram, etc.), the refinement rationale, which is the accurate way to guarantee the consistency the specification process, is barely supported by UML.