

Specification and Prototyping of Reactive Distributed Systems with Algebraic Term Nets

Faiza Bouchoul and Mohammed Mostefai
Laboratory of Automatics of Sétif, Ferhat Abbas University, Algeria

Abstract: *The specification of the dynamic behaviour of reactive distributed systems must exhibit the structures of control and has to imply explicitly the relevant aspects of the distribution, such as the concurrency, the reactivity and the interaction between the entities. Among the most common reactive distributed systems we can cite industrial ones; distributed networks occur for example in telecommunications, Internet, power and energy, transportation and manufacturing. Distributed computing will play an increasingly critical role in the global industrial-infrastructure. The need for trustworthy systems has received tremendous researchers' attention. The usage of formal tools for simulation and prototyping designed to facilitate the modelling of such systems is of great interest. Improved methods are needed to insure reliability, security and robustness of industrial distributed systems. This paper proposes the fundamentals of a formal approach for the specification of reactive distributed systems based on object-oriented paradigm. Object's behaviour is modelled as REACTNets. The REACTNets enhance the ECATNets that are a kind of high level algebraic Petri nets with explicit distribution and reactivity. We associate to the classic ECATNets MAUDE rules to handle interactions between objects. The two formalisms have a common semantics in term of rewriting logic so that interesting prospects are opened for their integration.*

Keywords: *Reactive distributed systems, object oriented paradigm, rewriting logic, ECATNets, Maude, rapid prototyping.*

Received February 5, 2008; accepted September 1, 2008

1. Introduction

A distributed system can be seen as a number of heterogeneous and autonomous entities which can interact by the means of suitable interfaces. The complexity of these systems increases with the number of entities which compose them. Various works relating to formal modelling of such systems are continuously proposed for the purpose of verification or rapid prototyping; each one with different objectives, concepts, tools and possibilities.

These models must exhibit the structures of control and have to explicitly imply the relevant aspects of distribution, such as the concurrency, the reactivity and the interaction between the entities. In particular, the expression of concurrency and reactivity constitutes a crucial aspect during the development of the model. Concurrency can arise between the system entities (inter-entities concurrency) and also inside the same entity (intra-entities concurrency). Reactivity deals with the possibility for the system to react dynamically to its environment. Thanks to their logical autonomy and to their modularity, objects are naturally predisposed for the role of concurrent units. They not only make it possible to describe structural properties of the system but also to handle naturally the distribution [14]. However, the object oriented approach presents an evident weakness to suitably express the dynamic aspects of distributed systems.

For this reason, the objects are often enhanced with

formalism for the description of the dynamic aspects of their behaviour. In particular, the approaches associating Petri nets and objects are more and more gaining the interest of several groups of researchers.

The aim of this work is to propose the fundamentals of a formal approach for the specification of reactive distributed systems with true concurrency semantics at inter- and intra-entities level. The idea is to associate the Extended Concurrent Algebraic Term Nets (ECATNets) [8] and the theory of concurrent objects proposed by Meseguer in MAUDE [23, 12]. The ECATNets are a kind of high level algebraic Petri nets with rewriting logic semantics. First we propose the REACTNets that enhance traditional ECATNets with reactivity. REACTNets should be used to describe individual objects' behaviours so that to provide them with a true intra-object concurrency; and to express not only the actions which the object carries out but also its interactions with its environment in term of messages emitting/ receiving.

2. Object/Petri Nets Formalisms

2.1. The Object/Petri Net Complementarity

Object/Petri nets association is based mainly on the interesting complementarity of the two formalisms for the specification of distributed systems. Petri nets deal with the most crucial aspects of concurrency; objects offer necessary tools to express various aspects of

distribution. Furthermore, distributed systems are often reactive and the behaviour of a reactive system is usually modelled by event-condition-actions rules called commonly production rules or simply rules Event Condition Action (ECA). The significance of an ECA rule is that if the event in the environment occurs, and the condition is true, the reactive system performs the action [16]. The problem is that the token-game semantics of Petri nets does not model behaviour of reactive systems, the non-reactivity of the token-game semantics can be seen immediately from the definition of the firing rule.

A transition in a Petri net is enabled once the conditions of firing are true, however the environment of the Petri net does not influence in any way its firing. In contrast, in a reactive system a relevant transition needs some additional input event to become enabled. So, the token-game semantics models closed systems, whereas a reactive system is open, otherwise it cannot interact with its environment. Furthermore, in a reactive system an enabled transition must fire immediately. In the token-game semantics, an enabled transition may fire, but does not necessarily have to. A Reactive Petri net can simply be built by changing for internal transitions the rule "the transition may fire" by the rule "the transition must fire" [16]; while for external transitions expressing the interactions with the environment the traditional rule can be preserved to ensure the network stability. Thus a reactive Petri net has two possible states: stable and unstable. The system must continue to fire the internal enabled transitions as a long time as it does not reach a stable state, in other words until no internal transition is enabled; before being able to fire external transitions from the environment. But, the Petri net must explicitly comprise sufficient constructions to model the interaction with the environment by external transitions handling the events that influence its internal behaviour and expressing the reactivity. For this purpose object paradigm offers to Petri nets an elegant solution. And we can conclude that the complementarity of the two approaches is twofold, on one hand objects need Petri nets to express their dynamic behaviour and on the other hand Petri nets need objects to have modularity and reactivity through object interaction mechanisms.

