HIERARCHICAL ALGEBRAIC NETS

Nicolas Guelfi

Software Engineering Laboratory
Swiss Federal Institute of Technology
CH-1015 Lausanne
Switzerland

phone : +41 21.693.67.68
fax : +41 21.693.50.79
email : guelfi@di.epfl.ch

ABSTRACT*

This paper presents a specifications formalism, Hierarchical Algebraic Nets (HAN), which allows the specification and design of heterogeneous concurrent distributed systems using hierarchical descriptions based on algebraic specifications and Petri nets. A simple example is firstly given in order to present the graphical representation associated to HAN, then the abstract syntax and formal semantics are described. The interesting aspects of HAN for reactive distributed systems development are listed and the link between the HAN model to the SANDS development environment is summarised.

1. INTRODUCTION

The interest of the mixed use of algebraic specifications and Petri nets (introducing a new basic class called algebraic nets [1]) has been shown in many papers since 1985. Several models have been defined and are now focused on the structuring primitives used. The principle of hierarchical construction of the specification has already been used in some Petri net models, for example hierarchical coloured nets [9], but, concerning algebraic nets, only the CO-OPN language (Concurrent Object-Oriented Petri Nets, introduced in 1991 by Didier Buchs and Nicolas Guelfi [2] has tried to introduce hierarchical construction primitives for algebraic nets. HAN proposed in this paper provides a formal model which can be used also to give a precise semantics to the CO-OPN language. The complete presentation of HAN can be found in [6].

Hierarchical Algebraic Nets (called HAN) have a graphic syntax which corresponds to an abstract syntax and they have a complete formal semantics which is given in order to determine a mathematical object for each HAN specification. HAN is an hybrid specification formalism since it uses two existing specification formalisms: hierarchical algebraic specifications (HAS), a property oriented specification formalism for the description of data structures, and Petri nets, a model based specification formalism for the description of concurrent behaviours.

* *This work has been sponsored partially by the Esprit Long Term Research Project 20072 ``Design for Validation'' (DeVa) with the financial support of the OFES (Office Fédéral de l'Education et de la Science), and by the Swiss National Science Foundation project ``Formal Methods for Concurrency''.•*

The syntax of HAN allows to define a hierarchy of algebraic net modules (ANM) based on two construction primitives: the union of two HAN and the enrichment of an HAN by an ANM. The enrichment operator is the one who builds the hierarchy. Each transition of a hierarchical level *n* can ask, through the notion of process, to the algebraic net modules of lower level to execute specific transition in an order specified using three operators: `sim` for simultaneity, `seq` for sequence and `alt` for choice.

The semantics of HAN is given in terms of distributed transition system (DTS). That is, a set of transitions each one indicating a state change of the whole hierarchical system and a multi-set of events indicating atomic events executed in parallel by the system. The distributed transition system is built in a way which follows the hierarchical construction of the specification. This semantics expresses all the concurrency of the system specified.

The paper is organised as follows: it starts with a simple presentation of a HAN using a small example also used to show the graphical representation associated to HAN, it presents the basic formalisms used in HAN, then it describes the syntax and semantics of HAN, and ends by giving interesting features of HAN for the formal specification, design and development of distributed systems.

2. EXAMPLE

The figure 1 gives a simple example of a system specification with a hierarchical algebraic net. It has two levels. The level 2 specify an ANM (`anm21`) with two parametered transitions (called methods and represented by black rectangles) `put(p)` and `get(p)` where `p` is a variable of type `product` which must be specified in a hierarchical algebraic specification associated to this ANM. Once the parametered transition `put(p)` is fired (maybe several times simultaneously) it puts a couple `<p,0>` made of the product and its price (which is not yet determined) in the place `stock` which is a multiset of value of type `cp2` (cartesian product of products and prices specified in the HAS) indicated by `stock:cp2`. Then , for each new product (with price equal to `0`) an internal transition `prepare` (white rectangle) is immediately executed. This internal transition has a process attach to it:
`sim(price(p,v), seq(in(p),out(p')))`. This process concerns parametered transitions of level lower than 2 in the hierarchy (here the transitions in the process belongs to

two ANM of level 1, `anm11` and `anm12`). The process asks to these ANM to execute in parallel the computation of the price of the product (`price(p,v)` in `anm11`), and its preparation in `anm12` through a treatment asked to, lets say, a production cell (put the product in the cell by `in(p)` and ask then in the next sequence step to return the packaged product by `out(p)`. The overall execution of this process made at level 1 is completely abstracted by the transition `prepare` (as well as all the other process which could have been refined in the ANM of level 1, if there were some).

