 © 2013 Operational Research Society Ltd. All rights reserved. 1747-7778/13

www.palgrave-journals.com/jos/

Use of formal languages to consolidate a Holonic MAS methodology: a specification approach for analysing Problem and Agency domains

B Mazigh[1], M Garoui[1]* and A Koukam[2]

[1]Department of Computer Science, Monastir, Tunisia; [2]University of Technology of Belfort Montbéliard, Belfort, France

In complex systems, multiple aspects interact and influence each other. A vast number of entities are present in the system. Traditional modeling and simulation techniques fail to capture interactions between loosely coupled aspects of a complex distributed system. The objective of this work is to formalize and to specify a part of the Agent-oriented Software Process for Engineering Complex Systems methodology (Problem and Agency Domains) for modeling the holarchy of studied system by using a formal specification approch based on two formalisms: Petri Net and Object-Z language. Such a specification style facilitates the modeling of complex systems with both structural and behavioural aspects. Our generic approach is illustrated by applying it to FIRA Robot Soccer and is validated with the Symbolic Analysis Laboratory framework.

Journal of Simulation advance online publication, 1 February 2013; doi:10.1057/jos.2012.24

Keywords: multi-agent systems; formal specification; Petri Nets; Object-Z; SAL

1. Introduction

In computer science, a formal specification is a mathematical description of software or hardware that may be used to develop an implementation. It describes what the system should do but not (necessarily) how the system should proceed. Given such a specification, it is possible to use formal verification techniques to demonstrate that a candidate system design is correct with respect to the specification. This has the advantage that incorrect candidate system designs can be revised before a major investment has been made in actually implementing the design. An alternative approach is to use provably correct refinement steps to transform a specification into a design, and ultimately into an actual implementation that is correct by construction.

A design (or implementation) can never be declared correct in isolation, but only 'correct with respect to a given specification'. Whether the formal specification correctly describes the problem to be solved is a separate issue. It is also a difficult issue to address, since it ultimately concerns the problem constructing abstracted formal representations of an informal concrete Problem Domain, and such an abstraction step is not amenable to formal proof. However, it is possible to validate a specification by proving challenging theorems concerning properties that the specification is expected to exhibit. If correct, these theorems reinforce the specifier's understanding of the specification and its relationship with the underlying Problem Domain. If not, the specification probably needs to be changed to better reflect the understanding domain of those involved in producing (and implementing) the specification.

In the specification domain, there are several methodologies to help the modelling and analysis phase of Multi-Agent Systems (MAS) and Holonic systems. Among these methodologies, we point out the well known: Tropos (Giorgini, 1995), PASSI (Azaiez, 1992) and Agent-oriented Software Process for Engineering Complex Systems (ASPECS) (Object Management Group, 2003; Gaud, 2007).

ASPECS uses UML as a semi-formal modeling language and consequently it makes this meta-model ambiguous. In D'inverno and Luck (2003), a principled theory of agency is developed by describing just such a framework, called the SMART agent framework. Using Z specification formal language, a sophisticated model of agent and their relationships is built up and illustrated with some applications. They demonstrated that Z language is well suited to model data structures and functionalities in a highly abstract fashion but do not treat the behavioral aspect. In this work, we use our specification language called PNOZ, based on two formalisms: Petri Nets (PN) and Object-Z (OZ). Such a specification style facilitates the modeling of systems with functional and behavioral aspects. The objective of this work consists of consolidating the ASPECS methodology by using our formal specification and analysis and validates such a specification with the framework SAL (Symbolic Analysis

**M Garoui, Department of Computer Science, Monastir 5000, Tunisia.*

Laboratory). This consolidation is done by formalizing the core concepts of the first two domains (Problem and Agency Domain) of ASPECS meta-model.

After a brief presentation of the simulator for the FIRA Robot Soccer competition, a quick overview of the ASPECS process and modelling approach will be presented in Section 3. Section 4 presents formal specifications of the first two domains (Problem and Agency Domains) of ASPECS process based on composition of PN and OZ language named PNOZ. Section 5 presents the validation process of PNOZ specification with the framework SAL. Finally, Section 6 summarises the results of the paper and describes some future work directions.

2. Case study: FIRA Robot Soccer

This case study intends to model a simulator for the FIRA Robot Soccer competition (FIRA Robot Soccer-ASPECSWiki).

This competition involves two teams of five autonomous robots playing a game similar to soccer (Figure 1). This is a classical example where real-time coordination is required. It constitutes a well-known benchmark for several research fields, such as MAS, image processing and control.

Robot soccer players are two wheel-driven small mobile robots, which are controlled by a host computer. A soccer team in the category of MiroSOT (Micro Robot World Cup Soccer Tournament) consists of five players, one goal keeper and four players for each team. There are several game categories. The size and form of the robot in each category is fixed by rules.

Robot soccer competitions give an opportunity to foster intelligent techniques and intelligent robotics research by providing a standard problem where a wide spectrum of technologies can be developed, tested and integrated, for example, collaborative multiple agent robotics, autonomous computing, real-time reasoning and sensor fusion. From a scientific point of view, the robot soccer players are 'intelligent, autonomous agents'. They have a global goal like 'win the game'. Robot soccer aims at promoting the developments in small autonomous robots and intelligent system (agent) that can cooperate with each other.

Figure 1 FIRA Robot Soccer.

3. A quick overview of the used ASPECS process

As proposed by Gaud (2007) and Cossentino *et al* (2007), ASPECS is a step-by-step requirement to code software engineering process based on a meta-model, which defines the main concepts for the proposed Holonic Multi-Agent Systems (HMAS) analysis, design and development based on *Capacities*, *Role*, *Interaction* and *Organization* (*CRIO*) framework. The target scope for the proposed approach can be found in complex systems, and especially in hierarchical complex ones. The main vocation of ASPECS is towards the development of societies of holonic (as well as non-holonic) MAS. ASPECS has been built by adopting the Model-driven architecture (Object Management Group, 2003). In Cossentino *et al* (2010), they label the three metamodels 'domains' thus maintaining the link with the PASSI metamodels. The three definite fields are:

- *The Problem Domain*: It provides the organizational description of the problem independently of a specific solution. The concepts introduced in this domain are used mainly during the analysis phase and at the beginning of the design phase.
- *The Agency Domain*: It introduces agent-related concepts and provides a description of the holonic, multiagent solution resulting from a refinement of the *Problem Domain* elements.
- *The Solution Domain:* It is related to the implementation of the solution on a specific platform *Janus* Project; (Gaud *et al*, 2008). This domain is thus dependent on a particular implementation and deployment platform.