2.2. Object/Petri Net Approaches: State of the Art

The object/Petri net association is not new, and among the multitude of works integrating objects and Petri nets, two tendencies are distinguishable, designated successively by "Objects in Petri nets" and "Petri nets in objects" [4]. The principle of the "objects in Petri nets" approach is to model a system by a single Petri net, whose tokens are objects. This single network can be structured by using a hierarchical decomposition,

typically in the form of super-transitions or super-places. The type of tokens is described in an external formalism to Petri nets, for instance an algebraic notation or a programming language.

The formalism POP/POT [15] belongs to this type of approaches. Parallel Object-based Transition (POT) system is another example: A POT is a simple Petri net where objects are tokens with associated structures of memories; the state of an object is explicitly modelled by places. Another example is given by LOOPN [22] which is a language for simulation and specification of distributed systems with timed coloured Petri nets. It includes object properties such as the sub-typing, inheritance and polymorphism which allow an adequate modularization of complex specifications.

The "Petri nets in objects" approach consists in using Petri nets to describe the internal behaviour of the objects. This approach proposes to model the system by several independent Petri nets (objects) which can interact. The network marking models the internal state of the object and the transitions model the execution of its methods. The fundamental interest of this type of approach is to allow the use of the concepts resulting from the object paradigm (classification, encapsulation) to describe the structure of the system, instead of using a purely hierarchical structuring.

The Competitor Object Oriented Petri Net (COOPN) [11] and PROTOB [2] belong to this type of formalisms. In particular, PROTOB is a Computer Aided Software Engineering (CASE) for the specification, simulation and prototyping of the concurrent systems. A PROTOB Object is defined by its attributes, actions and communication ports. The behaviour is described by a PROT which is a high level Petri net which integrates Petri nets and DataFlow Diagrams (DFDs). In [26] an other similar formalism is presented: the Colored Timed Object-Oriented Petri-Nets (CTOPN) are proposed for the modeling of the automated manufacturing systems. Objective-Linda [20] is another formalism for the formal specification of active objects' behaviour, using High Level Petri Nets (HLPN). The EP-Nets [18] associating objects and Petri nets are proposed for the modelling of the interactive multi-media orchestrations. In [3] the dynamic model of UML is enhanced by high level timed Petri nets to cover the language gaps. Another example is given by Hierarchical Object-Oriented Petri Net (HOONets). HOONets deal with several oriented object aspects such as abstraction, encapsulation, modularity, interaction by messages, inheritance and polymorphism [21]. However the work closest to our proposed approach is probably the CO-Nets [1]; the CO-Nets constitute a multi-paradigm integrating algebraic Petri nets and the object-oriented paradigm, the model is semantically interpreted by a rewriting logic theory largely inspired from that of ECATNets.

2.3. The Rewriting Logic

The rewriting logic is nothing but a generalization of equational logic in order to adapt it to changes [24]. The rules are similar to those of equational logic but have a completely different significance. A rule $T \Rightarrow T'$ do not mean any more T equal T' but T becomes T' . The rule is a basic action allowing the transition of the system from one state to another. The rewriting logic describes the changes of the system so that the state is represented by an algebraic term, the transition becomes a rewriting rule and the distributed structure, an algebraic structure modulo a set of axioms E . Syntax in logic of rewriting is given by a *signature* (Σ, E) , where Σ is a set of functions and E a set of axioms. A rewriting theory $T=(\Sigma, E, L, R)$ in rewriting logic is composed of a *signature* (Σ, E) and by a set of labelled rules R with labels in L . These rules describe the behaviour of the system and the rewritings are performed on the classes of equivalences of the terms modulo the axioms E . In practice a rewriting theory $T = (\Sigma, E, L, R)$ can be used as an executable specification allowing a rapid prototyping of the modelled system and its checking. One of the most powerful applications of this logic consists of the MAUDE concurrent objects theory; it is a theory enabling description of the system as a configuration of objects. Object systems from simplest to most complex can be modelled in this theory. This theory is at the origin of MAUDE language. The latter is a high level specification language for concurrent oriented objects systems where each elementary action is described by a rewriting logic rule. By integrating functional, object and concurrent programming, MAUDE enables specification of object systems in a declarative way with a high degree of abstraction and generality. MAUDE adopts OBJ3 [17] as a functional sub-language for the specification of data types. The behaviour of the system is described by a set of rewriting rules. Each rule called event of communication can imply several objects and several messages. The object is the unit of concurrency of the system (granule of concurrency) and evolves according to an interleaving semantics. Another application is the ECATNets

Concurrent Algebraic Term Nets (ECATNets) are a kind of high level Petri nets which associate rewriting logic to Petri nets. ECATNets integrate the Numerical Petri Nets NPN [27], Algebraic Data Types (ADTs) and the rewriting logic. By these three formalisms, ECATNets offer a powerful tool for the specification, prototyping and validation of concurrent systems. NPNs and ADTs define structural and syntactic aspects of ECATNets whereas the rewriting logic defines its semantics. ECATNets were subjects to several applications and extensions [5, 7, 6, 10, 13, 9], the last work [19] proposed the RECATNets that enhance the

ECATNets with the recursion and possibilities to specify complex workflow patterns.

3. The REACTNets

The REACTNets results from the integration of the ECATNets and MAUDE; in addition to the advantages of an object/Petri nets association as explained above, the two formalisms have the same semantics based on rewriting logic; on one hand this common semantic enable an homogeneous integration and attenuates the difficulties often encountered during the integration of ad hoc formalisms, on the other hand this association makes it possible to specify not determinist distributed systems with a true concurrency semantics at inter-object level (thanks to MAUDE rules) and intra-object level (thanks to ECATNets); finally the object paradigm adds the distribution and reactivity which are missing in traditional Petri nets to ECATNets.

