

Event-B based Verification of Interaction Properties In Multi-Agent Systems

Leila Jemni Ben Ayed

Faculty of Sciences of TunisResearch Laboratory UTIC,
ESSTT
Tunis, Tunisia
leila.jemni@fsegt.rnu.tn

Fatma Siala

Faculty of Science of Tunis
Tunis, Tunisia
Fatma.Siala@gnet.tn

Abstract— This paper presents a new event-B based approach to reasoning about interaction protocols. We show how an event-B model can be structured from AUML protocol diagrams and then used to give a formal semantic to protocol diagrams which supports proofs of their correctness. More precisely, we give rules for the translation of protocol diagrams into event-B language. In particular, we focus on the translation of messages with response delay, complex messages and protocol's states. The event-B language allows the definition of invariant describing required interaction properties and provides an automatic proof. By an example of multi-agent systems interaction protocol, we illustrate the proposed translation.

Keywords- Multi-Agent Systems; specification; verification; AUML; Event B

I. INTRODUCTION (HEADING 1)

Multi Agent Systems (MAS) [8] is an area of distributed artificial intelligence that emphasizes the joint behaviors of agents with some degree of autonomy and the complexity arising from their interactions. Most of which require a high level of safety and reliability. It is therefore necessary to follow a strict process of modeling, and formal verification. This allows one to rigorously verify required properties before the implementation. The last few years have seen the emergence of successful specification approaches to reasoning about Multi Agent Systems. Agent UML [13] extends different diagrams of UML [15] to model MAS in four views: Agents, Environment, Interaction and Organization. In particular, Statechart diagrams represent agents behavior in the environment, protocol diagrams that extend sequence diagrams, represent agent interactions and class diagrams model the organization and represent the different agent roles and the relations between them. AUML as a semi formal notation provides many advantages to agent systems design, such as simplified training and unified communication between development teams. However, the fact that AUML lacks a precise semantics is a serious drawback because it does not allow proofs and in consequence, with AUML, we can not verify required properties of interaction protocols in MAS like safety (non deadlock), liveness (precedence relation) and fairness properties. In the other hand, formal methods are the mathematical foundation for software. They increase the quality of applications development and perform the reliability of the applications.

II. RELATED WORK

Several solutions have been proposed for the specification of MAS using formal methods. Bakam [4] proposed to model protocol interactions in MAS with colored petri networks. This formalism is limited by the space explosion which requires some simplification of the model. Regayeg & al. [14] defined a new language based on the Z notation and the linear temporal logic LTL allowing specifications of the internal part of agents and protocol. The problem in using this solution is related to combinatorial explosion in state number in the verification.

In this context, we provide a specification and verification technique for multi-agent systems interaction protocols using AUML protocol diagrams which give readable models and the Event B method to allow verification of required properties which concern agents relations, precedence relations between agents, response delay and protocols states. We give a map from protocol diagrams and their properties to Event B language which allows the definition of invariant properties and provides proof by using a B tool: B4free [5] to ensure that in all the behavior, properties are verified. Hence, AUML models could be verified by analyzing derived Event B specifications. Other works proposed the use of semi-formal and formal method for the design of interaction protocols in MAS. Fadil & al. [7] proposed a solution combining AUML with B AMN [1] (Abstract Machine Notation). Our work is near to the one of [7]. However, we propose translation rules for the concepts of AUML into the notation of the Event B which is more adapted than the B AMN to the specification of MAS as reactive systems. Also, there is a semantic equivalence between messages and interactions in AUML protocol diagram and events in Event B which does not exists with operations in B AMN because operations may be called by the environment. In this paper, we present the proposed Technique. In addition to the translation of messages, agents, relations between messages into Event B, we also propose a solution to model time in Event B. This solution is used to translate messages with delay into Event B. By an example of a Contract-Net protocol [9], we illustrate the proposed technique.

III. AUML OVERVIEW

Agent Unified Modeling Language (AUML) [13] is a graphical modeling language that is being standardized by the Foundation for Intelligent Physical Agents (FIPA) [9]

Modeling Technical Committee. AUML was proposed as an extension of the Unified Modeling Language (UML) in order to tackle the differences between agents and objects. So far, there is no recognized standard for modeling a MAS and AUML emerged as a candidate to assume such a position.

AUML uses decomposition, abstraction and organization characteristics, which are the attitudes for reducing the complexity of development software. AUML decomposes a system into small parts of objects, models, use-case or class, various operational actions. Concerning the abstraction, it provides a specialized abstract view of modeling (class, use-case, diagram, interface etc.). It is used to create a set of semantics and conditions for operation and infrastructure services. The agent-oriented organization defines a range of elements and notations as a requirements specification for domain modeling. It aims to provide a model and an internal architecture of an agent system. It usually offers some frameworks (class, diagram, interface, etc.) to show how agents can be constructed in an agent system. The modeling focuses on one aspect at a time and increases the ability to understand complex problem issues during the time of system design. The core parts of AUML are mechanisms to model protocols for multi-agent interaction. This is achieved by introducing a new class of diagrams into UML: protocol diagrams. Protocol diagrams extend UML state and sequence diagrams in various ways. Particular extensions in this context include agent roles, multithreaded lifelines, extended message semantics, parameterized nested protocols, and protocol templates.

IV. EVENT B OVERVIEW

Event-B (Fig. 1) is a variant of the B method introduced by Abrial to deal with reactive systems [1], [2]. An event consists of a guard and an action. The guard is a predicate built on state variables and the action is a generalized substitution which defines a state transition. An event may be activated once its guard evaluates to true and a single event may be evaluated at once. The system is assumed to be closed and it means that every possible change over state variables is defined by transitions; transitions correspond to events defined in the model. The B method is based on the concept of machines (or systems) [1].

```

MACHINE <nom>
SETS <sets>
VARIABLES <variables>
INVARIANT <invariants>
INITIALISATION <initialization of variables>
EVENTS
<events>
END
    
```

Figure 1. The event B model

A machine consists on:

- Descriptive specification which describes what the system does using a set of variables, constants, properties over constants and invariants which specify required properties to be verified in each state. This constitutes the static definition of the model.

