Sciences & Technologie **B**– **N°24**, décembre (2005), pp. 47-56.

CIRTA: A FORMAL LANGUAGE FOR MODULAR ECATNETS SPECIFICATION

Reçu le13/06/2004 – Accepté le 12/11/2005

Résumé

CIRTA (*"**C**onstruction **I**ncrémentale des **R**éseaux de Petri à **T**ermes **A**lgébriques"*) est un langage de spécification dotant les ECATNets (*"**E**xtended **C**oncurrent **A**lgebraic **T**erms **N**ets"*) [6][10] avec des concepts de modularité leur permettant d'être plus appropriés pour des applications réelles. Cet article examine les mécanismes de structuration fournis par CIRTA, pour la conception des systèmes concurrents complexes. Deux techniques de structuration sont présentées. La première se base sur l'utilisation des *modules CIRTA* qui étendent les ECATNets avec les concepts de *nœuds interfaces* et de *nœuds composés*. Le second mécanisme concerne quelques opérations de structuration des modules CIRTA telles que : *l'importation, la composition* et *le renommage*. La sémantique de chaque spécification CIRTA utilisant ces mécanismes est définie par la donnée de l'ECATNet ayant le comportement équivalent.

Mots clés: *Langage de spécification, réseaux de Petri, ECATNets, spécification algébrique.*

Abstract

CIRTA (*"**C**onstruction **I**ncrémentale des **R**éseaux de Petri à **T**ermes **A**lgébriques"*) is a specification language endowing ECATNets (*"**E**xtended **C**oncurrent **A**lgebraic **T**erms **N**ets"*) [6][10] with modularity concepts to make them more suitable for real-world applications. This paper addresses the structuring mechanisms provided by CIRTA, for the design of complex concurrent systems. Two structuring techniques are presented. The first one relies on the usage of *CIRTA modules* which extend ECATNets with the concepts of *interface nodes* and *composed-nodes*. The second mechanism concerns with some *structuring operations* on CIRTA modules namely: *importation*, *composition* and *renaming*. The semantics of each CIRTA specification using these constructs is defined by giving the behavioral equivalent ECATNet.

Key words: *Specification language, Petri nets, ECATNets, algebraic specification.*

N. ZEGHIB[1]
M. BETTAZ[2]

[1]Département d'Informatique
Université Mentouri-
Constantine. Algérie.
[2]Department of Computer
Science.Philadelphia
University- Amman Jordan

ملخص

سيرتا (بناء تزايدي لشبكات بيتري ذات الحدود الجبرية) هي لغة وصف تعزز الشبكات المتنافسة الموسعة ذات الحدود الجبرية بمفاهيم التركيب لتجعلها مهيئة للتطبيقات الواقعية. هذه المقالة تدرس ميكانزمات البناء التركيبي التي توفرها لغة سيرتا لتصميم الأنظمة المتنافسة المعقدة. هناك تقنيتان للبناء التركيبي. التقنية الأولى ترتكز على استعمال "وحدات الوصف" التي توسع الشبكات المتنافسة الموسعة ذات الحدود الجبرية بمفهومي"العقد المشتركة" و "العقد المركبة". أما التقنية الثانية فتخص بعض عمليات التركيب التي تطبق على وحدات الوصف و نخص بالذكر: "الاستيراد"، "التجميع" و" إعادة التسمية". المعنى الرياضي لكل وصف في لغة سيرتا يستعمل هذه التقنيات هو معرف بإعطاء الشبكة المتنافسة الموسعة ذات الحدود الجبرية المكافئة.

الكلمات المفتاحية: *لغة الوصف، شبكات بيتري، الشبكات المتنافسة الموسعة ذات الحدود الجبرية، الوصف الجبري.*

Concurrent systems are characterized by their dedicated function, real-time behavior, and high requirements on reliability and correctness. In order to devise systems with such features, the design process must be based upon a formal specification that captures the characteristics of concurrent systems. Many computational models have been proposed in the literature to specify such systems, including extensions to finite-state machines, data-flow graphs, and communicating processes. Particularly, Petri nets (PNs) are interesting for the specification of this sort of systems: for instance, they may specify parallel as well as sequential activities and they easily capture non-deterministic behaviors. PNs have been extended in various ways to fit the most relevant aspects of concurrent systems. We can find several PN-based models with different flavors in [2][3][4]. ECATNets [5][8][10] are high-level Petri nets model, in which tokens are algebraic terms [17] holding information. The important intrinsic features of ECATNets are their concurrency and asynchronous nature. These features together with their flexibility have stimulated their application in different areas [5][6][10]. However, the main weakness of classical ECATNets pointed out along the years, is the lack of modularity, forcing the system designer to cope with many details at the same time. In order to develop and analyze complex systems, the system developers need structuring and abstraction concepts that allow them to work with selected part of the specification without being distracted by the low-level details of remaining parts. Therefore, the use of techniques to build up compact specifications through the use of a "divide to conquer" strategy is nowadays, commonly accepted as necessary. Common techniques use different levels of abstraction enabling the construction of the specification in an incremental way. Consequently, for large and complex systems, an adequate specification formalism must deal with modularity concepts. To overcome these limitations of ECATNets, the CIRTA [24] language has recently been introduced as formalism for specifying complex and concurrent systems.

It extends ECATNets with module concept and structuring operations so that large systems can be specified and understood stepwisely. The design of complex systems may thus be reduced to the design of simpler and more manageable modules. In this paper, we provide a formal definition of CIRTA modules, and we precise the formal semantics for each structuring mechanism by giving the equivalent (not modular) ECATNet.

- **Paper organization**

The remainder of this paper is organized as follows. Section.2 gives an overview of ECATNets. Section.3 shows how ECATNets are extended by the concept of CIRTA *module*. Section.4 defines the syntax and the semantics of some structuring operations on CIRTA modules. In Section.5, an example of specification is presented to illustrate the use of CIRTA language. Section.6 proposes an approach to translate CIRTA specifications into rewriting theories to allow formal analysis of complex concurrent systems. Some concluding remarks are set in Section.7.