3.1. The ECAObjects

The object that we call ECAObject, as shown in Figure 1, is described by its structural aspects and its behavioural aspects. The structure of an ECAObject consists of its static description in term of its name (unique identifier), its attributes, its communication ports and the events describing its behaviour. The attributes model the ECAObject's static properties such as:

- Parameters of ECAObject (name, first name, age...).
- States of ECAObject (busy, idle ...).
- References to other ECAObjects.

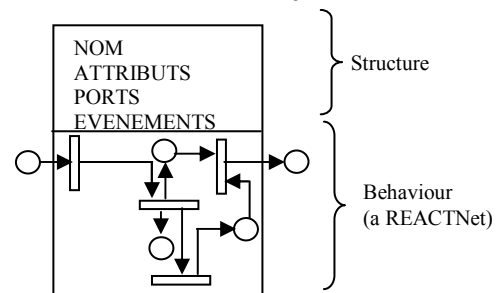


Figure 1. Abstract architecture of an ECAObject.

The ports are the ECAObject's access points used for messages' emission and reception. The current state of an ECAObject is given by the set of its attributes. The event is the elementary activity of the ECAObject dependent on its state and modifying it. It is the granule of its concurrent behaviour. An ECAObject can carry out several events in parallel. The identification of the events depends on the level of abstraction agreed to describe this behaviour. The events can be either internal (local operations in the ECAObject) or visible (emission or interception of messages). The visible events constitute the interface of ECAObject and model the services needed or

offered by him. The behaviour of the ECAObject consists of its dynamic evolution and can be described by the set of its acceptable life cycles. A life cycle represents a possible succession of events implying this ECAObject during its evolution and can comprise concurrency, mutual exclusion, and sequencing. A place may be:

- An attribute of the ECAObject.
- A port for an external interaction.
- An intermediate place added for the needs of specification.

The behavior of an ECAObject is described by a REACTNet exhibiting not only its internal events but also its external events expressing its interaction with the environment through emission or reception of message in specific ports, as shown in Figure 2. The places P-out (emission) and P-in (reception) are communication ports for the ECAObject’s visible events. This case of figure could be brought back to a composition by transition (also called by rendez-vous) of the two Petri nets as shown in Figure 3. It is a particular case of the composition by a sequential process and it was proven that properties of aliveness and boundness are preserved in the composite network [25]. In addition we agree that REACTNets are considered with respect to the stability rules of classical reactive Petri nets theory as presented in [16]. The communication ports allow to specify the simultaneous emission and interception of several different messages in parallel whereas The input/output places of classical Object/Petri nets approaches are generally managed in FIFO in accordance with the traditional vision of communication ports of concurrent objects.

The transition T models an internal event which is an action undertaken by the ECAObject. Let us note here that any change which can affect the state of an ECAObject (its attributes) constitutes a stage of one of its possible life cycles and have to be expressed in the REACTNet

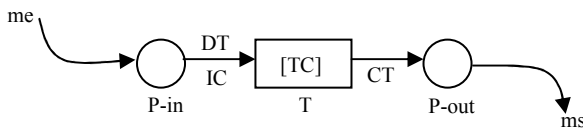


Figure 2. A generic REACTNet.

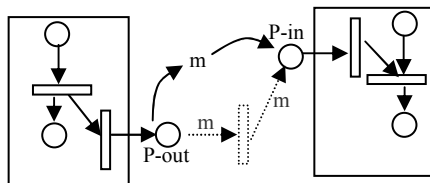


Figure 3. Emission/reception of messages as a composition by “rendez-vous”.

3.2. The REACTNets’ Semantics

The state of the system called configuration is specified as a multi-set of ECAObjects and messages, provided with an operator ACI, with the identity element \emptyset . The pair (P, M (P)) defines the current state of the place P. the set of these pairs (place, marking) has a structure of a multi-set with \otimes union on this multi-set and \emptyset_B the identity element. The state of the ECOBJect is the union (\otimes) of the states of all its places and is expressed with the term $\langle O: C/ P_1: m_1 \dots P_n: m_n \rangle$ where, O: name of the ECAOBJect, C: classe of the ECAOBJect, P_i : i^{th} place of the associated REACTNets, and m_i : marquing of the i^{th} place of the associated REACTNets.

The tokens are algebraic terms: Input Condition (IC), Destroyed Tokens (DT), Created Tokens (CT) are multi-sets of terms (tokens), where $\oplus, \cap, \subset, \ominus$ stand for, respectively, union, intersection, inclusion and difference on the multi-sets and \emptyset_M the element identity. [TC] is a Boolean algebraic expression eventually containing variables appearing in IC, DT and CT. To each place P are associated a sort S(P) and a capacity C(P) defined as a multi-set of closed terms (constants). The marking M(P) of a place is defined in respect of its capacity (which can be infinite).The transition T materializes an internal event and is enabled if the following conditions are true:

- IC(P_1, T) is enabled: IC indicates the multi-set of tokens that have to be present in P_1 .
- TC(T) is true.
- the addition of CT to the place P_2 must not result in exceeding its capacity.

When T is fired:

- The multi-set $M(P_1 \cap DT)$ is removed from the input place P_1 .
- The multi-set CT is added to P_2 .