- Operational specification which describes the way how the system operates, it is composed of an initial state and various transitions represented by events which are spontaneous and which show how the set of variables of the descriptive specification can evolve in time. An event has a guard and an action. It may occur only when its guard evaluates to true. An event has one of the general forms (Fig. 2) [3] where the select form is just a particular case of the any form.

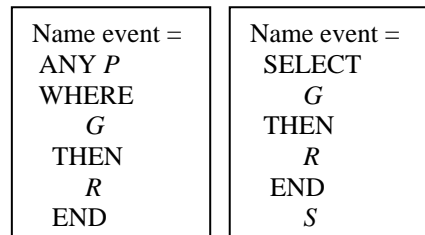


Figure 2. General Forms of events in event B

The consistency of an event B model is established by proof obligations which guarantee that the initialization should verify the invariant and that each event should preserve the invariant. The guard and the action of an event define a before-after predicate for this event. It describes relation between variables before the event holds and after this. Proof obligations are produced from events in order to state that the invariant condition is preserved. Let *M* be an event B model with *v* being variables, carrier sets or constants. The properties of constants are denoted by *P(v)*, which are predicates over constants, and the invariant by *I(v)*. Let *E* be an event of *M* with guard *G(v)* and before-after predicate *R(v, v')*. The initialization event is a generalized substitution of the form *v: init(v)*. Initial proof obligation guarantees that the initialization of the machine must satisfy its invariant: $\text{Init}(v) \Rightarrow I(v)$.

The second proof obligation is related to events. Each event *E*, if it holds, it has to preserve invariant. The feasibility statement is illustrated in Lemma 3.1 and the invariant preservation is given in Lemma 3.2. [12]

Lemma 3.1. $I(v) \wedge G(v) \wedge P(v) \Rightarrow \exists v' R(v, v')$

Lemma 3.2. $I(v) \wedge G(v) \wedge P(v) \wedge R(v, v') \Rightarrow I(v')$

An event B model *M* with invariants *I* is well-formed, denoted by $M \models I$ only if *M* satisfies all proof obligations.

V. THE PROPOSED TECHNIQUE

As shown in Fig. 3, the proposed technique consists mainly on three steps. In the first step, the MAS interaction protocol is modeled graphically with AUML protocol diagram and required properties are described by the user (liveness and safety invariants). In the second step, the resulting graphical readable model is translated into Event B in incremental development.

This resulting model is enriched by invariants describing relevant properties that will be proved in the third step using the B4free tool [5]. The third step verifies if the resulting B

model respects the system's invariants. The verification of these properties ensures the correctness and the validation of the described MAS.

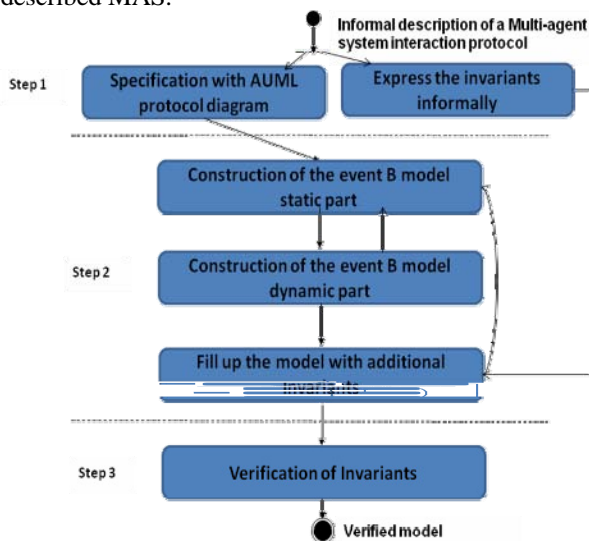


Figure 3. The proposed technique

Translating the structure of an AUML protocol diagram embedding the structure of a protocol diagram into event B consists of describing Messages, Agents and Order between messages. These are encoded into event B with a set of Messages, a set of Agents and a set of States. Each message is described as an event. We also consider the type of an event (simple, complex, with response delay). Therefore, some new variables, invariants and events may be added.

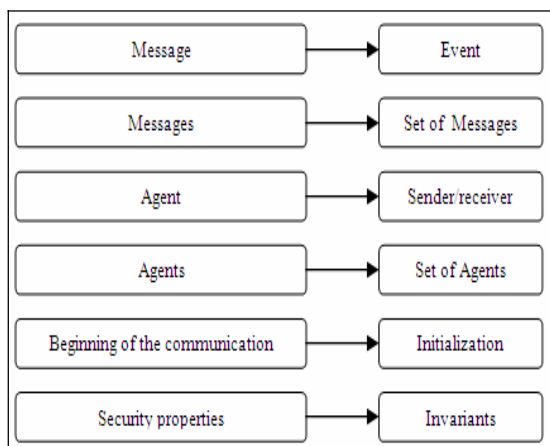


Figure 4. Mapping Interaction protocols primitives into event B

Fig. 4 shows mapping relations from AUML protocol diagram to event B notation.

A. Step 1: Specification with AUML protocol diagram. The communication in AUML, takes place between different agents. Each role represents all the agents having the same properties, the same interface, the same services or the same conduct. Fig. 5 shows the protocol diagram's skeleton.

B. Step 2: Translation into event B.

Our main contribution concerns this step which is divided into three sub-steps (Fig. 3): The construction of the B machine static part, the construction of the B machine dynamic part and the generation of the invariants describing required properties (liveness and safety invariants). These invariants will be verified at the third step.