AN OVERVIEW OF ECATNETS

ECATNets are High-Level nets [6][10][8][5] devoted to modeling non-deterministic and concurrent systems. They allow to describe complex systems, using Petri nets [2][19] to model synchronization constraints and abstract data types [17] for specifying the data structures.

Definition

An *ECATNet* is a pair **E** =(Σpec, **N**) where Σpec = (Σ,E) is an algebraic specification and

N = (P, T, σ, Cap, IC, DT, CT, TC, M) is a net such that:

- Σ=(S, Op) where S is a set of sorts and Op is a set of operations.
- E: is a set of Σ-equations.
- P: a finite set of places.
- T: a finite set of transitions $P \cap T = \emptyset$.
- σ: P →S is a map which associates a sort to each place.
- Cap :P → 4-{0} is the place capacity function.
- IC : PxT → $mT_{\Sigma}(V) \cup \{\sim\alpha \,/\, \alpha \in mT_{\Sigma}(V)\} \cup \{empty\}$ is the ***I****nput* ***C****ondition* function.

($mT_{\Sigma}(V)$ denotes the set of ∑-terms multisets with variables in V)

- DT : PxT → $mT_{\Sigma}(V) \cup \{\forall\}$ specifies the ***D****estroyed* ***T****okens*.
- CT : PxT →$mT_{\Sigma}(V)$ is a function defining the ***C****reated* ***T****okens*.
- TC :T→ $T_{\Sigma,bool}(V)$ is the ***T****ransition* ***C****ondition*.

($T_{\Sigma,bool}(V)$ is the set of ∑-terms of sort *bool* using variables in V)

- M : P → $mT_{\Sigma}(\emptyset)$ is a *marking* of the net , such that:

$\forall p \in P \ \ (M(p) \in mT_{\Sigma,s}(\emptyset)) \wedge (\sigma(p)=s) \wedge (|M(p)| \leq Cap(p))$

The graphical representation of a generic net **N** is given in Figure 1, where *p:s,n* denotes a place *p* of sort *s* and capacity *n* (we note *p:s* a place *p* with infinite capacity).
It should be noted that for notation convenience, we omit DT(p,t) (or CT(p,t)) in the graphical representation of the net when IC(p,t)=DT(p,t).

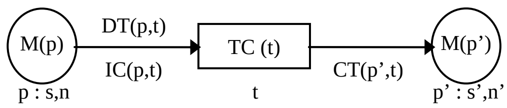


Figure1: A generic representation of an ECATNet.

At any time, a transition *t* is enabled to fire in making *M*, when various conditions are simultaneously true. The first condition is that every *IC(p,t)* for each input place *p* is enabled (as shown in Figure 2). The second condition is that the transition condition *TC(t)* is true. Finally the addition of *CT(p',t)* to each output place *p'* must not result in *p'* exceeding its capacity when this capacity is finite. When *t* is fired, *DT(p,t)* is removed from the input place *p* and simultaneously *CT(p',t)* is added to the output place *p'*.

IC(p,t)	***Enabling conditions***
IC(p,t)=α where $\alpha \in mT_{\Sigma}(V)$	$\alpha \subseteq M(p)$
IC(p,t)=~α where $\alpha \in mT_{\Sigma}(V)$	not ($\alpha \subseteq M(p)$)
IC(p,t)= empty	M(p)= ∅

DT(p,t)	***Destroyed tokens***
DT(p,t)=α where $\alpha \in mT_{\Sigma}(V)$	M(p):= M(p)-α
DT(p,t)= ∀	M(p) := ∅

Figur.2: Input conditions and destroyed tokens.

Example

Figure 3 shows a simple example used to illustrate the main characteristics of ECATNets.

Spec E_0 **is**
  sort data
  op a :→data
  op b :→data
  op f(_) :data→data
  vars X,Y : data
  eq f(a)=b
  eq f(b)=a

end

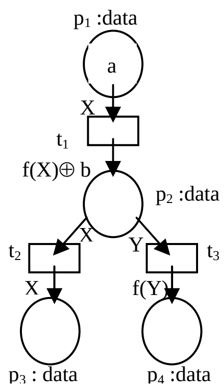


Figure 3: An ECATNet example.

The tokens in the net are of sort *data*. The algebraic specification of this sort is given by defining two constants (*a* and *b*) and an unary operation *f* which semantics is given by two equations (*f(a)= b* and *f(b)=a*). The *Places* of the ECATNet, drawn as circles or ellipses, represent the possible states of the data. The actions performed by this system are indicated by means of rectangles called *transitions*. Places and transitions are connected by directed arcs, which are annotated by algebraic terms. In the initial state only transition t_1 is enabled (may fire). When t_1 fires, the token a ($X=a$) is removed from place p_1 and simultaneously two tokens b ($f(X)=b$) are added to p_2. Hence transitions t_2 and t_3 becomes enabled. In commonly known algebraic nets, only one transition in this case is arbitrary selected and fired; while in ECATNets, *concurrent* as well as *sequential* firing are possible. Hence after the firing of t_1 in the example of Figure.3, we can have one of the following execution sequences: $t_1;t_2$ $t_2;t_1$ $t_1;t_1$ $t_2;t_2$ $t_1//t_2$ $t_1//t_1$ $t_2//t_2$ (where $t_i;t_j$ denotes the sequential execution of t_i and t_j , and $t_i//t_j$ is the concurrent firing of t_i and t_j).