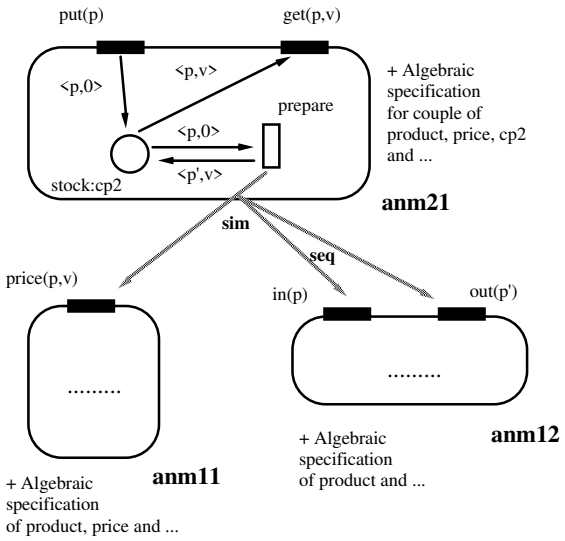

Figure 1 : A Hierarchical Algebraic Net

3. BASICS

3.1 Hierarchical Algebraic Specifications

Algebraic specifications are a property oriented formalism based on logic and used for the specification of data structures (called sorts or types). A presentation module (PM) is usually defined in two parts : the signature and the axioms : (1) the signature is made of a set *'S'* of sorts names (for example boolean, integers, stacks, ...); a set *'O'* of operation names made of a generator names set *'G'* which build values of the sort and a function names set *'F'*. These generators and functions names are typed by a function τ which indicates for each operation name the number of parameters with their domain names (i.e. the sort names) and the sort name of the resulting value (i.e. the codomain name of the operation); (2) the properties of these operations are expressed in the axioms which are logical formulas often restricted to positive conditional axioms of the form $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t_{n+1} = t'_{n+1}$ where t_i, t'_i are free terms built with the operators of the signature and with a set *'V'* of variables typed on *S*. Hierarchical Algebraic Specifications (HAS) offer two structuring primitives which are the union of two HAS ($\coprod$) and the enrichment of a HAS by a presentation module (+). In order to avoid the construction of problematic HASs, we use the notion of well structured HAS which puts three syntactic constraints : (1) if two sub-specifications of a HAS are enriched by the same PM then they must be equal; (2) all the PMs of a HAS must be disjoint; (3) the generators of a sort and the sort itself must be defined in the same module. The semantics of a PM is given in terms of a set of 'Σ -algebras'. A Σ -algebra *A* for a signature $\Sigma =< S, G, F >$ is a set of domains A_s for each sort name*s* of *S* and a set of total functions 'f_A' for all function name of $G \cup F$ such that if the signature of f_A is $f_A : s_1 \ldots s_n \rightarrow s_{n+1}$ then f_A must be from $A_{s_1} \times \ldots \times A_{s_n}$ *to* $A_{s_{n+1}}$. A Σ -algebra is a model of a PM $< \Sigma, V, Ax >$ if and only if all the axioms of *Ax* are valid in *A*. That is, the functions f_A have really the properties expressed in *Ax* (perhaps more, but not less !). Concerning the HAS semantics, it is defined using the notion of hierarchical model (HM). A HM is a finitely generated[1] Σ -algebra *A* model of the PM associated to the HAS *'Spec'* (i.e. its flattening) with some structuring constraints : (1) if the HAS *Spec* is empty, then *A* must be a special Σ -algebra written $\mathbb{I}$; (2) if *Spec* is the enrichment of the empty HAS by a PM $\Delta Pres$ then *A* must be the initial model[2] of $\Delta Pres$; (3) if *Spec* is the union of two HASs *Spec'* and *Spec"* then *A* restricted to *Spec'* (resp. *Spec"*) must be a HM of *Spec'* (resp. *Spec"*); (4) if *Spec* is the enrichment of a HAS *Spec'* by a presentation module $\Delta Pres$ then *A* restricted to *Spec'* must be a HM of *Spec'* .