4. The Rewriting Theory of the System

A REACTNets-based specification has a rewriting logic semantics combining the semantics of ECATNets and that of MAUDE and therefore it is a particular case of a conditional rewriting theory. The rewriting system obtained inherits the four groups of ECATNets’ rules to which we add two other groups derived from MAUDE, the first one expresses the reactivity by the means of the interaction with environment and the second models the creation/destruction of objects.

4.1. Equational Logic Rules

These rules are derived from the algebraic equations describing the types of tokens (by ADTs). Usually, the ECATNets use OBJ3 [17] as a functional sub-language. The evaluation of the tokens can be done using a concurrent equational rewriting.

4.2. Transitions Rules

The form of the rules derived from the transitions depends on the form of IC. The form of the rule is derived from the ECATNets rules as well as MAUDE events of communication in the sense that we explicitly express the object nature of the REACTNet. If we suppose that the generic REACTNet presented at the preceding paragraph is associated to an object O of class C which we represent in accordance with MAUDE notation by the expression $\langle O:C \rangle$, we will have the following cases:

- *Case 1:* IC is of the form $[m]_{\oplus}$
 $IC = DT$

We agree to express the rule as follows:

$$T: \langle O:C/P_1: IC \rangle \Rightarrow \langle O:C/P_2: CT \rangle$$

where expressions $P_1: IC$ and $P_2: CT$ are in conformity with the ECATNet notation, i.e., they respectively express the suppression of IC of P_1 and the addition of CT to P_2 .

$$IC \cap DT = \emptyset_M$$

The multi-set IC must be included in M(P) but does not have to be removed after firing, to express it the idea is to transform IC into itself:

$$T: \langle O:C/P_1: IC; P_1:DT \cap M(P_1) \rangle$$

$$\Rightarrow \langle O:C/P_1:IC; P_2:CT \rangle$$

$$IC \cap DT \neq \emptyset_M$$

For this case, it was shown that it is possible to split the transition T in two transitions T_1 and T_2 of the simple type (two preceding cases) whose simultaneous firing is equivalent to that of T so we derive two rules:

$$T_1: \langle O:C/P_1: IC_1 \rangle \Rightarrow \langle O:C/P_2: CT_1 \rangle$$

$$T_2: \langle O:C/P_1, IC_2 \rangle \otimes \langle O:C/P, DT_2 \rangle$$

$$\Rightarrow \langle O:C/P, IC_2 \rangle \otimes \langle O:C/P_2, CT_2 \rangle \text{ With:}$$

$$IC = IC_1 \cup IC_2, DT = DT_1 \cup DT_2$$

$$IC_1 = DT_1, IC_2 \cap DT_2 = \emptyset_M$$

- *Case 2:* IC is of the form $\sim [m]_{\oplus}$

The form of the rule is given by:

$$T: \langle O:C/P_1: DT \cap M(p) \rangle$$

$$\Rightarrow \langle O:C/P_2: CT \rangle \text{ if } (IC \setminus (IC \cap M(p)) = \emptyset_M) \Rightarrow \text{false}$$

- *Case 3:* $IC = \emptyset_M$

The form of the rule is given by:

$$T: \langle O:C/P_1, DT \cap M(p) \rangle \Rightarrow \langle O:C/P_2, CT \rangle$$

$$\text{if } (M(p) = \emptyset_M) \Rightarrow \text{true}$$

When the place capacity C(p) is finite, the conditional part of the rewrite rule will include the following component:

$$(CT \oplus (M(p) \cap C(p))) \Rightarrow CT \otimes M(p) \text{ (Cap)}$$

In the case where there is a transition condition TC, the conditional part of our rewrite rule must contain the following component: $TC \Rightarrow \text{true}$.

4.2.1. Identity Rules

$$\emptyset_M \oplus X \Rightarrow X$$

$$\emptyset_B \otimes Z \Rightarrow Z$$

4.2.2. Inferences Rules

The two following rules allow by splitting and recombination of the set of tokens, to carry out the rewriting rules with a maximum of concurrency at the level of the ECAObject itself, in fact this splitting/recombination of the state of the ECAObject exhibits explicitly intra-object concurrency which is missing in MAUDE.

Splitting:

$$\langle O:C/P:X \oplus Y \rangle \Rightarrow \langle O:C/P:X \rangle \otimes \langle O:C/P:Y \rangle$$

Recombination:

$$\langle O:C/P:X \rangle \otimes \langle O:C/P:Y \rangle \Rightarrow \langle O:C/P:X \oplus Y \rangle$$

4.2.3. Visible Events Rules

They are asynchronous events related to the ports of the ECAObject. The explicit separation between the communication interface and the other activities for the same object makes it possible to have an additional level of intra-object concurrency. The communications can be done in a completely independent manner of the internal activities.

Intercepting a message

This rule can be expressed according to the adopted syntax as follows:

$$m \langle O:C \rangle \Rightarrow \langle O:C/(P\text{-in}, m) \rangle$$

Emitting a message

The agreed rule is as follows:

$$\langle O:C/(P\text{-out}, m) \rangle \Rightarrow m \langle O:C \rangle$$

4.3. Object Creation/Destruction Rules

The object creation/destruction model considered is borrowed from that of MAUDE and inherits in particular, its declarative nature.

4.3.1. Object Creation

The creation of an object requires a rule which makes it possible to specify explicitly that a message m_C is a creation message, while revealing the object created on the right of the rule in accordance with MAUDE syntax.