CIRTA MODULE

The modularity mechanism is defined in the "common" way used by *top-down* and *bottom-up* approaches, supporting refinements and abstractions, and it is based on the concept of *module*. In CIRTA language, a module is stored in a *page*. Every page may be used several times in the same specification. The pages with references to a given module $\mathcal{M}$ are referred as super-page (upper-level pages) of $\mathcal{M}$, while the pages contained in the module $\mathcal{M}$ are referred as sub-pages (lower-level pages).

Intuitively, a CIRTA module contains three types of nodes (a node is either a place or a transition): *composed nodes*, *elementary interface nodes* and *elementary non-interface* nodes. Distinctive graphical notations are used for the representation of each type of nodes as shown in Figure4.

	Place	Transition
Elementary non-interface nodes		
Elementary interface nodes		
Composed nodes		

Figure 4: Graphical representation of nodes in CIRTA modules.

- ***Elementary non-interface nodes***

An elementary non-interface node is an ordinary place or transition as commonly known in the Petri nets formalism. These nodes are private to the module in which they are declared, and consequently they are not accessible to other modules.

- ***Elementary interface nodes***

Informally, interface nodes in a CIRTA module is a set of places and/or transitions which are *shared* with other modules. The intuitive idea behind sharing places is to allow the user to specify that all instances of an interface place are considered to be identical, i.e., they all represent a single place even though they are drawn as individual places in different modules. This means that they have the same initial marking and when a token is added/removed to/from one instance of this place, an identical token has to be added/removed to/from all other instances of it. Similarly, all instances of an interface transition have to be seen as representing one indivisible action. Each instance of an interface transition describes a part of a more complex action and all parts must occur together. An interface transition is enabled if all its instances in all modules are enabled. The change produced by the firing of an interface transition is the sum of changes produced by all its instances. For simplicity, each interface node is considered having the same name in all modules.

- ***Composed nodes***

The composed nodes allow the user to relate a node to a more complex module which usually gives a more precise and detailed description of the activity represented by the composed node. The idea is analogous to the hierarchy constructs found in many graphical description languages (such data flow diagrams). At one level, we want to give a simple description of the specified system without having to consider internal details about how it is carried out. At another level, we want to specify the more detailed behavior. Every composed-node has an associated sub-page. However, the places connected to a composed-transition exist only at the super-page level; the places presented at the sub-page will be merged with the associated places at the super-page. As far as one composed-transition can not be executed like an ordinary transition and also that a composed-place cannot be considered as an ordinary place (which means that it cannot hold a marking), they act just as a graphical modeling convenience enabling the designer to structure the graphical model in an expressive way.

Definition

A *CIRTA module* is a tuple $\mathcal{M}$ =(Σpec, **N,** Pageset, π, δ) where Σpec = (Σ, E) is an algebraic specification and **N** = (P, T, τ, σ, Cap, IC, DT, CT, TC, M) Such that:

- Σ=(S, Op) where S is a set of sorts and Op is a set of operations.
- E: is a set of Σ-equations.
- P: a finite set of places.
- T: a finite set of transitions $P \cap T = \emptyset$.
- τ: $P \cup T \rightarrow \{e, c, i\}$ is a map which associates a type for each node of the net:
  - $\tau(n)$ = e if n is an elementary non-interface node.
  - $\tau(n)$ = c if n is a composed node.
  - $\tau(n)$ = i if n is an elementary interface node.
- σ: $\{p \in P/\tau(p) \neq c\} \rightarrow S$ is a map which associates a sort to each no-composed place.
- Cap : $\{p \in P/\tau(p) \neq c\} \rightarrow 4\text{-}\{0\}$ is the place capacity function.
- IC : $PxT \rightarrow mT_{\Sigma}(V) \cup \{\sim\alpha / \alpha \in mT_{\Sigma}(V)\} \cup \{empty\}$ is the Input Condition.
- DT : $PxT \rightarrow mT_{\Sigma}(V) \cup \{\forall\}$ is the Destroyed Tokens.
- CT : $PxT \rightarrow mT_{\Sigma}(V)$ Created Tokens.
- TC : $\{t \in T/\tau(p) \neq c\} \rightarrow T_{\Sigma,bool}(V)$ Transition Condition.

- M : $\{p \in P/\tau(p) \neq c\} \rightarrow mT_{\Sigma}(\varnothing)$ is a marking of the net, such that: $\forall p \in \{p \in P/\tau(p) \neq c\}$ ($M(p) \in mT_{\Sigma,s}(\varnothing)$) $\wedge$ $(\sigma(p)=s) \wedge (|M(p)| \leq Cap(p))$
- Pageset: is a finite set of pages (the sub-pages of $\mathcal{M}$)
- π: $\{n \in P \cup T/\tau(n) = c\} \rightarrow$ Pageset is the map of page assignment.
- δ: $\{n \in P \cup T \;/\; \tau(n) = c\} \rightarrow$ Br is a port assignment function. It is defined from composed nodes into binary relations such that $\delta(n) \subseteq {}^{\bullet}n^{\bullet} \times Node(\pi(n))$ (where ${}^{\bullet}n^{\bullet}$ denotes adjacent nodes of n, and Node($\mathcal{M}$) is the set of nodes in module $\mathcal{M}$))

Example

For instance, let us consider the CIRTA module $\mathcal{M}_1$ of Figure 5. $\mathcal{M}_1$ has two elementary interface places p_1 and p_3 which are shared with other modules. Apart from t1 which is a composed transition, all other nodes are elementary non-interface nodes. The part $\mathcal{M}_2$ in Figure 6 is a sub-page describing the refinement of the composed transition t_1, the functions τ, π and δ are defined by:

$\tau(p_1)=\tau(p_3)=i$, $\tau(t_1)=c$, $\tau(t_2)=\tau(t_3)=\tau(p_2)=\tau(p_4)=e$, Pageset $=\{\mathcal{M}_2\}$, $\pi(t_1)=\mathcal{M}_2$ $\delta(t_1)=\{(p_1,p_1'), (p_2, p_2')\}$

Spec $\mathcal{M}_1$ **is**
Use $\mathcal{M}_2$
op f(_) :data→data
vars X,Y : data
eq f(a)=b
eq f(b)=a
end

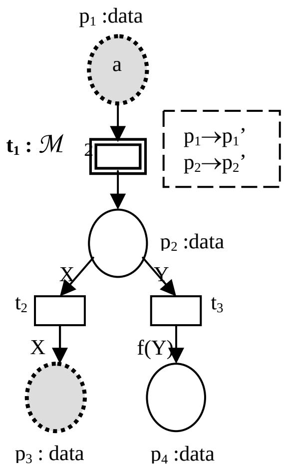


Figure 5: An example of CIRTA module $\mathcal{M}_1$.