A general and complete presentation of algebraic specifications can be found in [3] and references including hierarchical algebraic specifications are given in [4].

3.2 Algebraic Nets

We have chosen to define the algebraic net modules, which are the basic component of hierarchical algebraic nets, as a mixing of the approaches of J. vautherin [5] and W. Reisig [1]. In algebraic nets a key point concerns the places definition. A place is a multiset of typed values : (1) J. Vautherin considers the multisets at a semantic level by enhancing the AS models to multisets; (2) W.Reisig considers the multisets at the syntactic level by adding an algebraic specification part for the multisets for each sort used to type a place. Then, the algebraic specification semantics is used to give a semantics to the multisets. Both approaches of Vautherin and Reisig were considering non hierarchical algebraic specifications.

We consider that the syntactic approach is interesting for the places definition. It can be a way of being more abstract on the place definition if we take care of the semantics consequences. In our case, we have chosen to make a syntactical extension of the HAS by introducing, for each sort *s* which types a place *p* of an ANM, a sort *[s]* for the type 'multiset of values of sort s'. As we have a notion of hierarchical model which forces to have an initial approach for the models of the basic PM , we specify the sort 'bool' in a basic PM thus forcing the values true and false to be different. Then, as we use the equality operators on the booleans for the definition of the multiset sort, we avoid the sets as semantics of the multisets.

If *Spec* is the HAS associated to an ANM, then *[Spec]* will indicate its syntactic extension to the multisets which for each sort s associated to a place, adds a sort *[s]* and the operations $\varnothing_{[s]} : \rightarrow [s]$ (the empty multiset), $[_]_{[s]} : s \rightarrow [s]$ (the coercion), $_ +_{[s]} _ : [s], [s] \rightarrow [s]$ (the union) and other classical operations such as inclusion , cardinality, equality ...

---

[1] *all the values of each domain must be reachable using the generators only.*

[2] *the model which satifies the smallest set of properties (only the one asked in the axioms).*

4. SYNTAX

4.1 Basic concepts

We consider as given the subsets of a universe *U* for the sets of all place names, transition names, method names, the sets of all HAS, all algebraic values, all algebraic terms with or without variables. Generic variable names representing subsets or elements of these sets are used: *P, p* for places, *T*, *t* for transitions, *M, m* for methods, *Spec* for a HAS, *A* for an algebra, *a* for an algebraic value, *t* for algebraic term.

Interfaces An interface is the visible part of a HAN and defines a set of methods with their signature. It indicates the parametered transitions offered by a HAN.

Definition **:** An interface is a tuple made of : *M* a set of method names, S_M a set of sort names and τ_M an application such that $\tau_M : M \rightarrow S_M^*$. We write $mt : s_1,\ldots,s_n$ if $\tau_M(mt) = s_1 \ldots s_n$.

Atomic event Atomic events are those offered to the next hierarchical level in a HAN. As each HAN has a corresponding interface (made of all the interfaces of its ANMs), all its methods can be 'offered to' (i.e. called by) a new higher ANM specification component.

Definition **:** An atomic event is a well formed term $mt(t_1,\ldots,t_n)$ which can be built over a an interface and such that t_i is a typed term with variables built over a HAS.

Processes An algebraic net module can be refined during execution into an expression built on other lower ANMs. We call a process such an expression which is built from atomic events and composition operators. These operators are the simultaneous (sim), the sequential (seq) and the alternative (alt) operators. These processes can be associated to all kind of transition and are part of their execution.