Example: $m_C \Rightarrow \langle O:C/S \rangle$

This rule specifies that m_C is a message of creation; the effect is the generation of an object O of class C; S is the initial state of the associated REACTNet, i.e., the pairs set (place: marking) which starts the life cycle of the ECAObject created.

The identity of the ECAObject O and its initial state S can be the message parameters. Creation can be made, as presented in [23] in two stages, initially the sending of a message to a particular object (Meta-object) associated to the class then the emission by this last of the effective message of creation. The objective is to manage the unicity of the identity and the validity of the creation.

4.3.2. Object Destruction

The destruction can be specified by the interaction of a destroying message and the object to be destroyed, which will have to disappear from the right of the rule.

Example: $m_D < O:C > \Rightarrow \emptyset$

Just as for creation, the destruction of an object can be processed by a particular object (a priori the same charged by creation) in order to check that the object to be destroyed really exists and to eliminate it in the affirmative from the list of the objects of the current configuration, by transmitting the destructive message.

5. Case Study: the Router System

The usage of multiple switches to connect test points or devices to instruments for the purpose of testing, measuring or monitoring some systems such as industrial ones is very common. The router system seems to be a good example for our approach. This choice is also motivated by the high degree of parallelism implied in such systems.

5.1. Abstract Specification

The system is composed of several senders and several receivers communicating via the router. A sender emits from a queue of packets. Each emitted packet must be acknowledged. The sender does not send a new packet to a given receiver if its predecessor is not acknowledged yet. The receiver receives the packets in a queue. For each received packet, an acknowledgement is sent to the sender. The router has at a given moment a set of packets and acknowledgements to treat. It can intercept many packets and acknowledgements in parallel and route them in the same time to the receivers.

5.2. A Formal Model for the Router System

The system is composed of three ECAObjects classes: Sender, Receiver and Router. The messages' exchange between these three ECAObjects can be done according to the protocol presented in Figure 4 where S, RT and R, respectively, the ECAObjects of the Sender class, the Router class and the Receiver class. The sender S sends a $Pck(S, D, R)$ message to the router RT who transmits it to the concerned receiver R in the form of the routed $message(S, D, R)$. Data (D) is the contents of the message.

After the message reception, the receiver R returns an acknowledgement $Ack(R, S)$ which is routed to S in the $form(R, S)$. The distinction between packets and acknowledgements before and after routing is necessary since each message type is associated to a distinct visible event. Indeed, the $Pck(S, D, R)$ have to be intercepted by the router RT whereas the $message(S, D, R)$ have to be intercepted by the receiver R.



Figure 4. Interaction protocol between the ECAObjects.

5.2.1. The ECAObject Receiver

Attributes:

Recq : queue of received packets.

Ports:

Ack_out, Pck_in

Internal events:

Treating-Pck: processing of a received packet (queuing in Recq and emission of an acknowledgement)

Visible events:

Output messages: {Ack(R,S) }

Input messages: { (S,D,R) }

5.2.2. The ECAObject Router

Attributes:

Acknowledgement: a set of packages and acknowledgements to be treated at a given moment.

Ports:

Ack_in, Pck_in, Ack_out, Pck_out

Internal events:

Routing-Pck : Routing of a packet.

Routing-Ack : Routing of an acknowledgement.

Visible events:

Output Messages: { (S,D,R), (S, R) }

Input Messages: {Ack(R,S), Pck(S,D,R)}

5.2.3. The ECAObject Sender

Attributes:

Sendq: queue of the packets to emit.

Receiver: identifier of the receiver from which an acknowledgement is expected.

Ports:

Ack_in, Pck_out

Internal events:

Emitting-Pck : this action consists in emitting a packet when the conditions are true (file not empty and no acknowledgement waited from the receiver)

Treating-Ack:

Processing of a received acknowledgement.

Visible events:

Output Messages: { Pck(S,D,R) }.

Input Messages: {(R, S)}.

5.3. The REACTNets of the Router Model

The type Queue[elt] is supposed to be predefined with the usual operations remove, empty, add. We consider the functions *send*, *rec* and *data* which give, respectively, for a packet or an acknowledgement the Sender(S), the Receiver(R) and the Data(D).

5.3.1. The REACTNet “Receiver”

The *packets(S, D, R)* are received in the input port Pck-In, as shown in Figure 5. For each received message, an acknowledgement is emitted via the output port Ack-Out. The data D is added to the file Q in the Recq place. The parameter id used is supposed referring the identity of the object associated to the REACTNet.

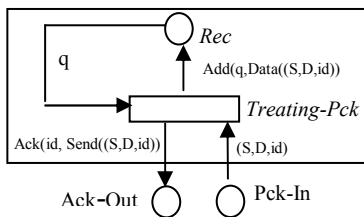


Figure 5. The REACTNet receiver.

5.3.2. The REACTNet “Sender”

Sendq contains the file Q of the packets (*Pck(S, D, R)*) to emit. The packets are emitted via the output port Pck-Out, as shown in Figure 6. For any emission a reference of the receiver R whose a acknowledgement is awaited is stored in the Receiver place. A packet *Pck(S, D, R)* is emitted only if no acknowledgement is awaited from the receiver R. The expression $\sim Rec(Head(Q))$ expresses that the identity of the receiver of the packet at the head of file should not be in the Receiver place and \emptyset_m indicates that no token is destroyed. The acknowledgements are received in the port Ack-In.