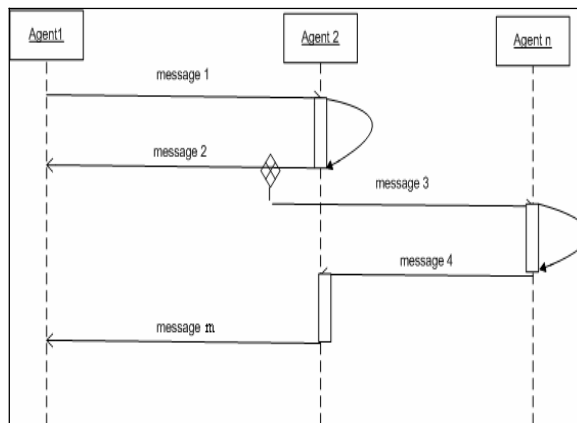


Figure 5. The protocol diagram's skeleton

Construction of the B machine static part: The basic units of the static part are agents, messages and roles. These are encoded into event B with a set of Messages, a set of Agents and a set of Roles. Each message is describe as an event. We also consider the TYPE of a message and the order of messages. Therefore, some new variables, invariants and events may be added.

B events are instantaneous and their effect can cause the occurrence of other events. This copes well with the semantics of AUML protocol diagrams; the sending of a message m_i is instantaneous and thus can lead to the sending of other transitions which has the output places of m_i from their input places. From an agent, a message leads to a new agent and soon. This is embedded in event B by an abstract system whose events correspond to messages.

Rule 1. From agents, their messages and roles we generate three sets AGENTS which corresponds to agent's roles, MESSAGES which corresponds to all messages and STATES_E which ccorresponds to different states of the system; We add three types of variables: *exchanged_msg*, *hand_name_event* and *ett*. *exchanged_msg* describes the exchanged messages between agents. It takes the (sender, message, receiver) form, where the sender and the receiver correspond to the names of the agent's roles. For each system state, identified by the event name, we generate a new variable *hand_name_event*. This variable takes the value 1 when the event holds and 0 in the other case. The variable *ett* represents system states.

The variable *exchanged_msg* is initialized to an empty set, each variable *hand_name_event* to 0 and the variable *ett* to *event_begin* in the *INITIALISATION* clause.

```

MODEL protocol_name
SETS
  AGENTS = {name_agent1, name_agent2, ..., name_agentn} ;
  MESSAGES = {msg1, ..., msgn} ;
  STATES_E = {evt_msg1, evt_msg2, ..., evt_msgn, evt_begin, evt_end}
VARIABLES
    
```

```

exchanged_msg, hand_msg1, hand_msg2, ..., hand_msgn, ett
INVARIANT
  Exchanged_msg ∈ AGENTS ↔ (MESSAGES ↔ AGENTS) ∧
  hand_msg1 ∈ N ∧ ... ∧ hand_msgn ∈ N ∧ ett ∈ STATES_E
INITIALISATION
  exchanged_msg := ∅ || hand_msg1 := 0 || hand_msg2 := 0 || ... ||
  hand_msgn := 0 || ett := evt_begin
    
```

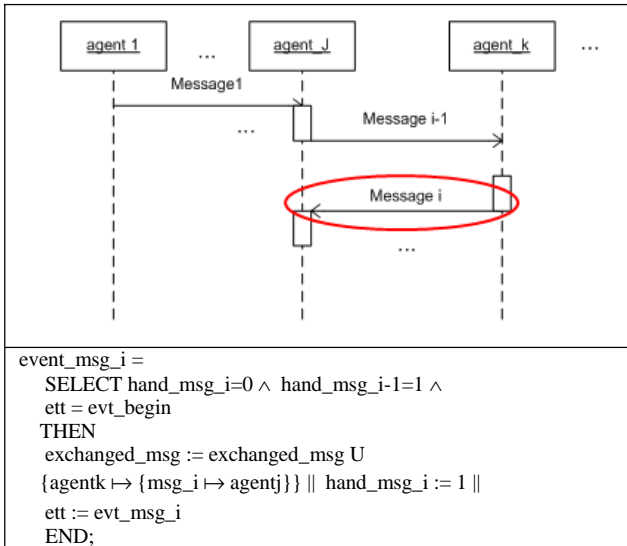
Construction of the dynamic part: B events are instantaneous and their effect can cause the occurrence of other events. This copes well with the semantic of AUML protocol diagrams. The sending of a message m_i is instantaneous and thus can lead to the sending of other messages which has the output agent of m_i from their input agent.

The dynamic part of the resulting Event B model is derived from simple or complex messages, protocol states and response delay. The result appears in the clause EVENTS.

Rule 2. Event occurrence: For each new event $event_name_event$ we generate a new variable $hand_name_event$. This variable takes the value 1 if the event holds and 0 in the other case.

Rule 3. Simple message: Each message msg_i is added as an event. The guard of this event is composed of a condition on state system defined with the variable ett , $hand_msg_i = 0$. The action of this event adds the new message to the variable $exchanged_msg$, Updates the variable ett with the new state and Adds $hand_msg_i = 1$.

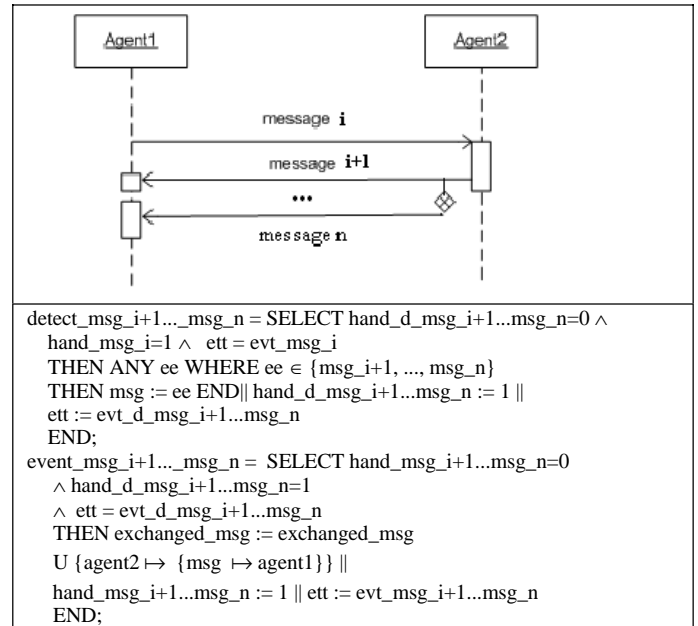
Rule 4. An event just necessarily following another event: For each message msg_i following another msg_i-1 , we add the condition $hand_msg_i-1=1$ to the guard of the event msg_i .