Spec $\mathcal{M}_2$ **is**
sort data
op a :→data
op b :→data
op h(_): data→data
op g(_,_) :data data →data
vars X,Y : data
eq g(X,X)=h(X)
eq g(a, b)=b
eq g(b, a)=a
eq h(a)=b
eq h(b)=a
end

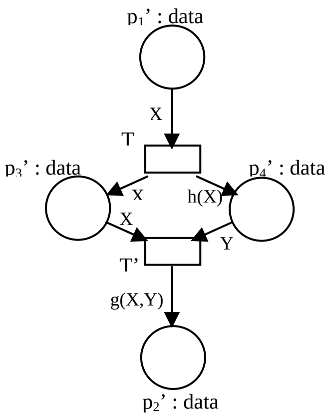


Figure 6: A sub-page of the CIRTA module $\mathcal{M}_1$.

Dynamic Semantics of a CIRTA module

The behavior of a CIRTA module is defined here by giving the behavioral equivalent ECATNet. Therefore composed-nodes must be substituted by the associated sub-pages. Each such substitution is a step refinement in the CIRTA specification. The following steps compose the merging process of the sub-page into the super-page:

- References of places and transitions used by the sub-page will be eventually changed in order to produce unique labels;
- One copy of each sub-page is inserted at the super-page; the composed-node is removed;
- The arcs connected with a composed-place are connected to the referred boundary place; for arcs connected with a composed-transition, void boundary places of the sub-page are merged with the associated places at the super-page (arcs and associated arc inscriptions in the sub-page are kept).
- The interface of the sub-page is composed by a set of boundary places or transitions. This set of nodes will constitute the glued points between the sub-page and the super-page, besides the common interface nodes.

Given the CIRTA module $\mathcal{M}_1$ of Figure.5; we can construct the equivalent ECATNet E_1 illustrated in **Figure7.**

Spec E_1 **is**
sort data
op a :→data
op b :→data
op f(_) :data→data
op h(_):data→data
op g(_,_) :data data →data
vars X,Y : data
eq f(a)=b
eq f(b)=a
eq g(X,X)=h(X)
eq g(a, b)=b
eq g(b, a)=a
eq h(a)=b
eq h(b)=a
end

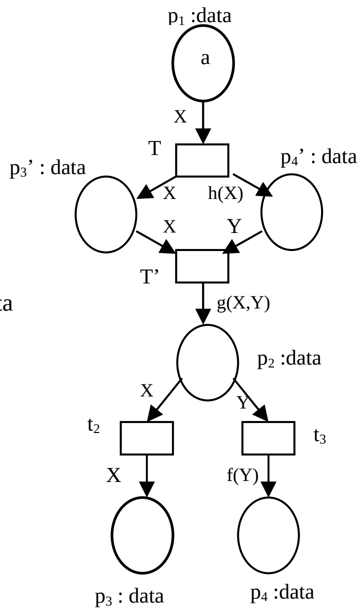


Figure 7: Equivalent ECATNet of the CIRTA module $\mathcal{M}_1$.

STRUCTURING OPERATIONS ON CIRTA MODULES

Importing modules

A CIRTA module can import an other CIRTA module in order to enrich the algebraic specification (with new sorts, operations, and axioms), and/or to extend the net (with new places, transitions, and arcs). The syntax of module importation is

< *importation*> **::= use** <*Mod*>

Where <*Mod*> is the identifier of the imported module and **use** is a CIRTA keyword.

For instance, let $\mathcal{M}_1$ be the CIRTA module of Figure 5, the module $\mathcal{M}_3$ in Figure 8 imports $\mathcal{M}_1$, adds a new operation *h(_,_)* and extends the net of $\mathcal{M}_1$ as follows.

Spec $\mathcal{M}_3$ **is**
  use $\mathcal{M}_1$
  op h(_,_) :data data→data
end

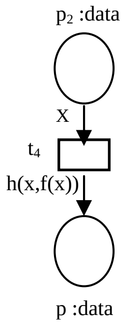


Figure 8: An example of module with use operation.

$\mathcal{M}_3$ is equivalent to module $\mathcal{M}_3'$ of figure 9.

Spec $\mathcal{M}_3'$ **is**
  use $\mathcal{M}_2$
  sort data
  op f(_) :data→data
  op h(_,_) :data data→data
  vars X,Y : data
  eq f(a)=b
  eq f(b)=a
end

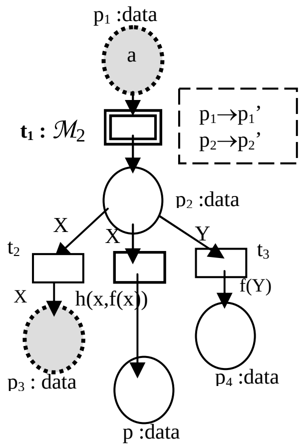


Figure 9: An equivalent CIRTA module to $\mathcal{M}_3$

Module composition

An important module building operator is *composition*. It has the syntax

<*composition*>:: = **make** <*Mod*> = <*Mod$_1$*> + <*Mod$_2$*> + …..+ <*Mod$_n$*> **endm**

This structuring operation creates a new module <*Mod*> that combines all the information in its summands (<*Mod$_1$*>, <*Mod$_2$*>, …..,<*Mod$_n$*>).