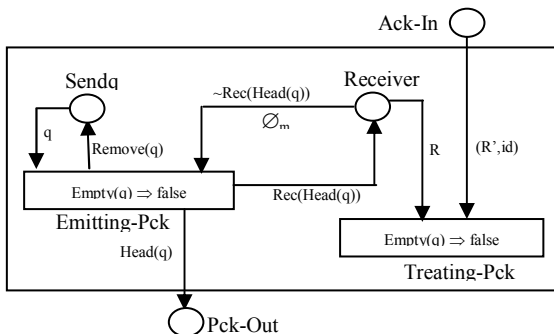


Figure 6. The REACTNet sender.

5.3.3. The REACTNet “Router”

The Router ECAObject, shown in Figure 7, has two input ports Pck-In and Ack-In, respectively for the packets and the acknowledgements and two output ports Pck-Out and Ack-Out. messages and are collected in the place “Messages”. The transitions

receiving-Pck and Receiving-Ack are used to pass the received messages of the input ports to the place “messages”.

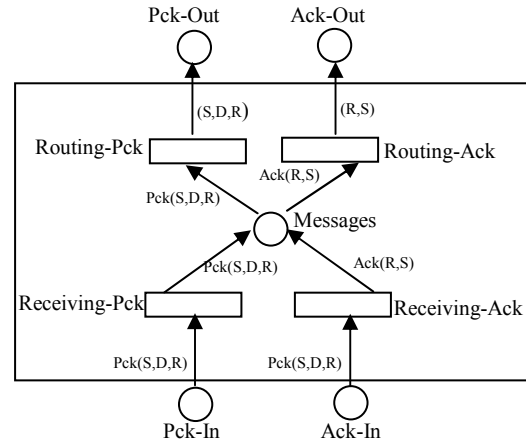


Figure 7. The REACTNet router.

6. Specification of the System

6.1. The Object Module “ROUTER”

The module object ROUTER (Box given below) is the specification in the adopted notation of the example of the router introduced in precedent paragraphs. The following types are supposed to be predefined: Mset[elt] (multi-set of elements), Queue[elt] (file of elements) and Bits (sequence of bits). Syntax is borrowed of course from MAUDE but with different concurrency semantics, MAUDE has an interleaving semantics whereas our approach has true concurrency semantics.

6.2. A Prototyping Scenario

We show in what follows how the specification above can be used for a rapid prototyping of the system, we start from a given configuration and execute the prototype. For the lack of space we omit intermediate states.

- *Initial Configuration*
 $\langle RT:Router/(Pck-Out, \emptyset_m) \otimes (Ack-Out, \emptyset_m) \otimes (Messages, \emptyset_m) \otimes (Pck-In, \emptyset_m) \otimes (Ack-In, \emptyset_m) \rangle$
 $\langle S1:Sender/(Sendq, Pck(S1, D1, R1), Pck(S1, D2, R2)) \otimes (Pck-Out, \emptyset_m) \otimes (Ack-In, \emptyset_m) \otimes (Receiver, \emptyset_m) \rangle$
 $\langle S2:Sender/(Sendq, Pck(S2, D3, R2)) \otimes (Pck-Out, \emptyset_m) \otimes (Ack-In, \emptyset_m) \otimes (Receiver, \emptyset_m) \rangle$
 $\langle R1:Receiver/(Recq, \emptyset_m) \otimes (Ack-Out, \emptyset_m) \otimes (Pck-In, \emptyset_m) \rangle$ $\langle R2:Receiver/(Recq, \emptyset_m) \otimes (Ack-Out, \emptyset_m) \otimes (Pck-In, \emptyset_m) \rangle$
- *Step (1)*
Firable rules:
Object S1: Emitting-Pck
Object S2: Emitting-Pck
Concurrency
Inter-object between S1 and S2.
- *Step (2)*
Firable rules:

- Object S1: R1
Emitting-Pck
Object S2: R1
concurrency
Inter-object between S1 and S2
Intra-object for S1
- Step (3)
Firable rules:
Object S1: R1
Object RT: R100 (twice)
Concurrency
Inter-object between S1 and RT
Intra-object for RT
 - Step (4)
Firable rules:
Object RT : R100
Receiving-Pck (twice)
Concurrency
Intra-object for RT
 - Step (5)
Firable rules:
Object RT: Receiving-Pck
Final Configuration
 $\langle RT:Router/(Pck-Out, (S1, D1, R1) \oplus (S2, D3, R2)) \otimes (Ack-Out, \emptyset_m) \otimes (Messages, Pck(S1, D2, R2)) \otimes (Pck-In, \emptyset_m) \otimes (Ack-In, \emptyset_m) \rangle$
 $\langle S1:Sender/(Sendq, \emptyset_m) \otimes (Pck-Out, \emptyset_m) \otimes (Ack-In, \emptyset_m) \otimes (Receiver, R1 \oplus R2) \rangle$
 $\langle S2:Sender/(Sendq, \emptyset_m) \otimes (Pck-Out, \emptyset_m) \otimes (Ack-In, \emptyset_m) \otimes (Receiver, R2) \rangle$
 $\langle R1:Receiver/(Recq, \emptyset_m) \otimes (Ack-Out, \emptyset_m) \otimes (Pck-In, \emptyset_m) \rangle$
 $\langle R2:Receiver/(Recq, \emptyset_m) \otimes (Ack-Out, \emptyset_m) \otimes (Pck-In, \emptyset_m) \rangle$

7. Conclusion

In this paper, we have proposed the fundamentals of a new approach for the specification of object-oriented distributed systems with true concurrency at both intra and inter-object levels. We associate two formalisms, the ECATNets and MAUDE. Thus bring together the advantages of both formalisms: the high degree of concurrency and expressiveness of the ECATNets and the object-orienteness of MAUDE. The main strength of our approach is probably its rewriting logic semantics; therefore, the obtained prototype can be executed and analyzed under the MAUDE environment.