Definition **:** A process is either the empty process $\varnothing$, an atomic event $mt(t_1,\ldots,t_n)$, or a composed process $sim(pr, pr')$, $seq(pr, pr')$ and $alt(pr, pr')$ where pr, pr' are processes.

Example In the example given in section 2, (a) the interfaces are {`put: product, get:product,price`} for anm21, {price:product , price} for anm1, and, {in, out: product} for anm12; (b) `put(p), price(p,v), in(plate)` are atomic events if plate is defined as a product in the HAS; (c) `sim(price(p,v),seq(in(p),out(p'))))` is a process.

4.2 Algebraic net module

We introduce in this part the algebraic net modules as algebraic nets to which we add the notion of interface and behavioural axiom. The transitions of an ANM are internal or parametered transitions (methods) with behaviours specified using behavioral axioms (DBA) which contains all the execution constraints. A process can be one of these constraints, It reflects the hierarchical relation between the transition which contains the process and the nets which contains transitions included in the process. The signification of this association will be the simultaneous execution of the transition and its attached process, but the only visible part of this execution will be the transition part and the process will be abstracted by the transition execution.

Definition **:** An algebraic net module is a tuple $anm = < Spec, P, T, M, X, \tau_P, \tau_X, \tau_M, BA >$ made of :

a) a hierarchical algebraic specification *Spec* ,
b) a set *P* of places names,
c) a set *T* of transitions names,
d) $< M, S_M, \tau_M >$ an interface such defined on *Spec,*
e) a set X of variables names,
f) τ_P, τ_X s.t. $\tau_P : P \rightarrow S$ and $\tau_X : X \rightarrow S$
g) a set *BA* of behavioural axioms over *Spec* , P, T, *M* and X.

Definition **:** A behavioural axiom is a tuple $< ev, Cond, In, Out, pr >$ such that :

a) *ev* is an atomic event or transition name of *T*,
b) *Cond* is either a set of pairs *(t,t')* where *t* and *t'* are same sorts algebraic terms with variables, either $\varnothing$. It is a boolean condition associated to the firing of *ev*.
c) *In* is a P-set $(In_p)_{p \in P}$ of terms of type multiset. It indicates the multiset of values taken by *ev* in each place *p*.
c) *Out* is a P-set $(Out_p)_{p \in P}$ of terms of type multiset. It indicates the multiset of values put by *ev* in each place *p*.
e) *pr* is a process.

Example In the example given in section 2, the `anm21` is defined by: (a) *Spec* is the HAS specifying the sorts `product, price and cp2`, (b) *P* is {`Stock: cp2`}, (c) *T* is {`prepare`}, (d) *M* is {`put:product, get:product, price`}, (e) *X* is {`p,p':product,v:price`}, (f) τ_P, τ_X are given with *P* and *X*, (g) the behavioural axiom of prepare is :

```
<prepare,∅,In(stock)=[<p,0>],out(stock)=[
<p',v>],sim(price(p,v),seq(in(p),out(p')))
>.
```

4.3 Algebraic net modules compatibility

Concerning hierarchical algebraic specification As hierarchical algebraic specifications are associated to algebraic net modules in hierarchical algebraic nets, we need to determine if two HAS are compatible before allowing the communication of typed values between modules. *Spec* and *Spec'* are compatible if and only if $Spec \coprod Spec'$ is a well structured hierarchical algebraic specification. The compatibility of two HASs insure that there common sorts are syntactically specified in the same manner. We have the similar notion for Σ-algebras which says that two Σ-algebras *A* and *A'*, respectively models of *Spec* and *Spec'*, are compatible on there common signature if and only if their restriction on this signature are equal. This ensures that we gave the same implementation to the common sorts. The union of two compatible models *A* and *A'* is defined naturally since no problem is encountered due to the compatibility of the Σ-algebras. If *A* and *A'* are compatible then their union is a model of $Spec \coprod Spec'$.

Concerning hierarchical algebraic net modules Two algebraic net modules are compatible if and only if their associated HAS are compatible for the sorts used in processes and if they use disjoint names for their net components.

Example In the example given in section 2 all the HAS must be compatible on the sorts `product` and only `anm11` and `anm12` for the `price`, in order to have compatible ANMs.