Formally, the composition of two modules $\mathcal{M}_1$ and $\mathcal{M}_2$ (where for i=1 to 2, $\mathcal{M}_i$=(Σpec$_i$, N$_i$,Pageset$_i$,π_i,δ_i) and N$_i$ =(P$_i$, T$_i$, τ_i, σ_i, Cap$_i$, IC$_i$, DT$_i$, CT$_i$, TC$_i$, M$_i$)) is a module $\mathcal{M}$ such that:

$\mathcal{M}$=(Σpec, N, Pageset, π, δ)   N =(P, T, τ, σ, Cap, IC, DT, CT, TC, M))   Σpec=(Σ,E)

- $\Sigma = \Sigma_1 \cup \Sigma_2$, $E=E_1 \cup E_2$, $P=P_1 \cup P_2$, $T=T_1 \cup T_2$, Pageset= Pageset$_1$∪Pageset$_2$.
- $\tau = \tau_1 \oplus \tau_3$, $\sigma = \sigma_1 \oplus \sigma_2$, $Cap = Cap_1 \oplus Cap_2$, $IC = IC_1 \oplus IC_2$, $DT = DT_1 \oplus DT_2$.
- $CT=CT_1 \oplus CT_2$, $M=M_1 \oplus M_2$, $\pi=\pi_1 \oplus \pi_2$, $\delta=\delta_1 \oplus \delta_2$, $TC(t)=TC_1(t) \wedge TC_2(t)$.

Notation: Let f_i: $A_i \rightarrow B_i$ be functions (i=1,..,n), we note $f_1 \oplus f_2 \oplus \ldots \oplus f_n$ the function defined by: $f_1 \oplus f_2 \oplus \ldots \oplus f_n$: $\cup_{i=1,n} A_i \rightarrow \cup_{i=1,n} B_i$, *where* $f_1 \oplus f_2 \oplus \ldots \oplus f_n (x)= f_i(x)$ *if* $x \in A_i$

Module renaming

The renaming of a module $\mathcal{M}$ allows to create a new module $\mathcal{M}'$ by changing the notations used in $\mathcal{M}$. The renaming operation uses a set of mappings (also called *renaming morphism*), namely a *sort mapping*, an *operator mapping*, a *place mapping* and *a transition mapping*. The syntax of renaming operation is:

<renaming> ::= **Make** <Mod'> = <Mod>***** <renaming morphism> **endm**

<renaming morphism>::=<sort mapping> <operator mapping >
<place mapping><tansition mapping>

<sort mapping> ::= {**so** <sort.identifier$_1$> **to** sort.indetifier$_2$>}

<operator mapping>::= {**op** <op.identifier$_1$> **to** <op.indetifier$_2$>}

<sort mapping> ::= {**pl** <place.identifier$_1$> **to** <place.indetifier$_2$>}

<sort mapping> ::= {**tr** <tans.identifier$_1$> **to** <tans.indetifier$_2$>}

For example, we rename the CIRTA module $\mathcal{M}_1$ to get the new module $\mathcal{M}_4$ as follows.

Make $\mathcal{M}_4$ = $\mathcal{M}_1$ *(**so** data **to** bool, **op** f (_) **to** not (_), **tr** t$_3$ **to** inverse) **endm**

The module designed by this specification is represented in figure 10.

Spec $\mathcal{M}_4$ **is**
  Use $\mathcal{M}_2$*(**so** data **to** bool)
  op not(_) :bool→bool
  vars X,Y : bool
  eq not(a)=b
  eq not(b)=a
end

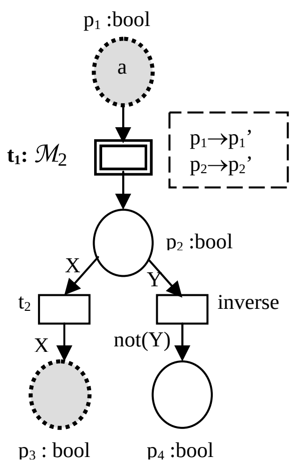


Figure 10: Example of renamed module.

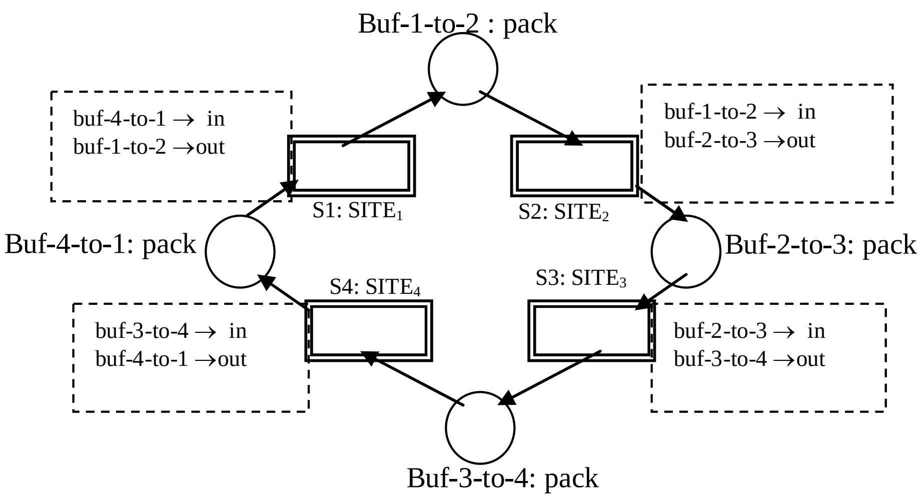
CASE STUDY

In the aim to illustrate the use of structuring mechanism presented in the above sections, we present an example consisting of a simple ring network with four different sites.