Appendix

OMOD ROUTER

protecting configuration / specification of the configuration withsortes msg, objects and Oid(object identifier) and communication events

protecting Queue[elt]

protecting Mset[elt]

protecting Bits

make Msg-queue is Queue[msg] endmk

make Msg-mset is Mset[msg] endmk

msg Pck(-,-) : Oid Bits Oid \rightarrow msg

msg (-,-) : Oid Bits Oid \rightarrow msg

msg Ack(-,-) : Oid Oid \rightarrow msg

msg (-,-) : Oid Oid \rightarrow msg

var q:msg-Queue

var S,R,R',RT : Oid

var D : Bits

Class Sender / Atts: Sendq:Msg-queue, Receiver:Oid; Ports :Ack-In,Pck-Out:Msg-mset

Emitting-Pck : $\langle S:Sender/(Sendq,q) \otimes Receiver, \emptyset_m \rangle$

$\Rightarrow \langle S:Sender/(Sendq,Remove(q)) \otimes (Pck-Out,Head(q))$

$\otimes (Receiver, Rec(Head(q))) \rangle$ if $((Empty(q) \Rightarrow false)$ and

$(M(Receiver) \otimes (M(Receiver) \cap Rec(Head(q))) =$

$\emptyset_m \Rightarrow false)$

Treating-Ack : $\langle S:Sender/(Receiver,R) \otimes (Ack-In, (R',S)) \rangle$

$\Rightarrow \emptyset_B$ if $((R=Rec((R',S))) \Rightarrow true)$

R1 : $\langle S:Sender/(Pck-Out,Pck(S,D,R)) \rangle \Rightarrow \langle S:Sender$

$Pck(S,D,R) / Pck-Out$ rule

R2 : $(R',S) \langle S:Sender \rangle \Rightarrow \langle S:Sender/(Ack-In, (R',S)) \rangle /$

$Ack-In$ rule

Class Receiver /Atts: Recq:Msg-queue; Ports :Ack-

Out,Pck-In:Msg-mset.

Treating-Pck : $\langle R:Receiver/(Pck-In,(S,D,R)) \otimes (Recq,q) \rangle$

\Rightarrow

$\langle R:Receiver/(Recq, Add(q,Data((S,D,R)))) \otimes (Ack-$

$Out,Ack(R,Send(S,D,R)) \rangle$

R10 : $\langle R:Receiver,(Ack-Out,Ack(R,S)) \rangle \Rightarrow \langle R:Receiver$

$Ack(R,S) /$ règle associée à la place Ack-Out.

R20 : $(S,D,R) \langle R:Receiver \rangle \Rightarrow \langle R:Receiver/(Pck-$

$In,(S,D,R)) \rangle /$ règle associée à la place Pck-In.

Class Router / Atts :Messages; Ports: Pck-Out,Ack-

out,Ack-In,Pck-In : Msg-mset

Receiving-Pck : $\langle RT:Router/(Pck-In,Pck(S,D,R)) \rangle \Rightarrow$

$\langle RT:Router/(Messages,Pck(S,D,R)) \rangle$

Receiving-Ack : $\langle RT:Router/(Ack-in,Ack(R,S)) \rangle \Rightarrow$

$\langle RT:Router/(Messages,Ack(R,S)) \rangle$

Routing-Pck : $\langle RT:Router/(Messages,Pck(S,D,R)) \rangle$

$\Rightarrow \langle RT:Router/(Pck-Out, (S,D,R)) \rangle$

Routing-Ack : $\langle RT:Router/(Messages,Ack(R,S)) \rangle \Rightarrow$

$\langle RT:Router/(Ack-Out, (R,S)) \rangle$

R100 : $Pck(S,D,R) \langle RT:Router \rangle \Rightarrow \langle RT:Router/(Pck-$

$In,Pck(S,D,R)) \rangle /$ règle associée à la place Pck-In.

R200 : $Ack(R,S) \langle RT:Router \rangle \Rightarrow \langle RT:Router/(Ack-In,$

$Ack(R,S)) \rangle /$ règle associée à la place Ack-In.

R300 : $\langle RT:Router/(Pck-Out, (S,D,R)) \rangle \Rightarrow \langle RT:Router$

$(S,D,R) /$ règle associée à la place Pck-Out.

R400 : $\langle RT:Router/(Ack-Out, (R,S)) \rangle \Rightarrow \langle RT:Router$

$(R,S) /$ règle associée à la place Ack-Out.

ENDOMOD

References

- [1] Aoumeur N. and Saake G., "A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems," *Data and Knowledge Engineering*, vol. 42, no. 2, pp. 143-187, 2002.
- [2] Baldassari M., Bruno G., and Castella A., "PROTOB: An Object Oriented CASE Tool for Modelling and Prototyping Distributed Systems," *Software Practice and Experience*, vol. 21, no. 8, pp. 823-844, 1991.