Modeling Complex messages: AUML contains three complex messages types: XOR, AND and OR.

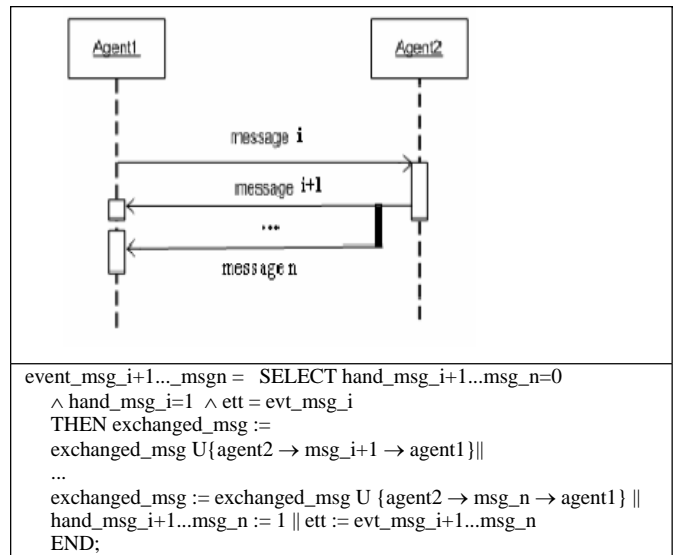
Rule 5. The XOR message: An XOR message consists on sending one and only one message at a same time. For each message XOR, which generates m_1, m_2, \dots, m_n messages, we add two events. The first event $Detect_m1_.._mn$ detects one of the messages randomly by using a variable ee that takes the value of one of these messages and assignment it to a variable

msg ; The second event $event_m1_.._mn$ adds the value of the variable msg to the variable $exchanged_msg$.



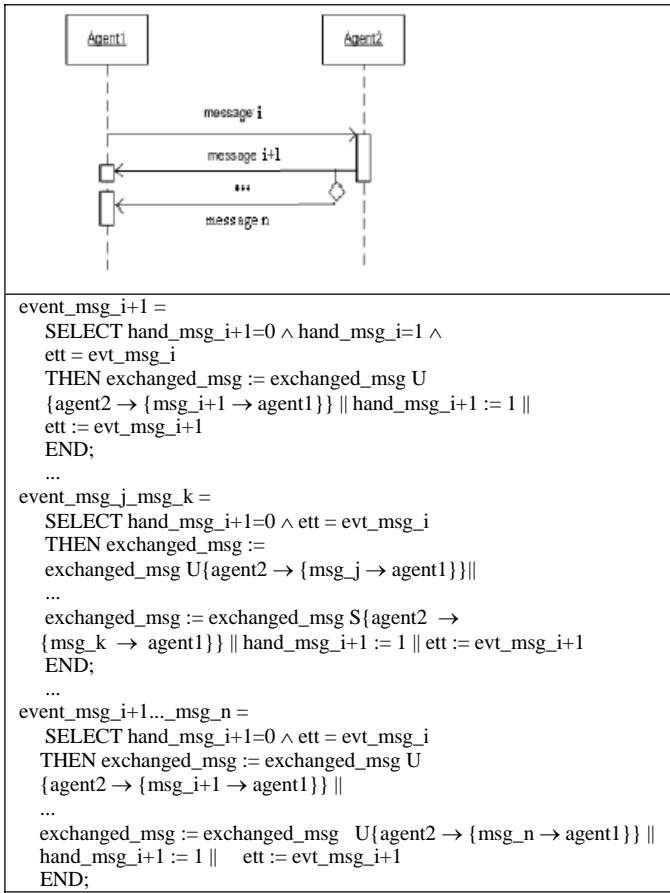
Rule 6. The AND message: An AND message consists on sending several messages at the same time.

If an AND connects a series of messages m_1, \dots, m_n , then we add to our B model a new event $event_m1_.._mn$ which adds to the variable $exchanged_msg$ all the messages m_1, \dots, m_n .



Rule 7. The OR message:

An OR message consists on sending one or many Messages at the same time. If an OR message is composed of a set of messages m_1, \dots, m_n then we add to our B model as many events as possible combinations between messages m_1, \dots, m_n . Each event adds to the variable $exchanged_msg$ messages of the combination. This message has the same guards and gives the same values to the variables $hand_msg_i$ and ett .