A downfall in ASPECS is that it uses UML as a modelling language. Because of the specific needs of agents and holonic organizational design, the UML semantics and notation are used as reference points, which makes ASPECS meta-model ambiguous.

Our contribution will relate to the consolidation of ASPECS process by formalizing the *Problem Domain* and the *Agency Domain* associated with the ASPECS meta-model, therefore facilitating the *Solution Domain.*

4. Formal languages

4.1. PN

PN (Murata, 1989) are 3-tuple $N=(P, T, F)$, where P and T are finite, non-empty and disjoint sets. P is the set of places and T is the set of transitions. The flow relation between P and T is denoted by $F \subseteq (P \times T) \cap (T \times P)$. The preset of a node $x \in P \cup T$ is defined by $\bullet x = \{y \in P \cup T / (y, x) \in F\}$. The

postset of a node $x \in P \cup T$ is defined by $x^{\bullet} = \{y \in P \cup T / (x, y) \in F\}$. The preset (postset) of a set is defined by the union of the presets (postsets) of their elements. A marking of N is a mapping $M{:}P \rightarrow \mathbb{N}$. (N, M) is called a net system or a marked net. A transition t is said to be enabled if each of its input place p is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of arc from p to t. $M[t\rangle$ means that transition t is enabled under M. After t fires at M, a new marking M' results. This is denoted as $M[t\rangle M'$. The set of all markings reachable from a marking M_0, in symbols $R(N, M_0)$, is the smallest set in which $M_0 \in R(N, M_0)$ and $M_0 \in R(N, M_0)$ if both $M \in R(N, M_0)$ and $M_0[t\rangle M'$ hold. For a PN with n places and m transitions, its incidence matrix A is an $n \times m$ matrix of integers and its typical entry is given by $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$, where $a_{ij}^{+} = w(i,j)$ is the weight of arc to place p_i from its input transition t_j and $a_{ij}^{-} = w(i,j)$ is the weight of arc from place p_i to its output transition t_j. An example of PN is shown in Figures 2(a) and (b).

In Figure 2, one can verify that $P = \{p_1, p_2\}$, $T = \{t_1\}$ and $F = \{(p_1, t_1), (t_1, p_1)\}$. Moreover, $p_1^{\bullet} = \{t_1\}$, $t_1^{\bullet} = \{p_2\}$, $p_2^{\bullet} = \oslash$, ${}^{\bullet}p_1 = \oslash$, ${}^{\bullet}t_1 = p_1$, ${}^{\bullet}p_2 = t_1$. The initial marking is $M_0 = [1, 0]^T$ under which t_1 is enabled since $M(p_1, t_1) = 1 \geqslant w(p_1, t_1)$. After t_1 fires, one token is removed from its preceding place, that is, p_1, and deposited into its succeeding place, that is, p_2.

4.2. OZ

OZ (Smith, 1995) is an extension of the Z formal specification language to accommodate object orientation. The essential extension to Z in OZ is the class construct, which groups the definition of a state schema with the definitions of its associated operations. A class is a template for objects of that class: the states of each object are instances of the state schema of the class, and its individual state transitions conform to individual operations of the class. An object is said to be an instance of a class and to evolve according to the definitions of its class. For example, operation schemas have a Δ-list of those attributes whose values may change. By convention, no Δ-list means that no attribute changes value. The standard behavioral interpretation of OZ objects is as a transition system (Duke and Rose, 2000).

5. Our formal specification language: PNOZ

In this section, we present an integration method of PN and OZ by defining a coherent formalism called PNOZ (Garoui, 2011).

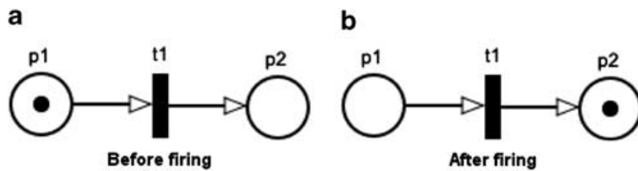

Figure 2 A Petri Nets example.

In others works, Xudong (2001) proposes a formal method, PZ Nets, integrating PN with Z. He have provided a unified formal definition of combining PN and Z, an approach for developing PZ net specifications and a technique for analyzing invariant properties of PZ net specifications. He believes that PZ nets are a powerful formal method for specifying and analysing many different system aspects (structure, control, data and functionality) and system kinds (sequential, concurrent and distributed). But this formal method has disadvantages: we notice that this method specified each aspects (structural and behavioral) separately and have not integrated in the same formalism.

Considerable work has been devoted to formal specifications that combine two or more languages. The main advantage of the multi-formalism approaches comes from a substantial gain in expressive facility. For instance, the approach presented in this paper assigns to OZ, the description of data structures and functions, and to the PN, the description of behavioural aspects. This section presents a simplified description of the operational syntax and semantics of PNOZ specification models. An *electronic key* system will be used along with this work to illustrate different aspects of the approach. The system was treated with greater details in Duke *et al* (2005) in which an effective solution to the problem is given.

5.1. PNOZ syntax

Syntactically, PNOZ specification units are like OZ classes, with the addition of a Behavior schema, which includes a PN. We add the given type [Event] to allow for the declaration of Petri Nets primitive events as variables of the class.

The *Door* class below specifies a component of the *electronic key* system. The class schema *Door* includes, from top to bottom, an abbreviation declaration, the behavior schema, containing a Petri Nets. Next comes an unnamed schema generally called the state schema, including the declaration of all class attributes. Next schema, named (INIT or Init), includes a list of predicates that characterize the initial state of the class. The two last schemas define specific operations of the class.

Formally, a PNOZ class C is defined by giving a triple (V_C, B_C, O_C). The set V_C includes the variables of the class, as named in the state schema. B_C is the behavior PN and O_C is the set of operations of the class, the names of the operation schemas of the class. For the Door class, variables and operations are:

$$V_{Door} = \{closed,\ opened\}$$
$$O_{Door} = \{open,\ close\}$$

Operations *Open* and *Close* determine the opening and closing of the door.