4.4 Hierarchical Algebraic Nets

This part introduces the structuring primitives over algebraic nets modules in order to obtain hierarchical algebraic nets. We have chosen a similar approach than the one chosen in hierarchical algebraic specifications.

Definition **:** A hierarchical algebraic net is :
(a) either the empty basic algebraic net, written $\emptyset$,
(b) either an enrichment of a hierarchical algebraic net han' by an algebraic net module *anm*, written : $han' + anm$,
(c) either a union of two hierarchical algebraic nets han' and han'', written : $han' \amalg han''$.

Remarks: (1) The hierarchical operator is the enrichment, thus a level can be associated to every ANM in the hierarchy; (2) A HAN is said well structured if an event of an ANM of level *n* include in its process only events defined in level lower than *n*; and if all the ANMs of the HAN are compatible.

Example In the example given in section 2, the HAN is defined by : anm21 + ((anm11 + $\emptyset$) $\amalg$ (anm12 + $\emptyset$)).

5. SEMANTICS

5.1 Basic concepts

The semantic concepts introduced here focus, for a HAN, on the semantic interpretation of data structures, state variables (the places) and events (internal tranistions and methods).

Models Distribution A models distribution is a partial function A_r which associates to each algebraic specification of all the ANMs of a HAN an algebraic model.

Markings A marking is a partial application *m* from places to values associating an algebraic value of type multiset of *s* to a place of sort *s*. Two markings *m* and *m'* are compatible if their algebraic value domains are compatible for all their common places.

Distributed Markings are introduced in order to handle the notion of HANs states. A distributed marking is defined for a models distribution and must 'respect' it. That is, its restriction to each ANM of the HAN must be a marking of this ANM over its associated algebraic specification. Furthermore, it is important to notice that a distributed marking is defined as a classical marking which have some constraints on the domain for the algebraic values and which must mark only places which are defined in some modules of the HAN. A distributed marking consider all the places of the HAN at the same level independently of their hierarchical level due to the ANMs.

Semantic processes are used to label the transitions of the distributed transition systems. They are built as the syntactic process but the methods parameters are Σ-algebra values corresponding the algebraic terms' interpretations. Thus, the semantic processes are defined for a models distribution.

5.2 Distributed Transitions System

A distributed transition system (DTS) *dts* is a set of triplets made of an initial distributed marking dm_i, two semantic processes pr_e et pr_i and a final distributed marking dm_f. It indicates that a hierarchical algebraic net in the initial state dm_i executes explicitly the process pr_e and needs implicitly the simultaneous execution of pr_i to reach the final state dm_f.

A distributed transitions system restriction is defined in order to restrict the *dts* relation to a set of specific places. It is useful to extract the subpart which concerns a sub-HAN and is written $dts_{|P}$.

Distributed Transitions System Stabilisation treats the internal transitions automatic execution. We consider internal transitions are reactive transitions which execute nothing 'visible' and which must be always automatically executed when possible. The stabilisation merges an internal transition (with empty explicit and implicit processes) with the preceding one having a non empty explicit process and an empty implicit process (ex: $dm \xrightarrow{<pr,\emptyset'>} dm'$ and $dm' \xrightarrow{<\emptyset,\emptyset>} dm''$ adds a new transition $dm \xrightarrow{<pr,\emptyset'>} dm''$).

Synchronised Closure of Distributed Transitions System defines the set of 'deductible behaviours' from the transitions of a DTS. It is done in two steps associated to the explicit and implicit parts of the transition labels.