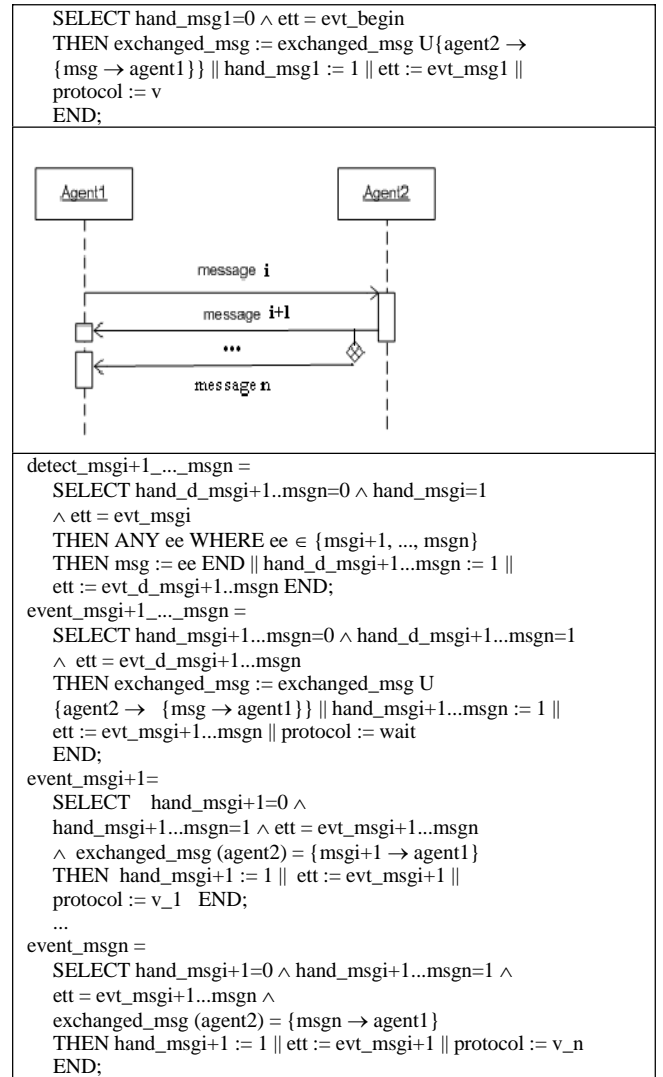


In the following, we focus on the translation of protocol states which are changing depending on the received and sent messages.

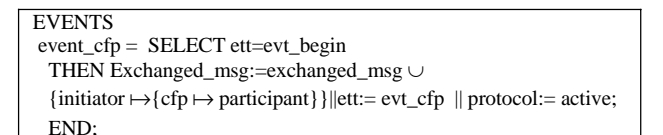
Rule 8. For each message, we assume that the protocol passes through four states: end, active, error and wait. We add a new set ($STATES_P = \{end, active, error, wait\}$) and a variable protocol modeling protocol states. The action of the event associated to this message will be updated by adding ($protocol:=v$) where $v \in STATES_P$.

The variable protocol takes the value:

- *active* when it is a negotiating message that expects a certain response ;
- *end* when it is a message that ends the communication ;
- *error* when it is an error message ;
- *wait* when it is an XOR type message where each basic message makes the protocol in a different state.



Modelling time in Event B: The B machines are sequential systems and it is not possible to dispose of the time along with another activity. To model time in Event B, we need to add a clock which is defined by an event tick and whose role is to advance the time represented by a variable time. We propose to model the parallelism between time and system with interleaving. We introduce a new variable *system* and the control is given alternatively to the clock when $system = 2$ and to the system in the other cases. *system* is also used to model relation between messages. Initially, $system = 1$. The event *detec_evt* sets this variable to 0 and the variable *v-e* to a Boolean value. If ($v-e=FALSE$) then the variable *system* passes to 2 (event AtoA) to increment the time (event tick) but if ($v-e=TRUE$) then the variable *system* passes to 4 (event AtoB) to execute the following message. When the time expired, the variable *system* passes to 3 (event AtoC) to take the hand to the event OFF which be detailed in the following.



```

Tick = SELECT System = 2 ^ state=stateA ^ ett=evt_cfp
      THEN time:=time+1 || System :=1 END;
detec_evt = SELECT System = 1 THEN ANY tf
      WHERE tf ∈ BOOL THEN v_e:=tf END || System :=0
      END;
AtoA= SELECT System = 0 ^ ett=evt_cfp ^
      time<time_out ^ v_e=FALSE ^ state=stateA
      THEN System :=2
      END;
AtoB = SELECT state=stateA ^ System = 0 ^
      ett=evt_cfp ^ time<time_out ^ v_e=TRUE ^
      hand_cfp=1
      THEN hand_cfp:=2 || state:=stateB || System :=4
      END;
AtoC = SELECT state=stateA ^ System=0 ^ ett=evt_cfp
      ^ time=time_out ^ v_e=FALSE
      THEN System:=3 || state :=stateC END;
OFF= SELECT protocol=active ^ (time ≥ time_out) ^
      system=3 THEN protocol:=end || ett:=evt_fin || time:=0
      END;
ON= SELECT protocol=end ^ time< time_out
      THEN protocol:=active || ett:=evt_begin
      END;
    
```

Rule 9. For each message with a response delay *time_out*, we add three states StateA, StateB and StateC. Three events are added AtoA, AtoB and AtoC. When the system is in stateA and if the message response holds before *time_out*, then the system passes to the state B (event AtoB). If nothing happen before *time_out* then the system switches to the state C (event AtoC). We add a precondition ($time < time_out$) to verify that this deadline is still valid. A boolean variable *v_e* takes a random value of a boolean *tf* to model response occurrence.

Rule 10. The Deadlock: When no event can be triggered, the system is blocked. That is the deadlock problem. Especially the situation holds when the two conditions: protocol is active and ($time \geq time_out$) are verified. To solve this problem, we add a new event OFF, which puts the protocol to end under these conditions and the event ON to ensure the resumption of protocol.

We conclude that the order of rules application respects the diagram shown in Fig. 6.

Filling up the system with properties: In this step, we enrich the model with invariants describing required properties.

The P1 invariant expresses that if ($system = 2$), then necessarily the system is in the same state (stateA):

$$(system=2) \Rightarrow (state=stateA)$$

The P2 invariant expresses that if the system is in the state stateB, then necessary the delay does not yet failed: ($state = stateB \Rightarrow (time < time_out)$)

The P3 invariant expresses that if the system is in the state stateC, then necessarily the deadline has arrived: ($state = stateC \Rightarrow (time = time_out)$)

The P4 invariant expresses that whenever the system is in CFP state then the protocol is active: ($ett=evt_cfp \Rightarrow (protocol=active)$).