Formally, the PN B_C is defined by giving the structure $\psi{:}(V, T, \phi)$. Symbol V denotes a set of state variables. In the

rest of the paper, we assume $V=\{x_1 \ldots x_n\}$ unless otherwise stated. A state s is an interpretation of V. Symbol T denotes a set of transitions; each transition $\tau \in T$ is defined by a transition relation ρ_τ, a first-order formula in which the unprimed variables refer to the values in the current state and the primed variables refer to the values in the next state. Symbol ϕ denotes an assertion over V that represents the initial condition. The assertion ϕ is assumed to be satisfiable. Each one of these sets can be precisely determined by giving an inductive definition of its syntax.

As an example, the B_{Door} PN has:

$$V = \{closed, opened\}$$

$$T = \{open, close\}$$

5.2. PNOZ semantics

The semantic description of a PNOZ class C consists in representing the set of computations that C can yield. Computations are sequences of states subject to causal restrictions imposed by the elements and the structure of C. The state of class C, which we call a situation, is essentially a pair $s=(v,k)$. Symbol v: $V_C \rightarrow D$ denotes a valuation of all the variables of C, with D denoting the super domain where all the variables take values, each one according to its type. Symbol k represents a state configuration of the Behavior PN. A state configuration is a maximal subset of states that can be active. The initial situation $s_0=(v_0,k_0)$ is determined as follows. The initial valuation v_0 is a valuation that satisfies the predicates of the INIT schema. Variables that do not appear in the INIT schema usually are given default values. k_0 is the initial state configuration. The basic evolution stage is the situation change, called step, which we describe now. Step $i+1$ takes the system from situation i to situation $i+1$ and is noted

$$(v_i,k_i) \xrightarrow{E_i} (v_{i+1},k_{i+1}) \quad (1)$$

where E_i is the set of events activated at step i. The step occurs when at least one of the PN's transitions is active.

To describe the situation transformation produced by a step, we adopt the formalism of transition systems, particularly the Mana and Pnueli notation style by means of predicates (Manna and Pnueli, 1995). With class C, we associate the transition system $S_C=(V,\Theta,T)$. Symbol $V=V_C \cup \{k\}$ represents the set of variables. Variable k takes value in 2^{Σ} and represents PN's state configuration. The states of the transition system S_C are the situations of C, that is, valuations of the variables in V. If s denote a state of S_C, we simplify notation as follows: for all $v \in V, s[v]$ denotes the value of v at s. Symbol Θ represents the initial state predicate. Any valuation of V that satisfies is an initial state of the system: s is an initial state if $s[\Theta]=true$ (where $s[\Theta]$ denotes the valuation of formula Θ from the value of its variables in s). Symbol T represents the set of transitions. A transition $\tau_i \in T$ defines an elementary change of the state of the transition system. Such a change is described by a transition relation

$$\rho_i = V \times V' \rightarrow \{TRUE, FALSE\} \quad (2)$$

Generally, a transition $\tau_i \in T$ is characterized by its transition relation ρ_i (see expression (3)), that is, a predicate conjunction.

For the notations Z/OZ, we adopt the conventions before/after (Pre/Post) to define predicates. Among the general forms possible for the transition relation, we adopt the following form:

$$\rho_i = Pre(\tau_i) \wedge (v_1' = Exp_1) \wedge \ldots \wedge (v_n' = Exp_n) \quad (3)$$

where $Pre(i)$ is the pre-condition of the transition ρ_i and s is a source state for the transition ρ_i if and only if $s| = Pre(t)$.

For each partner of the expression (3), $(v_j'=Exp_j)$ $v_j \in V \times V'$ and $Exp_j, j \in [1,n]$ is an expression without primed variables.

To the set V of variable symbols, we add the set V' of variable symbols decorated with a prime character ('). For any $v \in V$, an occurrence of symbol v in ρ_i represents the valuation of v in the source state of transition τ_i and an occurrence of v', the valuation of v in the destination state of τ_i.

To illustrate the general principle for the derivation of T from the PNOZ model, let us consider the PN specifying the behaviour of the class *Door* (Figure 3). It shows some portion of the behaviour PN of a PNOZ class.

It is easily inferred that the PN of Figure 4 yields τ_1, τ_2 with

$$\rho_1 = (status = closed) \wedge (Open = TRUE) \wedge (status' = opened)$$

$$\rho_2 = (status = opened) \wedge (Close = TRUE) \wedge (status' = closed)$$

Transition relation $\rho_{1,}$ ρ_2 describes the case where both PN transitions are triggered in diverse step. Note that an event is represented by a Boolean variable that gets value *TRUE* during a step and goes back to *FALSE* in the following step, unless the environment or the system itself assert it again.

The treatment, within the transition system S_C, of events produced by the environment (external, input events) is not trivial, but seems more adequate to consider this question in the section devoted to the timed version of the PNOZ semantics.

As an example, consider some elements of the transitions system associated with the PNOZ class Door, introduced in section 3. The initial state predicate is