The system description will contain a set of eight CIRTA modules.

```
Spec  NETWORK is
  use Pack
  use  SITE1
  use  SITE2
  use  SITE3
  use  SITE4

end
```



- ***Module PACK***

PACK describes the packages which are sent in the network. Each package contains four fields: *n-field* for the package number, *se-field* for the identity of the sender, *re-field* for the identity of the receiver, and *d-field* for the data content of the package. In this module, we define only the data type by an algebraic specification which can be seen as a CIRTA module with an empty net.

```
Spec  PACK  is
    Use DATA  NAT SITE-ID
    Sort  pack
    op   <_,_,_,_>: nat site-id site-id data → pack
    op   re(_): pack →site-id
    op   se(_): pack →site-id
    op   da(_): pack→data
    Vars  n: nat   s, r: site-id    d:data
     eq    se(<n,s,r,d>)= s
     eq    re(<n,s,r,d>)= r
     eq    da(<n,s,r,d>)= d

 end
```

The modules DATA NAT and SITE-ID used above are respectively the algebraic specifications of *data*, *natural numbers* and *site identifiers*. The algebraic specifications DATA and NAT are trivial. They are omitted here since the main principle of the ring network can be understood.

- ***Module NETWORK***

The figure bellow shows a CIRTA module *NETWORK* which has four places and four *composed transitions* positioned in a ring.

- ***Module SITE-ID***

This module contains the elements id_1, id_2, id_3 and id_4 which are used to identify the individual sites of the network.

```
Spec SITE-ID  is
   Sort site-id
   Ops   id1:→ site-id
         id2 :→ site-id
         id3 :→ site-id
         id4 :→ site-id
end
```

- ***Module SITE***

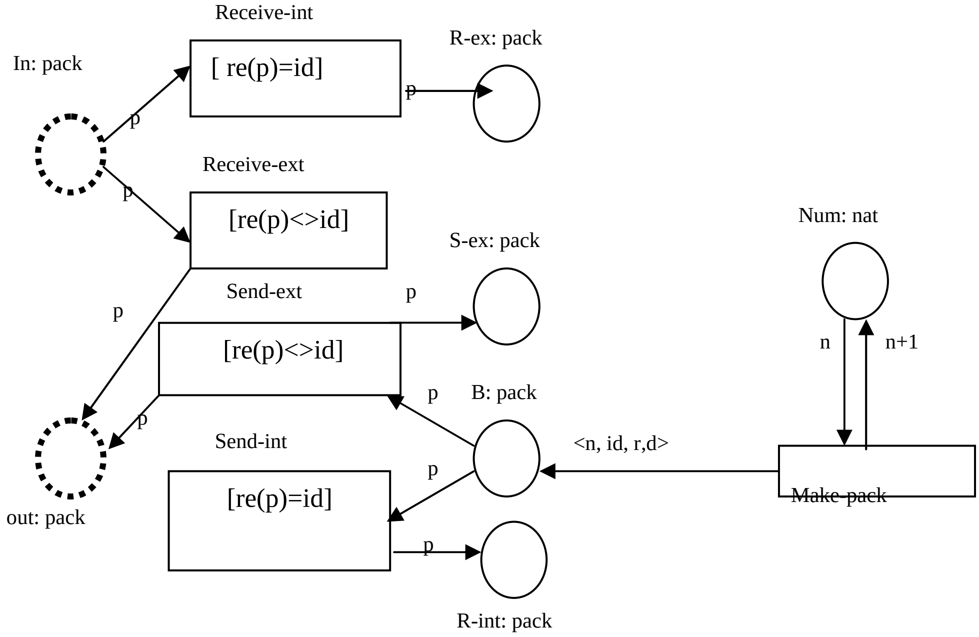
In this example we can see that all four composed transitions of NETWORK share the same following CIRTA module SITE.

The SITE module has three different transitions. Each occurrence of *Make-pack* creates a new package. The *n-field* of the new package is determined by the token in the place *Num*. The *se-field* of the new package is the site identifier of the site. Finally the *re-field* and *d-field* are determined by variables *r* (which can take an arbitrary value from SITE-IDENT) and *d*.

The created packages are handled by transition *Send*, which inspects the *re-field* of the package. This is done by means of the *re(p)* which denotes the *re-field* of *p*. When *re-field* indicates that the receiver is different from the present site, the package is transferred to the place *out* and a copy of the package is put in the place *S-ex* (indicating that the package is sent to an external receiver).

Otherwise the package is sent directly to the place *R-int* (indicating that the package is received from an internal sender).

Spec SITE **is**
use PACK
op id:→ site-id
end



- ***Modules SITE$_1$, SITE$_2$, SITE$_3$ and SITE$_4$***

Each site has the same behavior as SITE. The main difference concerns the identifier *id* of the site. Hence each module SITE$_i$ (i=1,..,4) can be obtained by renaming the module SITE as follows.

The modular specification of the network given above is behaviorally equivalent to the ECATNet presented in Figuere 11:

Make SITE$_1$= SITE*(**op** id **to** id$_1$) **endm**.
Make SITE$_2$= SITE*(**op** id **to** id$_2$) **endm**.
Make SITE$_3$= SITE*(**op** id **to** id$_3$) **endm**.
Make SITE$_4$= SITE*(**op** id **to** id$_4$) **endm**.