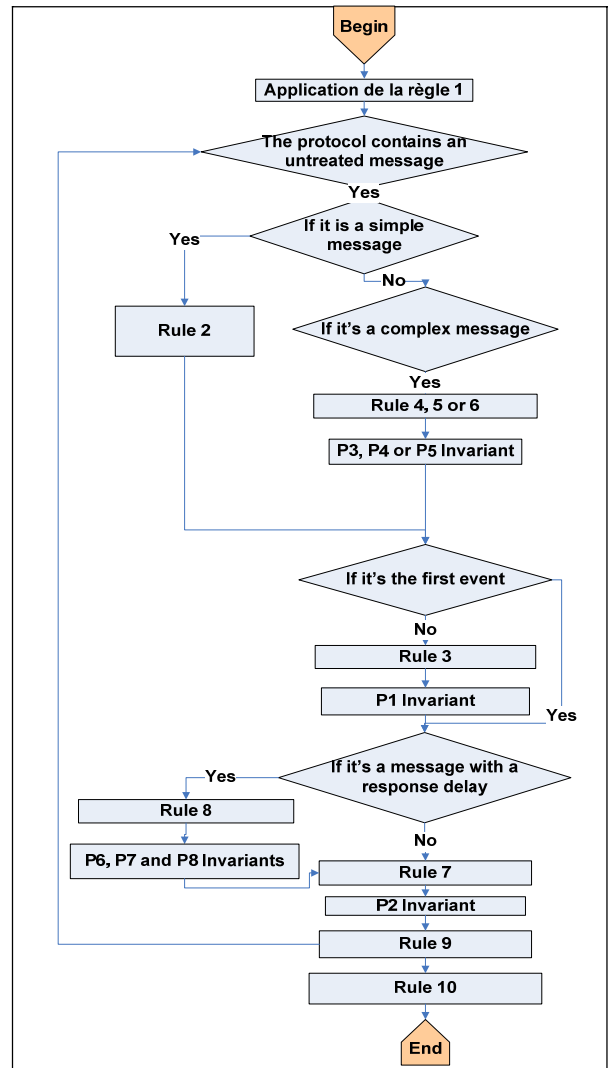


Figure 6. The translation process

The P5 invariant concerns the invariant process associated to the complex message type XOR as shown below.

$$(\forall (msg1, msg2). (((ett=evt_p) \vee (ett=evt_r) \vee (ett=evt_n)) \wedge (hand=4) \wedge (hand_prn=1) \wedge msg1 \in \text{MESSAGES} \wedge msg2 \in \text{MESSAGES} \wedge (agent2 \mapsto \{msg1 \mapsto agent1\}) \in \text{exchanged_msg} \wedge (agent2 \mapsto \{msg2 \mapsto agent1\}) \in \text{exchanged_msg} \wedge msg1 \in \{\text{propose, refuse, n_understood}\}) \Rightarrow (msg2 \notin \{\text{propose, refuse, n_understood}\})))$$

I. ILLUSTRATION

To illustrate our approach, we present in the following an example of a protocol diagram (Fig.7).

The properties added by the user are:

- Property 1 (safety property): The user can add a property concerning the messages with a response delay. The associated invariant is as follows:

$$\{Participant \mapsto \{\text{propose} \mapsto \text{Initiator}\}\} \Rightarrow ((\text{initiator} \mapsto \{\text{cfp} \mapsto \text{participant}\}) \wedge (time < time_out))$$

- Property 2 (safety property): The user can also add a property concerning the XOR messages:

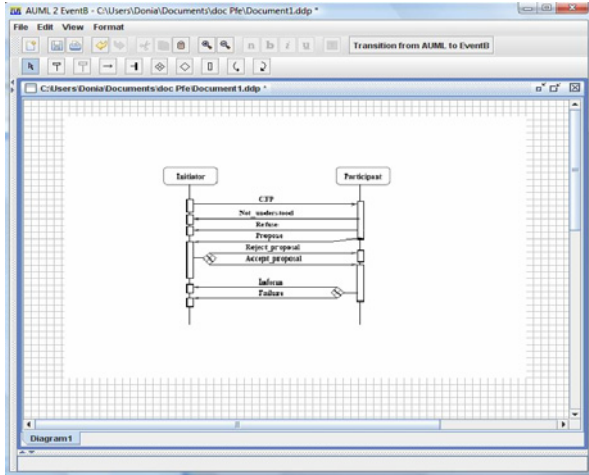
$$(\forall (msg1, msg2). (((msg1 \in \text{MESSAGES} \wedge msg2 \in \text{MESSAGES} \wedge (agent2 \mapsto \{msg1 \mapsto agent1\}) \in \text{exchanged_msg} \wedge (agent2 \mapsto \{msg2 \mapsto agent1\}) \in \text{exchanged_msg} \wedge msg1 \in \{\text{propose, refuse, n_understood}\}) \Rightarrow (msg2 \notin \{\text{propose, refuse, n_understood}\})))$$


Figure 7. Example of a protocol diagram

By using the rule 1, we obtain the specification following the formal part of the static CNET machine.

```

MODEL MCnetprotocol
SETS
  AGENTS = {initiator, participant};
  MESSAGES = {cfp, propose, refuse, n_understood, reject_proposal,
  accept_proposal, inform, failure};
  STATES_E = {evt_cfp, evt_propose, evt_refuse, evt_n_understood,
  evt_reject_accept, evt_ra, evt_inform_failure, evt_if, evt_begin,
  evt_end}
VARIABLES
  Exchanged_msg, hand_cfp, hand_propose, hand_refuse,
  hand_n_understood, hand_d_reject_accept, hand_reject_accept,
  hand_ra, hand_d_inform_failure, hand_inform_failure, hand_if, ett,
  protocol
INVARIANT
  exchanged_msg \in AGENTS \leftrightarrow (MESSAGES \leftrightarrow AGENTS) \wedge
  hand_cfp \in N \wedge hand_propose \in N \wedge hand_refuse \in N \wedge
  hand_n_understood \in N \wedge hand_d_reject_accept \in N \wedge
  hand_reject_accept \in N \wedge hand_ra \in N \wedge hand_d_inform_failure \in N \wedge
  hand_inform_failure \in N \wedge hand_if \in N \wedge ett \in STATES_E
INITIALISATION
  Exchanged_msg := \emptyset || hand_cfp:=0 || hand_propose:=0 ||
  hand_refuse:=0 || hand_n_understood:=0 || hand_d_reject_accept:=0 ||
  hand_reject_accept:=0 || hand_ra:=0 || hand_d_inform_failure:=0 ||
  hand_if:=0 || hand_inform_failure:=0 || ett:=evt_begin
    
```