$$\Theta_{Door} \equiv (k = closed) \wedge (status = opened)$$

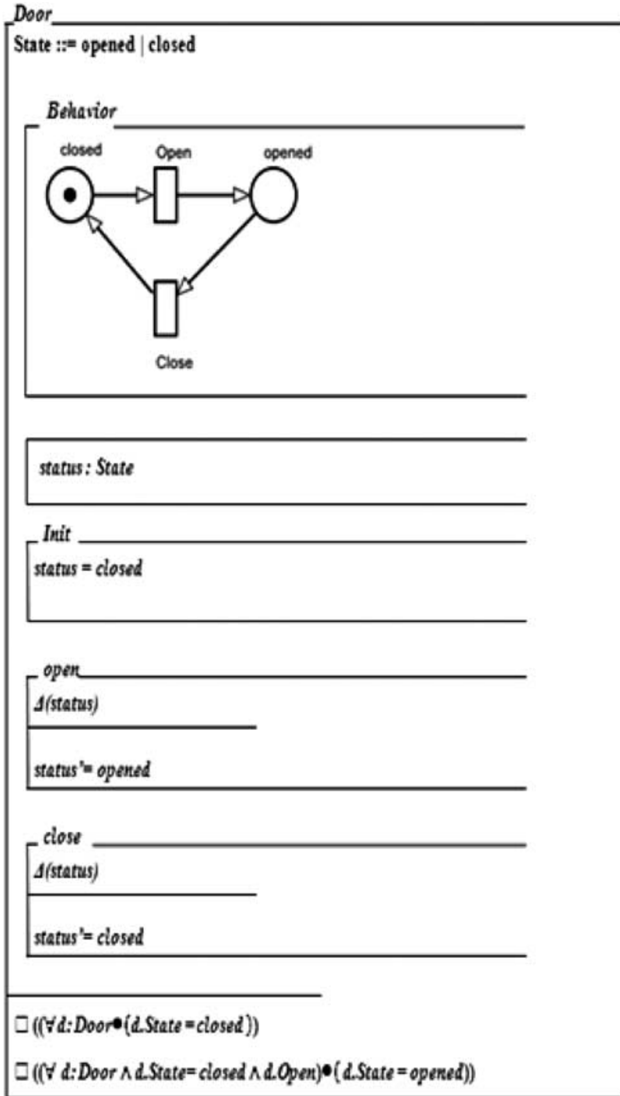

Figure 3 Door system specification based on the syntax PNOZ.

PN transitions outing state opened and state closed yields transition relations

$$\rho_1 \equiv (k = opened) \wedge (status = closed) \wedge (Open = TRUE)$$

$$\rho_2 \equiv (k = closed) \wedge (status = opened) \wedge (Close = TRUE)$$

6. Formal ASPECS metamodel

We use our specification formalism PNOZ introduced in Garoui (2011); Mazigh *et al* (2011); for efficiency modelling and analysis of FIRA Robot Soccer competition. This specification formalism combines two formal languages: PN

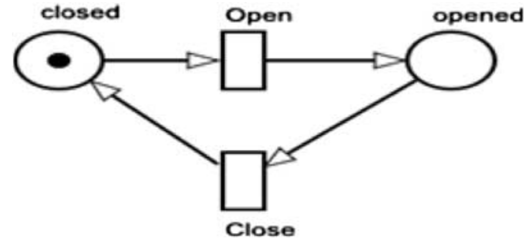

Figure 4 Petri Nets associated with Door's behaviour.

(Vinh Duc, 2005; Smith, 1995) and OZ (Vinh Duc, 2005; Smith, 1995).

Our approach consists in giving a formal modeling based on PNOZ language. To validate our formalization approach, we have limited our work to the specification of the *Team Simulation Organization*, which is a part of the holonic structure of the system studied.

6.1. Problem Domain associated with our case study

In this section, we use the ASPECS methodology to describe partially the Problem Domain in the form of concepts: *Role*, *Organization*, *Interaction* and *Capacity*. All these concepts are inherited from the classes of *CRIO* framework.

To do this work, we use our PNOZ language introduced in previous sections to formalize the main concepts of the Problem Domain associated with ASPECS metamodel. The organization *Team Simulation* (Figure 5) is composed of four roles (*PlayersSimulator*, *RoleAssigner*, *StrategySelector* and *GameObserver*), six interactions between these roles and three capacities (*PlayStrategy*, *ObserveGame* and *ChooseStrategy*) required by the roles.

Formally, the roles are specified by OZ classes containing behaviour schemas. These schemas include the PN which specifies the system's behaviour. The incoming and outgoing arrows indicate that the behaviour of such a role is related to the behaviour of another role. They explain that there is an information exchange between the roles.

The class *PlayersSimulatorRole* (Figure 6) specifies the role *PlayersSimulator*. It inherits from the role class and adds these attributes: *requiredCapacity*, which is a set of Capacities required by the role.

The role requires a capacity which is named *PlayStrategy*. The behaviour of the *PlayerSimulatorRole* specified by the behaviour schema consists of four states. We suppose that our system can be in these four different states (*PlayersSimulatorReady, PlayersReady, InitializationDone and Playing*). For this reason, we use a type *State* to describe the system states. The incoming and outgoing arrows show, as we said, the interconnection between different roles of the same organization. The role *PlayersSimulatorRole* exchange data with the others roles of *Team Simulation* organization such as *RoleAssignerRole*, *GameObserver* Role and other external actions.

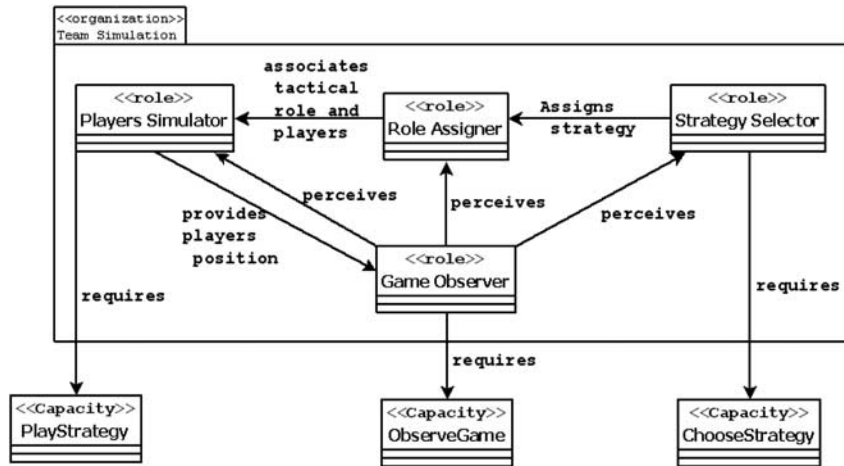

Figure 5 Team Simulation Organization.