Spec NETWORK **is**

use DATA NAT
Sort pack site-id
op <_,_,_,_>: nat site-id site-id data $\rightarrow$ pack
op re(_): pack $\rightarrow$site-id
op se(_): pack $\rightarrow$site-id
op da(_): pack$\rightarrow$data
op id$_1$:$\rightarrow$ site-id
op id$_2$:$\rightarrow$ site-id
op id$_3$:$\rightarrow$ site-id
op id$_4$:$\rightarrow$ site-id

Vars n: nat s, r: site-id d:data
eq se(<n,s,r,d>)= s
eq re(<n,s,r,d>)= r
eq da(<n,s,r,d>)= d

end

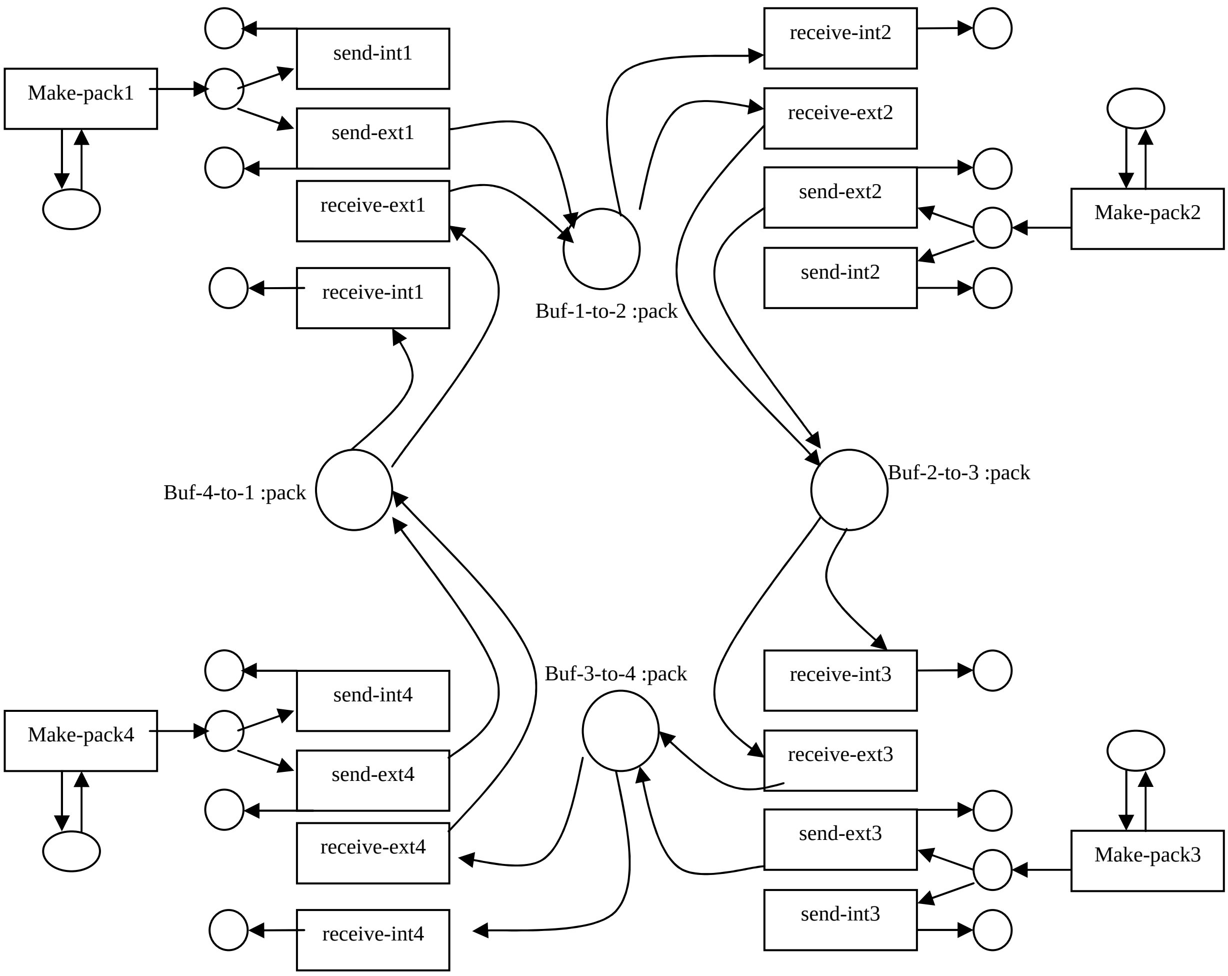


Figure 11: The equivalent ECATNet of the module NETWORK

VERIFICATION OF CIRTA SPECIFICATION

For the levels of complexity typical to concurrent systems, traditional validation techniques like simulation and testing are neither sufficient nor viable to verify their correctness.

Formal methods are becoming an alternative to ensure the correctness of designs. In this section we present a systematic procedure to translate modular CIRTA specifications into rewriting theories.

The advantages of using rewriting logic as a semantic framework for concurrency models has been amply demonstrated in [13][14]. Essentially, rewriting logic has a simple formalism, with only few rules of deduction. It supports user-definable logical connectives, which can be chosen to fit the problem at hand. Besides, it is intrinsically concurrent; and it is realizable in a wide spectrum of logical languages [9][7][15] supporting executable specifications.
To verify the correctness of a concurrent system (specified using CIRTA language) we execute the following steps:

- _Step 1_: We translate a CIRTA specification into an ECATNet as stated in definition Section 3.3 and Section.4.
- _Step 2_: We generate a rewrite theory as detailed in [5][11]. Hence, the effect of transitions firing is expressed by rewrite rules which depend strongly on the form of the Input Conditions (*IC*), and Destroyed Tokens (*DT*).
- _Step 3_: We use existing analysis tools [9][7][15] to check properties expressed as a rewriting logic formulas.

It should be noted that all the translation steps can be done automatically so that the designer is not concerned with this translation.

CONCLUSION

Our investigation has shown the advantages of using CIRTA; an ECATNets based language, for complex concurrent systems specification. CIRTA allows capturing relevant information characteristics of such systems. In our approach it is feasible to specify large systems as a set of comprehensible structured modules and, at the same time, the essential characteristics of the system may be captured by the model. Moreover we have also presented an example of a practical system specification, namely a ring in order to illustrate the modeling capabilities of CIRTA. To make easy the formal analysis step of concurrent systems specified using CIRTA, we have proposed an approach to translate CIRTA specifications into rewriting theories. Hence, we take advantages of practical tools developed in the rewriting logic framework. In future, we will use CIRTA to develop a formal approach to verification and transformation based synthesis of concurrent systems.