By using the Rules 2, 3 and 4, following events are added.

```

Event_cfp = SELECT ett:=evt_begin
  THEN Exchanged_msg:=exchanged_msg \cup
  {initiator \mapsto \{cfp \mapsto participant\}} || ett:=evt_cfp || hand_cfp :=1 ||
  protocol:= active END ;
event_propose = SELECT hand_cfp=1 \wedge hand_propose=0 \wedge ett:=evt_cfp
  THEN Exchanged_msg:=exchanged_msg \cup
  {initiator \mapsto \{propose \mapsto participant\}} || protocol:=active ||
  hand_propose:=1 || ett:= evt_propose END;
event_refuse = SELECT hand_propose=1 \wedge hand_refuse=0 \wedge
  ett:=evt_propose
    
```

```

  THEN Exchanged_msg:=exchanged_msg \cup
  {initiator \mapsto \{refuse \mapsto participant\}} || protocol:=end || hand_refuse:=1 ||
  ett:= evt_refuse END;
event_n_understood = SELECT hand_refuse=1 \wedge
  hand_n_understood=0 \wedge ett:=evt_n_understood
  THEN Exchanged_msg:=exchanged_msg \cup
  {initiator \mapsto \{n_understood \mapsto participant\}} ||
  protocol:=error || ett:=evt_n_understood || hand_n_understood:=1 END ;
    
```

By rule 5, we obtain:

```

Detect_reject_proposal_accept_proposal =
  SELECT hand_n_understood=1 \wedge hand_d_reject_accept=0 \wedge
  ett:=evt_n_understood
  THEN ANY ee WHERE ee \in \{reject_proposal, accept_proposal\}
  THEN msg3:=ee END || hand_d_reject_accept:=1 ||
  ett:= evt_d_reject_accept END ;
event_reject_proposal_accept_proposal = SELECT hand_d_reject_accept=1
  \wedge hand_reject_accept=0 \wedge ett:=evt_d_reject_accept
  THEN Exchanged_msg:=exchanged \cup \{participant \mapsto \{msg3 \mapsto initiator\}\}
  || hand_reject_accept:=1 || protocol:=wait || ett:=evt_reject_accept END ;
event_reject_proposal = SELECT hand_reject_accept=1 \wedge hand_ra=0 \wedge
  ett:=evt_ra \wedge exchanged_msg(participant)= \{ reject_proposal \mapsto initiator\}
  THEN protocol:=end || hand_ra:=1 || ett:= evt_ra END;
event_accept_proposal = SELECT hand_reject_accept=1 \wedge hand_ra=0 \wedge
  ett:=evt_ra \wedge exchanged_msg(participant)= \{ accept_proposal \mapsto initiator\}
  THEN protocol:=active || hand_ra:=1 || ett:= evt_ra END;
Detect_inform_failure =
  SELECT hand_ra=1 \wedge hand_d_inform_failure=0 \wedge ett:=evt_ra
  THEN ANY ee WHERE ee \in \{inform, failure\}
  THEN msg3:=ee END || hand_d_inform_failure:=1 ||
  ett:= evt_d_inform_failure END ;
event_inform_failure = SELECT hand_d_inform_failure=1
  \wedge hand_inform_failure = 0 \wedge ett:=evt_d_inform_failure
  THEN Exchanged_msg:=exchanged \cup \{participant \mapsto \{msg3 \mapsto initiator\}\}
  || hand_inform_failure:=1 || protocol:=wait || ett:=evt_inform_failure END ;
event_inform = SELECT hand_inform_failure =1 \wedge
  hand_if=0 \wedge ett:=evt_inform_failure \wedge
  exchanged_msg(participant)=\{ inform \mapsto initiator\}
  THEN protocol:=end || hand_if:=1 || ett:= evt_if END;
event_failure = SELECT hand_inform_failure =1 \wedge hand_if=0 \wedge
  ett:=evt_inform_failure \wedge
  exchanged_msg(participant)= \{ failure \mapsto initiator\}
  THEN protocol:=active || hand_inform_failure:=1 ||
  ett:= evt_inform_failure END;
OFF= SELECT protocol=active \wedge (time \geq time_out) \wedge system=3
  THEN protocol:=end || ett:=evt_end || time:=0 END;
ON= SELECT protocol=end \wedge time < time_out THEN protocol:=active ||
  ett:=evt_begin END;
    
```

Verification of proof obligations: When the B4free is used, the events (event_propose, event_refuse, event_n_understood) do not verify the properties 1 and 2 (Fig. 8).

This is because the initial protocol diagram is not coherent with the specification. So, we have to modify this diagram as shown in Fig. 9, by adding response delay constraint and the Xor message.