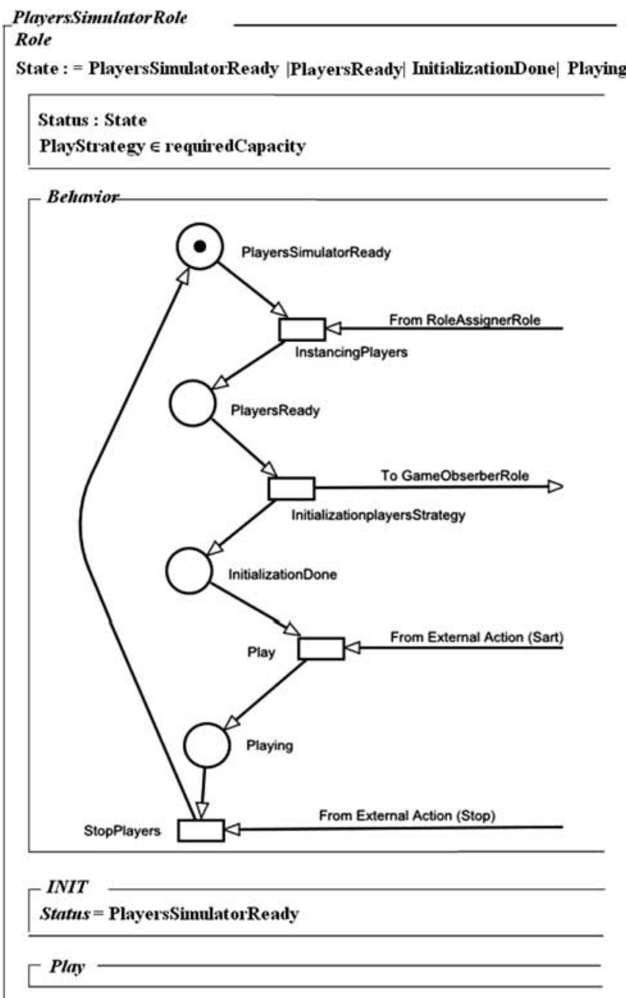

Figure 6 PlayersSimulatorRole class.

The class *PlayStrategy* (Figure 7) specifies the *PlayStrategy Capacity*. Its inputs are a set of *Strategies* and it produces a *PlayingStrategy* as output.

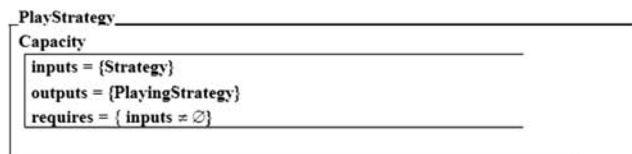

Figure 7 PlayStrategyCapacity class.

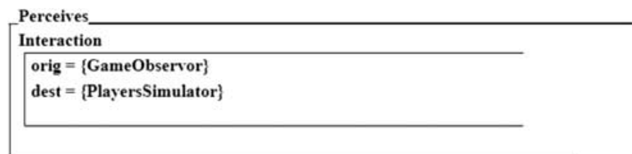

Figure 8 PerceivesInteraction class.

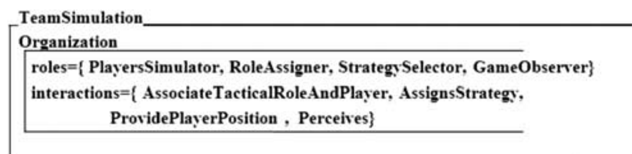

Figure 9 Team SimulationOrganization class.

The class *Perceives* (Figure 8) specifies the *Perceives Interaction*. It adds the following attributes: *orig* and *dest* to specify the origin and destination of an interaction between two roles.

The other roles of Team Simulation organization (Figure 9) will be specified in the same way as PlayerSimulatorRole. The class StrategySelectorRole, RoleAssignerRole and GameObserverRole specify, respectively, the role StrategySelector, RoleAssigner and GameObserver.

Remark: The class *TeamSimulation* (Figure 9) specifies the *TeamSimulation* Organization. It has the following attributes: *roles* representing a set of *Role*, *interactions* representing a set of *Interaction* in the same organization.

6.2. Agency Domain associated with our case study

In this section, the ASPECS methodology is used to describe partially the analysis phase, the design of the agent society and propose a holonic structure of the FIRA Robot Soccer system.

The Agency Domain includes the elements that are used to define an agent-oriented solution for the problem depicted in the previous stage. Among these elements, we will focus on the most important elements such as *AgentRole*, *Agent*, *Holon*, *HolonicGroup* and *ProductionGroup*.

AgentRole is an instance of the *Problem Domain Role*. It is a behaviour and it owns a set of rules in a specific group context.

AutonomousEntity is an abstract rational entity that adopts a decision in order to obtain the satisfaction of one or more of its own goals. An autonomous entity may play a set of Agent Roles within various groups. These roles interact with each other in the specific context provided by the entity itself. The entity context is given by the knowledge, the capacities owned by the entity itself. Roles share this context by the simple fact of being part of the same entity.

An *Agent* is an *autonomousEntity* that has specific individual goals and the intrinsic ability to fulfil some capacities. *Holon* is an *autonomousEntity* that has collective goals (shared by all members) and may be composed of other holons, called members or sub-holons. A composed holon is called super-holon. The concept *Group* is an instance in the *Agency Domain* of an *Organization* defined in the *Problem Domain*. It is used to model an aggregation of *AgentRoles* played by *Holons*.

The structure of the holonic solution dealing with our Robot Soccer simulator is presented in Figure 10, Groups (g1, g5, g3 and g7) are holonic ones (HG). At Level 2 of the holarchy, two super-holons H1 and H2 play the role of the *Team* Role in g0 group. Thus, g0 is an instance of a *Game Simulation* Organization. Each of these two super-holons contains an instance of *Team Simulation* Organization (group g2 and g6). Inspired by a monarchic government type, holon members playing the roles of *StrategySelector* (H5 and H9) are automatically named Head and Representative of the other members. Holon Part H3 playing the role of *PlayersSimulator* is decomposed and contains an instance of the *PlayerSimulator* Organization. Its government is inspired by the Apanarchy where all the members are implied in the process of decision making (all holons are Heads). The atomic holon H6 plays the role of Multipart as it is shared by two couples of super holons (H1, H2) and (H3, H7). This holon represents the environmental part of the application.

The class *PlayersSimulatorAgentRole* (Figure 11) specifies the *PlayersSimulator* AgentRole. It has the following attributes: its *requiredCapacity* is a set of Capacities and it provide services illustrated by a set of services (*providedService*) and *agenttask* is a set of actions specific to agent.