REFERENCES

[1]- L. Lavagro, A. Sangiovanni-Vicentilli, and E. Sentovich, "Models of Computation for Embedded System Design", in *System-Level Synthesis*, A.A. Jerraya and J. Mermet, ed. Dordecht: Kluwer,1999, pp.45-102.

[2] P.Maciel, E. Barros, and W. Rosenstiel, " A Petri net Model for Hardware/Software codesign", in *Design Automation for Embedded Systems*, Vol.**4**, pp.243-310, Oct.1999.

[3] M.Varea, and B. Al-Hashimi, "Dual Transitions Petri Net Based Modelling Technique for Embedded Systems Specification", in *Proc. DATE Conference*, 2001, pp.566-571.

[4] M.Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, "Synthesis of Embedded Software Using Free-Choice Petri Nets", in *Proc. DAC*, 1999, pp. 805-810.

[5] M. Bettaz, M. Maouche "Modelling of Object Based Systems with hidden sorted ECATNets". *MASCOTS' 95, Durham, North-Carolina, IEEE*, 1995, pp.307-311.

[6] M. Bettaz, M. Maouche, M. Soualmi, and M. Boukebeche, "Compact modelling and rapid prototyping of communication software with ECATNets: a case study". *Simulation Series* Vol.**25**, N°1, SCS 1993, pp149-154.

[7] P. Borovansky, C.Kirchner, H. Kirchner, P6E. Moreau, and M. Vittek. "ELAN: A logical framework based on computational systems". *Proc. first Intl. Workshop on Rewriting Logic and its applications*, Vol. 4 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1996.

[8] M. Bettaz, G. Reggio "A SMoLCS Based Kit for Defining the Semantics of High Level Algebraic Petri Nets", *LNCS 785*, pp.98-112, Springer-Verlag, 1994.

[9] M. Clavel, F. Duran, S. Eker, J. Meseguer, and M.-O. Stehr. "Maude as a Formal meta-tool". In J. M. Wing, J.Woodcock, and J. Davies, editors, *Proc. FM'99, LNCS 1709* , pp 1684-1703. Springer,1999.

[10] K. Djemame, D.G. Gilles, L.M. Mackenzie, M. Bettaz, "Performance comparison of high-level algebraic nets distributed simulation protocols". in *Journal of systems architecture* **44** (1998) pp.457-472.

[11] F. Belala "Un cadre Formel pour l'Analyse des ECATNets" Thèse Dept. Informatique, Univ. Constantine, 2002.

[12] E. Pelz, and H. Fleishhack, "Compositional high-level Petri nets with timing constraints". *Third international Conference on application of concurrency to system design ACSD' 03*, June 2003, Guimaraes, Portugal.

[13] J.Meseguer. "Rewriting Logic as a Semantic Framework for Concurrency". In U. Montanari and V. Sassone, editors, *Proc. Concur'96, Volume **1119** of LNCS*, pp.331-372, Springer, 1996.

[14] J.Meseguer. "Resarch Directions in Rewriting Logic". In U. Berger H. Schwichtenberg, editors, *NATO ASI Series F: Computer and Systems Sciences **165***, pp 347-398. Springer,1999.

[15] K.Futatsugi and R. Diaconescu. "CafeOBJ report". *AMAST Series*, World Scientific, 1998.

[16] H. Fleishhack, and E. Pelz, "Hierarchical timed high level nets and their branching processes", *in ICATPN'03, LNCS 2679*, pp 397-416, Spinger, 2003.

[17] N. Zeghib "ASTRE: un environnement de spécification algébrique". in *Proc. of 2nd MCAISE*, Tunis 13-16/04/1992 pp.2-16.

[18] N. Zeghib, and M. Maouche "Proposition d'une approche pour la paramétrisation des ECATNets: application à un système de production". in *Proc. of MOSIM'97* Rouen France 5-6/06/1997 pp.425-433.

[19] H.J. Genrich. "Predicates / transition nets". In *High-Level Petri Nets: Theory and Practice*, page 3-43. Springer-Verlag, 1991.

[20] I. Suzuki, and T. Murata, "A Method for Stepwise Refinement and Abstraction of Petri Nets", *in Journal of Computer and System Sciences*, Vol.**27**, pp. 51-76,1983.

[21] P.C. Olveczky, and J. Meseguer, "Specification of Real-time and Hybrid Systems in Rewriting Logic", in *Theoretical Computer Science*, Vol.**285**, issue August 2002, pp. 359-405.

[22] R. Alur, C. Courcoubetis and D.L. Dill, " Model Checking for Real-Time systems", in *Proc. Symposium on Logic in Computer Science*, 1999, pp. 414-425.

[23] N.Zeghib and M. Bettaz , " On Synchronous and Asynchronous Communication in Modular High-level Nets: The case of ECATNets" in *Proc. ICTTA, International Conference on Information and Communication technologies: from Theory to Application, Damascus, 2004*.

[24] N.Zeghib, K.Barkaoui and M. Bettaz, " CIRTA : An ECATNets Based Model for Embedded Systems Specification", *Proceeding of the 2005 International Conference on Embedded Systems and Applications ESA'05, Las Vegas, Nevada, June 27-30, 2005*

[25] N.Zeghib, Barkaoui and M. Bettaz " Contextual ECATNets semantics in Conditionnal rewriting logic ", *Proceeding of The 4th ACS/IEEE International Conference on Computer Systems and Applications Mach 8-11, 2006, AICCSA'2006, Dubai/Sharjah, UAE.*