The first step is called behavioural closure and builds all the possible combinations of distributed transitions using the synchronisation operators *'sim'*, *'seq'* and *'alt'*. If $dm_1 \xrightarrow{<pr_1,\emptyset>} dm'_1$ and $dm_2 \xrightarrow{<pr_2,\emptyset>} dm'_2$ are two distributed transitions then: (a) $dm_3 \xrightarrow{<sim(pr_1,pr_2),\emptyset>} dm'_3$ is added such that $dm_3 = dm_1 + dm_2$ and $dm'_3 = dm'_1 + dm'_2$; (b) $dm_3 \xrightarrow{<seq(pr_1,pr_2),\emptyset>} dm'_3$ is added if dm'_1 and dm_2 have the same marking for their common places, and if dm_3 is dm_1 plus the marking of places marked only by dm_2 and not dm'_1, and, dm'_3 is dm'_2 plus the marking of places marked only by dm'_1 and not dm'_2; and, (c) $dm_3 \xrightarrow{<alt(pr_1,pr),\emptyset>} dm'_3$ is added for all semantic process *pr*. For example, the behavioural closure with *seq* of $dm_1 \xrightarrow{<g(4),\emptyset>} dm'_1$ and $dm'_1 \xrightarrow{<h(3)),\emptyset>} dm''_1$ adds a new transition $dm_1 \xrightarrow{<seq(g(4),h(3)),\emptyset>} dm''_1$.

The second step is a synchronisation step. To understand its meaning we must recall that if $dm \xrightarrow{<pr,pr'>} dm'$ is a transition of *dts*, then it informally means that *pr* is executed by the system and makes the system state evolve from *dm* to *dm'*, but, the system has executed only the visible part of the behaviour and has not executed the process *pr'*. So, the synchronisation step has to deduce new behaviours by synchronising transitions who needs an implicit process execution with the one who have this asked process as explicit process and no implicit process. If $dm_1 \xrightarrow{<pr_1,pr'_1>} dm'_1$ and $dm_2 \xrightarrow{<pr'_1,\emptyset>} dm'_2$ are two distributed transitions then $dm_3 \xrightarrow{<pr_1,\emptyset>} dm'_3$ is added such that $dm_3 = dm_1 + dm_2$ and $dm'_3 = dm'_1 + dm'_2$. For example, the synchronisation of : $dm_1 \xrightarrow{<f(2),seq(g(4),h(3))>} dm'_1$ and $dm_2 \xrightarrow{<seq(g(4),h(3)),\emptyset>} dm'_2$ adds a new transition $dm_1 + dm_2 \xrightarrow{<f(2),\emptyset>} dm'_1 + dm'_2$).

The Synchronised Closure of *dts* is the incremental application of behavioral closure and synchronisation steps on *dts* until a fixed point is reached (it is written *Clot(dts)*).

5.3 Algebraic Net Modules Semantics

The algebraic net modules semantics is given in terms of distributed transitions systems. We first present the definition of elementary model of a HAN for a given algebraic model of its associated algebraic specification.

Definition : The distributed transition system *dts* is an elementary model of an algebraic net module *anm* for an algebra *A* semantics *of the* HAS of anm if and only if each marking of *dts* concerns only the places of *anm* and if for all $dm \xrightarrow{<pr_1, pr'_1>} dm'$ of *dts* there is a behavioral axiom $ba = < ev, Cond, In, Out, pr >$ of *anm* and a substitution *sub* for the variables of *ba* such that the following properties are satisfied :
(0) pr_1 is a semantic process obtained by interpreting *ev* in *A* for the substitution *sub;*
(1) *dm* associates to each input place of *ev* all the resources indicated by the *In* term interpreted in *A* for the substitution *sub*;
(2) all the equations of *Cond* are satisfied *in A* with *sub;*
(3) *dm'* is obtained by subtracting, for each place of *anm*, the value indicated by *In* and adding the value indicated by *Out* (values which are interpretation of each input or output term in *A* for the substitution *sub)*;
(4) pr'_1 is a semantic process obtained by interpreting *pr* in *A* for the substitution *sub;*

An elementary model is not always stable and contains only transitions whose explicit process and not composed process.

*Definition***:** If *dts* is an elementary model of *anm* for an algebra *A* then the synchronised closure of its stabilisation is the partial model of *anm*.

Remark: The partial model of *anm* for a marking dm_0 of *anm* on *A*, is the sub-dts of the partial model of *anm* such that all its markings are accessible from dm_0.