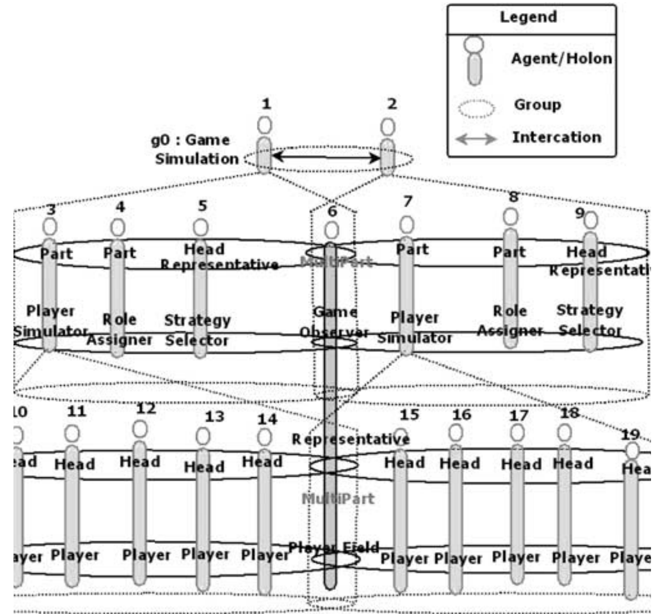

Figure 10 Holonic Structure of the FIRA Robot Soccer.

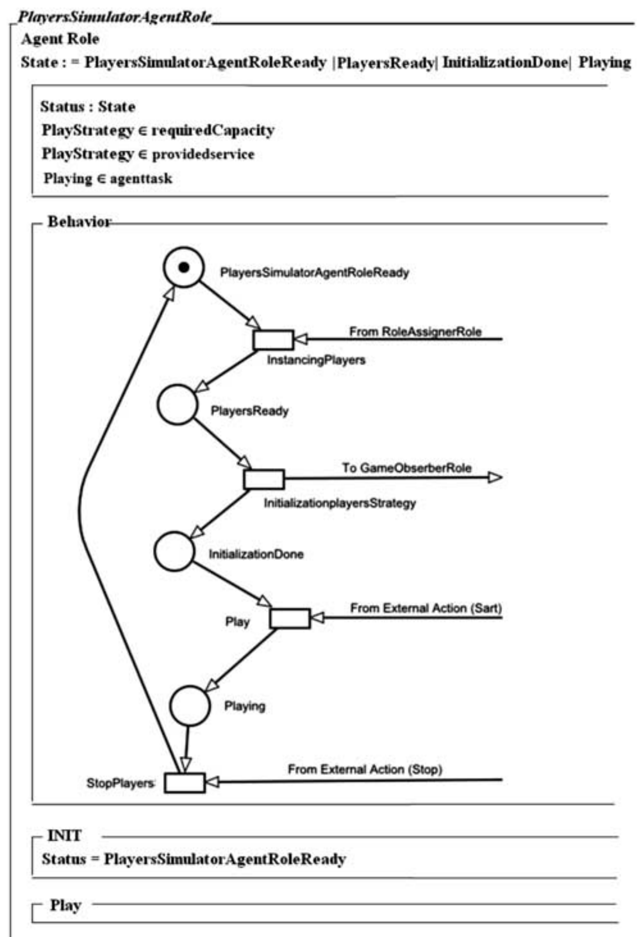

Figure 11 PlayersSimulatorAgentRole class.

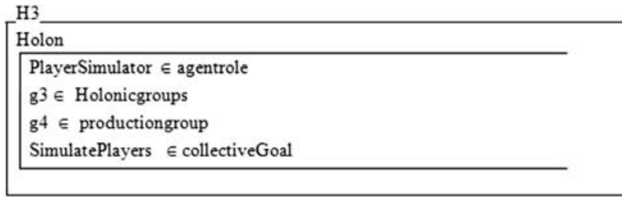

Figure 12 Holon H3 class.

The behaviour of the *PlayersSimulatorAgentRole* is specified by the behaviour schema. This schema contains PN, which are used to describe the behaviour of the class *PlayerSimulatorAgentRole*. This is last played by the holon H3.

Class H3 (Figure 12) specifies the *PlayersSimulatorHolon* of the holonic structure. It has the following attributes: *agentrole* (a set of played *AgentRole*), *Holonicgroups* (a set of *HolonicGroup*), *productiongroup* (a set of *ProductionGroup*) and collective Goal, which contains all goals of holon.

Class *g*3 (Figure 13) specifies the *HolonicGroup g*3. It adds the following attributes: its *members* are a set of *HolonicMembers*.

Class *g*4 (Figure 14) specifies the *ProductionGroup g*4. It has the following attributes: its *agents* are a set of *Agent*.

Among these agents, class *H11* (Figure 15) specifies *Agent H11*. It introduces the following attributes: its *agentroles* is a set of *AgentRole* played by *Agent H11* and *individualgoal* is a set of *Goals* used by agents to achieve some tasks.

7. Validation of PNOZ specification with SAL

SAL (Natarajan, 2000a, b) stands for Symbolic Analysis Laboratory. It is a framework for combining different tools for abstraction, programme analysis, theorem proving and model checking towards the calculation of properties (symbolic analysis) of transition systems. A key part of the SAL framework is a language for describing transition systems. This language serves as a specification language and as the target for translators that extract the transition system description for popular programming languages such as Esterel, Java and Statecharts. The language also serves as a common source for driving different analysis tools through translators from the SAL language to the input format for the tools, and from the output of these tools back to the SAL language.

The SAL distribution includes the following primary tools: the SAL well-formedness checker (*sal-wfc*), the SAL symbolic model checker (*sal-smc*), sal-deadlock-checker and so on. The other SAL tools should not be applied until the errors identified by the well-formedness checker have been corrected. Note that *sal-wfc* is not a full typechecker, so some type-incorrect SAL specifications will escape detection and produce unpredictable results. A BDD (Binary Decision Diagrams)-based model checker for finite state systems is

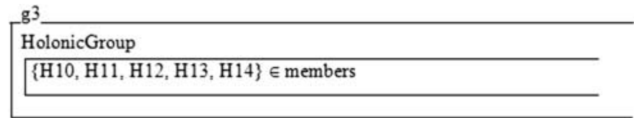

Figure 13 Holonicgroup g3 class.