- [3] Baresi L. and Pezzè M., "Improving UML with Petri Nets," *Electronic Notes in Theoretical Computer Science*, vol. 44, no. 4, pp. 107-119, 2001.
- [4] Bastide R., "Approaches in Unifying Petri Nets and the Object-Orientation Approach," in the *1st Workshop on Object-Oriented Programming and Models of Concurrency*, Italy, 1995.
- [5] Bettaz M. and Maouche M., "How to Specify Non-Determinism and True Concurrency With Algebraic Term Nets," *Lecture Notes in Computer Science*, vol. 655, pp 164-180, 1992.
- [6] Bettaz M. and Maouche M., "Modeling of Object Based Systems with Hidden Sorted ECATNets," in the *Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 307-311, USA, 1995.
- [7] Bettaz M. and Maouche M., "On the Specification of Protocol Objects," in the *Proceedings of the 10th ADT and Compass General Meeting*, Italy, 1994.
- [8] Bettaz M., Maouche M., Soualmi M., and Boukebeche M., "Protocol Specification Using ECATNets," *Networking and Distributed Computing*, vol. 3, no. 1, pp. 7-35, 1993.
- [9] Boudiaf N., Barkaoui K., and Chaoui A., "Implémentation des règles de Réduction Des ECATNets Dans MAUDE," in *6th Conférence Francophone de MOdélisation et SIMulation*, pp. 505-514, Morocco, 2006.
- [10] Bounoua O. and Bettaz M., "A Graphical Editor-Simulator for Algebraic Term Nets," in the *Proceedings of the Second Maghrebian Conference on Software Engineering and Artificial Intelligence*, pp. 2-16, 1992.
- [11] Buchs D. and Guelfi N., "A Formal Specification Framework for Object-Oriented Distributed Systems," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 635-652, 2000.
- [12] Clavel M., Durán F., Eker S., Lincoln P., Martí-Oliet N., Meseguer J., and Talcott C., "The MAUDE 2.0 System," in *Proceedings of Rewriting Techniques and Applications*, Springer-Verlag LNCS 2706, pp. 76-87, 2003.
- [13] Djemame K., Gilles D., Mackenzie L., and Bettaz M., "Performance Comparison Of High-Level Algebraic Nets Distributed Simulation Protocols," *Computer of Journal of Systems Architecture*, vol. 44, no. 6-7, pp 457-472, 1998.
- [14] Dovland J., Johnsen E., and Owe O., "Observable Behavior of Dynamic Systems: Component Reasoning for Concurrent Objects," *Electronic Notes in Theoretical Computer Science*, vol. 203, pp. 19-34, 2008.
- [15] Engelfriet J., Leth G., and Rozenberg G., "Net Based Description of Parallel Object Based Systems or POTs and POPs," *Technical Report*, Noordwij Kerhool Fool Workshop, 1990.
- [16] Eshuis R. and Dehnert J., "Reactive Petri Nets for Workflow Modelling," in *Application and Theory of Petri Nets*, pp. 295-314, Eindhoven, 2003.
- [17] Goguen C., Kirshner H., Meseguer J., Megrelis A., and Winkler T., "An Introduction to OBJ3," in *Proceedings of the 1st International Workshop on Conditional Term Rewriting Systems*, pp. 258-263, France, 1987.
- [18] Guan S. and Lim S., "Modeling with Enhanced Prioritized Petri Nets: EP-Nets," *Computer Communications*, vol. 25, no. 8, pp. 812-824, 2002.
- [19] Hicheur A., Barkaoui K., and Boudiaf N., "Modeling Workflows with Recursive ECATNets," in the *Proceedings of 8th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 389-398, Romania, 2006.
- [20] Holvoet T. and Kielmann T., "Behaviour Specification of Parallel Active Objects," *Parallel Computing*, vol. 24, no. 7, pp. 1107-1135, 1998.
- [21] Hong E. and Bae H., "Software Modelling and Analysis Using a Hierarchical Object-Oriented Petri Net," *Information Sciences*, vol. 130, no. 1-4, pp 133-164, 2000.
- [22] Lakos A. and Keen D., "Modelling Layered Protocols in LOOPN," in the *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*, pp. 22-30, Melbourne, 1991.
- [23] Meseguer J., "A Logical Theory of Concurrent Objects and Its Realization in the MAUDE Language," *Technical Report*, SRI International, 1992.
- [24] Meseguer J., "Conditional Rewriting Logic as a Unified Model of Concurrency," *Theoretical Computer Science*, vol. 96, no. 1, pp. 73-155.
- [25] Souissi Y. and Memmi G., "Composition of Nets ViaaA Communication Medium," *Advances in Petri Nets*, LNCS 483, pp. 457-470, 1990.
- [26] Wang C. and Wu S., "Modeling with Colored Timed Object-Oriented Petri Nets for Automated Manufacturing Systems," *Computers and Industrial Engineering*, vol. 34, no. 2, pp. 463-480, 1998.
- [27] Wilbur Ham M., "Numerical Petri Nets: A Guide Version 2," *Telecom Australia*, Research Laboratories, 1987.



Faiza Bouchoul is a lecturer in software engineering at Ferhat Abbas University, Algeria, she holds engineer diploma in software engineering at Ferhat Abbas University, and Magister Diploma at Mentoury University, Algeria, currently preparing Doctorate Diploma and activates as a research member in LAS laboratory in Ferhat Abbas university, she's working on paradigms and tools for specifying and verifying modern manufacturing systems models.



Mohammed Mostefai is senior lecturer in automatism with the title of professor at Ferhat Abbas University, Algeria. He holds engineer Diploma in electronics in 1988 at Ferhat Abbas University, the Masters and then the PhD of automatic and industrial software at Lille University, France in 1994. His research interests are in the area of monitoring of complex systems.