The last notion is the one of complete model as the "largest" partial model. That is, a complete model includes all the possible behaviours executable from a given state (initial or not). It corresponds to the union of all the partial models and is the usual model given for algebraic nets.

5.4 Hierarchical Algebraic Net Semantics

We define below the notions of partial and complete models of hierarchical algebraic nets or of marked hierarchical algebraic nets for a given models distribution A_r .

*Definition***:** Let *han*, A_r over *han* and *dts*, we define the property of partial model for *dts* by structural induction in the following way :
1) if $han = \varnothing$ then *dts* is a partial model of *han* iff $dts = \varnothing$.
2) if $(han = han' \coprod han'')$ then *dts* is a partial model of *han* if $dts_{|han'}$ is a partial model of *han'* for $A_{r\,|han'}$, $dts_{|han''}$ is a partial model of *han"* for $A_{r\,|han''}$, and if *dts* is included in $Clot(dts_{|han'} \cup dts_{|han''})$.
3) if $(han = han' + anm)$ then *dts* is a partial model of *han* if $dts_{|anm}$ is a partial model of *anm* $A_{r\,|man}$, $dts_{|han'}$ is a partial model of *han'* for $A_{r\,|han'}$ and if *dts* is included in $Clot(dts_{|han'} \cup dts_{|anm})$.

Similarly to ANM models, a DTS is a partial model of a marked HAN if all its distributed markings are reachable from distributed submarking of the given marking. The idea is that all distributed transition of the DTS concerns a subset of the modules of the HAN, so it must be possible to determine some initial states from which all marking of *dts* can be reached.

Example For the example given in section 2, the *dts1* and *dts2* will be easy computed models for anm1 and anm2 since they are at the lowest hierarchical level (no process calls). Then the closure will compute all the possible behaviours offered by these two ANM. Especially, the processes for `pr_ex= sim(price(p,v),seq(in(p),out(p')))` with specific values for p,v and p' depending on the initial markings and on the model distribution for the semantics of data structures. A synchronisation will then be possible between a transition with `pr_ex` as explicit process and the transition with `prepare` as explicit process and `pr_ex` as implicit process. It will give an empty distributed transition which will be stabilised to the previous transition (normally labelled by a `put`).

6. SYSTEM DEVELOPMENT

6.1 HAN interesting features

Inter-concurrency between hierarchical modules, if we consider an algebraic net specification module at a hierarchical level *i*, then all the specification modules of level lower than *i* offer services which can be requested concurrently. This is necessary to describe the concurrency between distributed modules.

Intra-concurrency inside a module, all the services offered by a specification module at a hierarchical level *i* can be requested concurrently by modules of hierarchical level greater than *i*. This allows, either to describe modules which will be implemented on a parallel computer node, or to describe a module which will be distributed in a refinement stage and to express the concurrency before the distribution.

Reaction, each module can define internal transitions which are reactive. This means that their execution is automatic as soon as the pre-conditions are satisfied. This is a way of describing reactive distributed systems in which internal transitions participate to the description of the behaviour of the module initiated by a method request.

Space complexity reduction, the use of algebraic specifications for describing the data structures managed by the Petri nets allows to have short descriptions even if the value domains are infinite. This is an important characteristic since real distributed systems often lead to enormous Petri net descriptions even if considering hierarchical coloured nets.

Synchronous and asynchronous communication between modules, HAN generalises the notion of synchronisation expression introduced firstly by CO-OPN. Each transition can be associated to an expression built with 3 operators: simultaneity, sequence and undeterminism. Thus, complex concurrent executions of services can be requested from a module to several other modules.

Heterogeneity for module resources implementation, each module of a HAN specifies completely its data structures and the semantics of these data types is independent from other modules data structures semantics. Thus, it allows heterogeneous implementations of data structures. Of course a protocol must be defined in order to ensure a coherent interpretation of data transmitted between two different modules.

Uniformity for the construction style of the hierarchies of abstract data types and of the algebraic net modules, in HAN the definition of the behavioural part (Petri net part) as well as the definition of the data structures (algebraic specifications) is made in a hierarchical way. We provide uniform structuring operators for these both parts in order to have an homogenous specification framework.