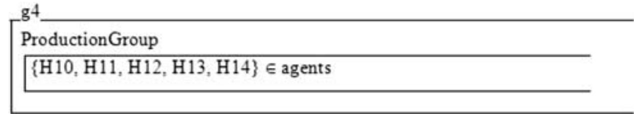

Figure 14 ProductionGroup g4 class.

used. SAL uses the CUDD (CU Decision Diagram) BDD package and provides access to many options for variable ordering, and for clustering and partitioning the transition relation. The model checker can perform both forward and backward search, and also prioritized traversal. *Sal-deadlock-checker*, an auxiliary tool is based on the symbolic model checker for detecting deadlocks in finite state systems. The SAL bounded model checker (*sal-bmc*) is a model checker for finite state systems based on SAT solving. In addition to refutation (ie, bug detection and counter-example generation), the SAL-bounded model checker can perform verification by *k*-induction. *Sal-path-finder* is an auxiliary tool based on the bounded model checker that generates random paths. The SAL simulator (*sal-sim*) is an interactive front end to other SAL tools.

In this section, we validate our specification with SAL tools. Each specification in PNOZ is transformed into SAL CONTEXT (the framework for declaring types, constants, modules and module properties).

7.1. From PNOZ to SAL

This approach of translation allows the flexibility required to directly translate PNOZ specification into SAL. The translation process is done by following steps: (1) Each OZ class is represented by SAL MODULE (is a self-contained specification of a transition system in SAL) in a specific SAL CONTEXT. (2) The variables of the state schema become local variables of the module, and inputs and outputs of the operations become input and output variables of the module.

Guarded commands may be used in the initialization and transition sections of a SAL module. (3) The initialization section of a SAL module may comprise a single guarded command. (4) The transition section may comprise a choice between several guarded commands separated by the syntax '[]'. These guarded commands may be labelled to aid the understanding of counter-examples generated by the SAL model checkers.

The part of PN describing the behaviour of class PNOZ converted into type represented by an enumerated state of

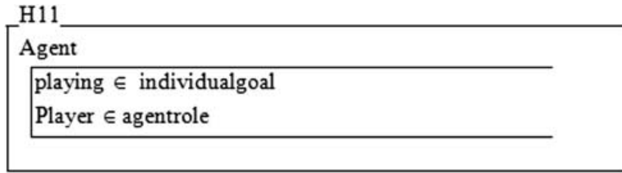

Figure 15 Agent H11 class.

the system and the transitions of PN are converted into INPUT variables of type Boolean in state schema of OZ class.

Figure 16 represents SAL CONTEXT associated with the class *PlayersSimulatorRole* role. This CONTEXT contains a definition of type *State*, which is a set that enumerates all possible states of the system. In addition, this CONTEXT also contains a SAL MODULE named *PlayersSimulator-Role*, which includes the declaration of Inputs (INPUT), Outputs (OUTPUT) and Locals variables.

OZ class operations are converted into a set of transitions in the section TRANSITION of the SAL MODULE. The translation of OZ operation schemas into SAL consists of three stages. First, all input and output variables are extracted and converted into their similar forms in SAL. In addition, each operation schema is converted into a single transition, knowing that the OZ schema predicate becomes the guard for the guarded command, expressing the relationship between the primed and unprimed versions of variables. The primed *status* is added to the guard to indicate that the state must hold after firing each transition.

SALenv contains a symbolic model checker called *sal-smc*, which allows users to specify properties in Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). However, in the current version, SALenv does not print counter-examples for CTL properties. When users specify an invalid property in LTL, a counter-example is produced. LTL formulas state properties about each linear path induced by a module. Typical LTL operators are:

- $G(p)$ (read as 'always p'), stating that p is always true.
- $F(p)$ (read as 'eventually p'), stating that p will be eventually true.
- $U(p, q)$ (read as 'p until q'), stating that p holds until a state is reached where q holds.
- $X(p)$ (read as 'next p'), stating that p is true in the next state.

For instance, the formula $G(p \Rightarrow F(q))$ states that whenever p holds, q will eventually hold. The formula G(F(p)) states that p often holds infinitely.

```
PlayersSimulator : CONTEXT =
BEGIN

  State : Type = {PlayersSimulatorReady, PlayersReady,
InitialisationDone, Playing};

      PlayersSimulatorRole : MODULE =

      BEGIN

          INPUT  InstancingPlayers : boolean
          INPUT  InitializationPlayersStrategy : boolean
          INPUT  Play : boolean
          INPUT  StopPlayers : boolean
          OUTPUT  status : State

      INITIALIZATION

          status = PlayersSimulatorReady

        TRANSITION [

            Inst_Players : status = PlayersSimulatorReady  and
 InstancingPlayers --> status' = PlayersReady
            []
            Init_Players : status = PlayersReady  and
 InitializationPlayersStrategy --> status' = InitialisationDone
            []
            Play : status = InitialisationDone  and Play --> status' =
 Playing
            []
            Stop : status = Playing  and  StopPlayers --> status' =
 PlayersSimulatorReady
          ]
        end;

   %------------
   % Properties
   %------------

     livness : theorem system |- G( F(status =PlayersSimulatorReady));
     th1 : theorem system |- G( Play  => F( status = Playing));
     th2 : theorem system |- G(StopPlayers  => F( status =
 PlayersSimulatorReady));

 end
```

Figure 16 SAL CONTEXT associated with the class Players-Simulator.

7.2. Properties and proofs

The example illustrated by Figure 16 shows some properties of the system written in the form of theorems with the LTL and CTL formulas. The SAL language includes the clause *theorem* for declaring that a properties is valid with respect to a modelled system by a CONTEXT. These properties can be verified using the following commands:

The first theorem *th1* can be interpreted as 'whenever Play transition holds, the system will be in Playing State'. The following command line is used:

./sal-smc PlayersSimulator th1
proved.

The second theorem *th2* can be interpreted as 'whenever StopGame transition holds, the system will be in Ready State'. The following command line is used:

./sal-smc PlayersSimulator th2
proved.

SALenv also contains a Bounded Model Checker called *sal-bmc*. This model checker only supports LTL formulas and is basically used for refutation, although it can produce proofs by induction of safty properties. The following command line is used:

./sal-smc PlayersSimulator th2
no counterexample between depths [0, 10].

Remark: The default behaviour is to look for counter-example up to a depth of 10??. The option $-depth = <num>$ can be used to control the depth of the search. The option *iterative* forces the model checker to use iterative deepening and is useful to find the shortest counter-example for a given property.