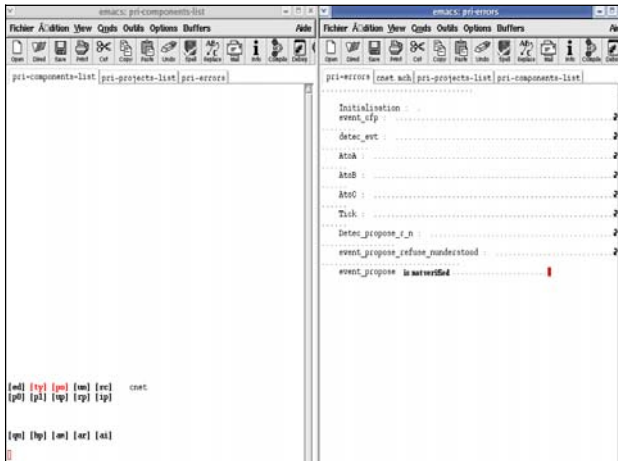


Figure 8. The verification with the B4free (error)

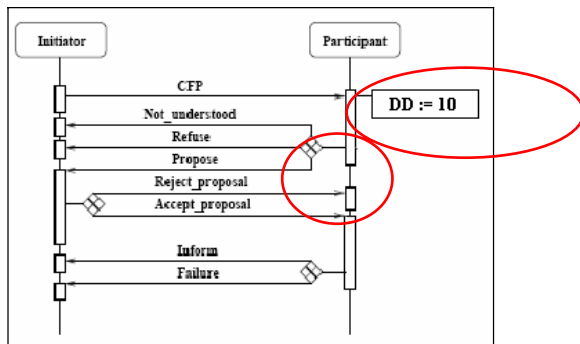


Figure 9. The validated example

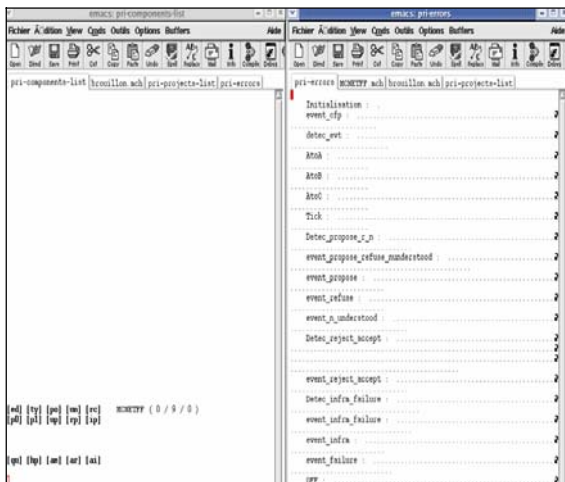


Figure 10. The verification with the B4free (valid)

The verification of proof obligations succeeds as shown in Fig. 10. Hence, the graphical diagram given in Fig. 9 is verified by verifying the derived B model.

II. CONCLUSION

In this paper, we have proposed a specification and verification technique of multi agent interaction protocols using

AUML and Event B. The system is at first modelled with AUML protocol diagrams which is understandable; the resulting model is translated into the Event B notation to verify required properties. This allows one to rigorously verify AUML model by analyzing derived Event B specifications and to prove that the modeled protocol respects all safety and liveness constraints. We have proposed translation rules from AUML protocol diagrams to Event B and we have illustrated these rules over an example: the Contract-Net protocol. Our future focus shall consists of validating transformation rules, studying the correctness of the translation process and proposing other translation rules for concepts that have not yet been considered related to temporal properties.

REFERENCES

- [1] J-R. Abrial, "The B book : Assigning Programs to Meanings," Cambridge University Press, 1996.
- [2] J-R. Abrial, "Extending B without changing it (for developing distributed systems)," First B Conference Putting Into Practice Methods and Tools for Information System Design, Nantes, 1996, pp. 169-190.
- [3] C. Attiogbe, P. Poizat, and G.Salaun., "Integration of Formal Data types within State Diagrams," FASE'2003 - Fundamental Approaches to Software Engineering, LNCS 2621, pp 344-355, 2003.
- [4] I. Bakam, F. Kordon, C. L-Page, and F. Bousquet, "Formalization of a specialized multi-agent system using coloured petri nets for the study of a hunting management system," FAABS, LNCS 1871, 2001, pp. 123-132.
- [5] Clearsy, "B4free," Available at <http://www.b4free.com>, 2004.
- [6] Clearsy, "B language reference manual," Version 1.8.6, 26 février 2007.
- [7] H. Fadil, and J-L. Koning, "Vers une spécification formelle des protocoles d'interaction des systèmes multi-agents en B," 6emeConfrence Francophone de MODlisation et SIMulation, MOSIM'06, Rabat, 2006.
- [8] J. Ferber, "Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence," Addison-wesley Professional, 1999.
- [9] FIPA, Communicative Act Library Specification, technical report, available at : <http://www.fipa.org/specs/fipa00037/>, 2007.
- [10] L. Jemni Ben Ayed, and Y. Hlaoui Ben Daly, "Translating STATEMATE models into FNLOG for the verification of safety requirements in reactive systems," Int. J. Internet Technology and Secured Transactions, Vol. 1, Nos. 3/4, 2009, pp. 236-271.
- [11] L. Jemni Ben Ayed, and F. Siala, "Specification and Verification of Multi-Agent systems interaction protocols using a combination of AUML and Event B," XVth International Workshop on the Design, Verification and Specification of Interactive Systems DSV-IS 2008, LNCS 5136, Kingston, Ontario, Canada, 2008, pp. 102-107.
- [12] C. Metayer, J. Abrial, and L. Voisin, "Event-B Language. Technical Report D7," z RODIN Project Deliverable, 2005.
- [13] J. Odell, H. V-D. Parunak, and B. Bauer, "Extending UML for agents," Proceedings of the Agent Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Austin, 2000.
- [14] A. Regayeg, A.H. Kacem, and M. Jmaiel, "Specification and verification of multi agent applications using temporal z," In Intelligent Agent Technology Conf (IAT.04), IEEE Computer Society, 2004, pp. 260-266.
- [15] J. Rumbaugh, I. Jacobson, and G. Boch, "The Unified Modeling Language reference Manual", Addison- Wesley, 1999.