Abstraction and refinement, the development of distributed concurrent systems is not a simple task and the software life cycle includes many steps. The hierarchical approach is well adapted to a development process based on refinement. Furthermore, HAN provides a way of specifying abstract data types which is useful for refinement and abstraction.

6.2 Tools

The prototyping techniques for distributed systems developed for CO-OPN and presented in [7] and the SANDS tool existing for CO-OPN, provide a way of supporting the development of CO-OPN specifications [8]. An algorithm, transforming CO-OPN specifications into an HAN specification has been developed [6] in order to use SANDS for the development of a subpart of hierarchical algebraic nets and also in order to furnish to CO-OPN specifications a formal semantics in terms of HAN. This algorithm has three main parts : (1) The acyclic graph is transformed into two binary trees (a net tree for the net modules which are also called CO-OPN objects and an algebraic specification tree for the presentations modules) where arcs corresponds to unions or enrichments. All the objects are renamed in order to be disjoint on their net part; (2) all the modules of the net tree are transformed into ANMs such that their associated HAS is the sub tree of the HAS tree including only the data structures used in the ANM; (3) the HAN is obtained by replacing the net modules by the previously built ANMs.

This algorithm needs an initial step which transforms the SANDS specification into an other one without generic modules. This is in fact easily done since SANDS uses genericity only as a syntactic facility.

7. CONCLUSION

Hierarchical algebraic nets are proposed as a generalisation of the theoretical work on CO-OPN which takes into account most of the comments made on CO-OPN during the last five years. Nevertheless they constitute an hybrid specification formalism inheriting from Petri Nets and algebraic specifications and integrating structuring primitives thus introducing a new class of structured algebraic nets. Its syntax and semantics have been given in an synthetic way in order to be complete w.r.t. the initials goals. Hierarchical algebraic nets are used to give a formal semantics to SANDS and are currently used to give a formal semantics to a purely object oriented evolution of CO-OPN. Thus , hierarchical algebraic nets can be seen as a semantic work as well as a practical one for whom the SANDS environment could be enhanced in order to allow the development of system using the entire hierarchical algebraic nets expressive power.

8. ACKNOWLEDGEMENTS

I would like to thanks D. Buchs who gave me some important remarks and ideas concerning this work.

9. REFERENCES

[1] W.Reisig, 'Petri nets and Algebraic specifications', *Theoretical Computer Science*, n°80, pp 1-34, 1991.

[2] D. Buchs, N. Guelfi, "CO-OPN : A Concurrent Object Oriented Petri Net Approach for system specification", 12th *International Conference on theory and application of Petri Nets*, Aahrus, pp 432--454, 1991.

[3] M. Wirsing, "Algebraic Specifications", in Handbook of *Theoretical Computer Science*, J. Van Leeuwen Eds, pp 675-788 , 1990.

[4] M. Bidoit & al. , "Algebraic System Specification and development a survey and annotated bibliography", *LNCS 501*, 1993.

[5] J.Vautherin "Parallel systems specification with coloured Petri nets and algebraic specification" LNCS 266, *International Conference on Application and Theory of Petri Nets*, 1987.

[6] N. Guelfi, "Les réseaux Algébriques Hiérarchiques : un formalisme de spécifications structurées pour le développement de systèmes concurrents", *PhD Thesis, Laboratoire de Recherche en Informatique (LRI), University of Orsay*, France, 1994.

[7] D. Buchs and J. Hulaas, Evolutive prototyping of heterogeneous distributed systems using hierarchical algebraic Petri nets. In Proceedings of the *International Conference on Systems, Man and Cybernetics*, 1996.

[8] D. Buchs, J. Flumet, and P. Racloz, SANDS: Structured algebraic net development system, In Buy Ugo, editor, *14th International Conference on Application and Theory of Petri Nets*, Tool presentation abstracts, pp25--29, Chicago, USA, 1993.

[9] Huber, P.; Jensen, K.; Shapiro R.M., Hierarchies in Coloured Petri Nets, *Advances in Petri Nets,* LNCS 483, pp 313-341, 1991.