Before proving a livness property, we must check if the transition relation is total, that is, if every state has at least one successor. The model checker may produce unsound results when the transition relation is not total. The totality property can be verified using the *sal-deadlock-checker*.The following command line is used:

./sal-deadlock-checker PlayersSimulator PlayersSimulator-Role
Ok (module does NOT contain deadlock state.

The livness theorem can be interpreted as 'the initial state of the system is always Ready state'. Now, we use *sal-smc* to check the property *livness* with the following command line:

./sal-smc -v 3 PlayersSimulatorRole livness
proved.

7.3. Compositions and proofs

SAL provides two composition operators for building complex systems from other modules. The asynchronous composition operator is denoted by the syntax '[]'. SAL also provides a synchronous composition operator denoted by the syntax '||'. The two types of compositions can be freely mixed. For example, one may construct a system as the synchronous composition of two modules that are themselves built by asynchronous composition of other sub-modules.

The composition operators have the usual semantics. In an asynchronous composition, only one module makes a transition at a time. In a synchronous composition, all modules must make simultaneous transitions.

In our work, we will specify *TeamSimulation* Organization with the framework SAL. TeamSimulation Organization is represented with a module SAL named *TeamSimulation*. This module is the main module of specification, which is a synchronous composition of all the modules associated with the roles of this organization: the module *PlayersSimulatorRole* refers to the *PlayersSimulator* Role, the module *StrategySelectorRole* refers to the *StrategySelector* Role, the module *RoleAssignerRole* refers to the *RoleAssigner* Role and the module *GameObserverRole* refers to the *GameObserver* Role.

%——————————

%Full system : synchronous composition

%——————————

TeamSimulation : *MODULE* = PlayersSimulatorRole || StrategySelectorRole || RoleAssignerRole ||GameObserver-Role;

8. Conclusion

In this article, we showed that HMAS is well adapted to analyse and design hierarchical complex systems. The meta-model utilized can be exploited in the implantation stage with the advantage of having formally validated its structure and behaviour by using our composition formalism approach based on PN and OZ named PNOZ. Our future works will focus on a quantative analysis and behavioural validation of different models of ASPECS metamodel. At the same time, it will be interesting to extend PN with FNLOG (A Logic-Based Function Specification Language) (Mosbahi *et al*, 2002).

References

Azaiez S (1992). *Approche dirigée par les modeles pour le développement de systèmes multi-agent*. PhD Thesis, Université de Savoie, France.

Cossentino M *et al* (2007). A holonic metamodel for agent-oriented analysis and design. *3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems in LNAI*, 4659, Springer-Verlag: Berlin, Heidelberg.

Cossentino M *et al* (2010). ASPECS: An agent-oriented software process for engineering complex systems: How to design agent societies under a holonic perspective. *Autonomous Agents Multi-Agent Systems* **20**: 260–304.

D'inverno M and Luck M (2003). *Understanding Agent Systems* Springer series on Agent Technology, 2nd edn, ISBN 3-540-40700-6, Springer Verlag: New York.

Duke R, Dong JS and Hao P (2005). Integrating Object-Z with timed automata. In: *ICECCS '05 Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, ISBN: 0-7695-2284-X, doi10.1109/ICECCS.2005.56, pp. 488–497.

Duke R and Rose G (2000). *Formal Object Oriented Specification Using Object-Z. Cornerstones of Computing*. Macmillan Press: Basingstoke, UK.

FIRA Robot Soccer-ASPECSWiki http://www.aspecs.org/FIRA_Robot_Soccer.

Garoui M (2011). *Vers une approche de spécification formelle des systèmes Multi-agent Holoniques*. Masters Thesis, Faculty of Science of Monastir, Tunisia.

Gaud N (2007). *Holonic multi-agent systems: From the analysis to the implementation: Metamodel, Methodology and Multilevel Simulation*. PhD Thesis, Université de Technologie de Belfort-Montbéliard, France.

Gaud N, Hilaire V, Galland S and Koukam A (2008). An Organizational Platform for Holonic and Multi-agent Systems, Multi-agent Systems Group, System and Transport Laboratory, University of Technology of Belfort Montbéliard. *Published in*

the Sixth International Workshop on Programming Multi-Agent Systems (ProMAS 08) of the Seventh International Conference on Autonomous agents and Multi-agent Systems (AAMAS), Springer: Berlin, Heidelberg.

Giorgini P (1995). *The Tropos Metamodel and Its Use*. University of Trento, via Sommarive 14, I-38050 Trento-Povo: Italy.

Janus Project developed by the multiagent teams of the Laboratoire Systémes, Transports and the Centro de Investigación de Tecnologías Avanzadas de Tucumán, http://www.janus-project.org.

Manna Z and Pnueli A (1995). *Temporal Verification of Reactive Systems—Safety*. Springer Verlag: New York, ISBN 0-387-94459-1.

Mazigh B, Hilaire V and Koukam A (2011). Formal specification of Holonic Multi-agent systems: Application to distributed maintenance company. *Published in the Proceedings of PAAMS 2011*, Vol. 7327. Springer Verlag: Salamanca, pp. 370–378.

Mosbahi O, Jemni L, Ben Ahmed S and Jaray J (2002). A specification and validation technique based on STATEMATE and FNLOG, LNCS 2495. In: *ICFEM '02 Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*. Springer-Verlag: London, UK, pp. 216–220.

Murata T (1989). Petri nets: Properties, analysis and applications. *Proceedings of IEEE* **77**(4): 480–541.

Natarajan S (2000a). *Symbolic Analysis of Transition Systems*, In: Yuri G, Phillip WK, Martin O and Lothar T (eds). *Lecture Notes in Computer Science*, Springer-Verlag: Switzerland, pp. 287–308.

Natarajan S (2000b). Combining theorem proving and model checking through symbolic analysis. Computer Science Laboratory SRI International, Invited Paper at CONCUR. *Lecture Notes in Computer Science*, Springer-Verlag: PA, pp. 1–16.

Object Management Group (2003). MDA guide. v1.0.1, OMG/2003-06-01.

Smith G (1995). A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing* **7**(3): 289–313.

Vinh Duc T (2005). *Reseau de Petri* Rapport final de TIPE, Institut de la Francophonie pour l'Informatique.

Xudong H (2001). PZ nets—a formal method integrating Petri nets with Z. *Information and Software Technology* **43**(1): 1–18.

Received 30 November 2012;
accepted 5 December 2012 after two revisions