A Distributed Algorithm for MLTS Generation with Aggregation of Transitions

Djamel Eddine SAIDOUNI Riadh MATMAT Nacer TABIB
MISC Laboratory: Mentouri University
Constantine, 25000, Algeria
saidounid@hotmail.com, matmat.riadh@gmail.com, tabib.univ@gmail.com

***Abstract*—Enumerative model checking tools are limited by the size of the stat space to which they can be applied. In an attempt to extend the size of state spaces that can be dealt with, we propose in this paper a distributed algorithm that generates a distributed maximality-based labeled transition systems (noted MLTS) from place/transition Petri net while performing aggregation of *equivalent* transitions proposed in [10].**

Keywords: Distributed Maximality-based labeled transition systems, Aggregation of transitions, Petri nets, Distributed algorithm.

I. INTRODUCTION

Model checking [1] is a powerful technique for verifying reactive systems, able to find subtle errors in real commercial designs. Its major advantage is automation, and therefore the ease of use for engineers.

The model checking exhaustively explores the analyzed models. This causes a major problem: combinatorial explosion due to the state spaces of large systems. Many solutions have been proposed in the literature for tackling this problem to push the limit of size, to be able to handle states spaces increasingly of large sizes. Some of them use equivalence relation to reduce the number of states and transitions in the model (bissimulation relations, alpha reduction relation, partial order based relations ...) [2, 3, 4], others use symbolic representations such as BDD [5, 6].

However, even these solutions reduce significantly the size of maximality-based labeled transition systems (MLTS) [7, 8], the resource limitation make yet a problem when coming to complex systems.

Nowadays, workstations clusters give more and more hardware resources availability, which leads to a new solution which is distributing the graph through the workstation network [9, 10].

This work we proposes a distributed algorithm for MLTS construction, combined with aggregation of transitions in order to reduce the graph size while generating it on the fly [11]. Our algorithm implements an operational semantics defined in [12]. So it takes a Petri net as a modeling language and produces an MLTS, it uses also the aggregation of transitions to reduce the graph on the fly, thus it takes the advantage of workstation hardware resources by distributing the graph and the reducing solutions while generating the graph.

II. PRELIMINARIES

A. *Petri nets related definitions*

- A Petri net is a tuple (S,T,W) where S is the set of places, T is the set of transitions such that $S \cap T=\emptyset$, and $W:((S\times T) \cup (T\times S)) \rightarrow \mathbb{N} = \{0,1,2,...\}$ is the weight function. Graphically, transitions of T are represented by rectangles, places of S by circles and weight function by arrows associated with their weights. We suppose that all nets are finite, i.e. $|S \cup T| \in \mathbb{N}$.
- For $x \in S \cup T$ the pre-set ${}^\bullet x$ is defined by ${}^\bullet x = \{y \in S \cup T \mid W(y, x) \neq 0\}$ and the post-set $x^\bullet$ is defined by $x^\bullet=\{y \in S \cup T \mid W(y,x) \neq 0\}$.
- The marking of a Petri net (S,T,W) is defined as a function $M: S\rightarrow\mathbb{N}$. A marking is generally represented graphically by putting tokens in places.
- The transition rule stipulates that a transition t is enabled by M iff $M(s) \geq W(s, t)$ for all $s \in S$. The firing of a transition t will produce a new marking M' defined by $M'(s) = M(s) - W(s, t) + W(t, s)$ for all $s \in S$. The occurrence of t is denoted by $M [t\rangle M'$.
- Two transitions t_1 and t_2 (not necessarily distinct) are concurrently enabled by a marking M iff $M(s) \geq W(s, t_1) + W(s, t_2)$ for all $s \in S$.
- A marked Petri net (S,T,W,M_0) is a Petri net (S,T,W) with an initial marking M_0.
- An alphabet $\mathcal{A}$ is a finite set; we suppose that $\tau \notin \mathcal{A}$ (τ will indicate invisible action, or silent action).
- The labeling of a Petri net $N = (S,T,W)$ is a function $\lambda : T \rightarrow \mathcal{A} \cup \{\tau\}$. If $\lambda(t) \in \mathcal{A}$ then t is said to be observable or external; at the opposite, t is silent or internal.
- $\Sigma=(S,T,W,M_0,\lambda)$ is a labeled system iff (S,T,W,M_0) is a marked Petri net and l is a labeling function of (S,T,W).
- An action $a \in \mathcal{A}$ of a system $\Sigma = (S,T,W,M_0,\lambda)$ is auto-concurrent in a marking M iff M concurrently enables two observable transitions t_1 and t_2 (not necessarily distinct) such that $\lambda(t_1) = \lambda(t_1) = a$.

- A sequence $\sigma = M_0t_1M_1t_2...$ is an occurrence sequence iff $M_{i-l}\ [ti\rangle M_i$ for $l \leq i$. A sequence t_1t_2 ... is a transition sequence starting with M iff there is an occurrence sequence $M_0t_1M_1t_2...$ If a finite sequence $t_1t_2...\ t_n$ leads from M to M' we write $M[t_1t_2\ ...\ t_n\rangle M'$. The set of reachable markings of a marked Petri net (S,T,W,M_0) is defined as $[M_0\rangle = \{M \mid t_1t_2\ ...\ t_n : M_0[\ t_1t_2\ ...\ t_n\rangle M\}$.

B. *Maximality based Labeled Transition System*

Let $\mathcal{M}$ be a countable set of events names, maximality based labeled transition system of support $\mathcal{M}$ is a quintuplet: $(\Omega, \lambda, \mu, \xi, \psi)$ with:

$\Omega = (S, T, \alpha, \beta)$ is a transition system such that:

S is the countable set of states in which the system can be found.

T is the countable set of transitions indicating the change of system states.

α and β are two applications of T in S such that for all transition t we have: $\alpha(t)$ is the origin of the transition and $\beta(t)$ its goal.

(Ω, λ) is a transition system labeled by an alphabet A.

$\psi : S \rightarrow 2^{\mathcal{M}}$ which associates to each state the finite set of maximal event names present in this state.

$\mu : T \rightarrow 2^{\mathcal{M}}$ is a function which associates to each transition the finite set of event names corresponding to actions that have already begun their execution and of which the end of execution enables this transition.

$\xi : T \rightarrow M$: is a function which associates to its transition an event name identifying its occurrence. Such that for any transition t, $\mu(t) \subseteq \psi(\alpha(t))$, $\xi(t) \notin \psi(\alpha(t)) - \mu(t)$ and $\psi(\beta(t)) = (\psi(\alpha(t)) - \mu(t)) \cup \{\xi(t)\}$.

C. *Operational maximality semantics for Petri nets*

In this section, we introduce a notations and functions for the definition of marking graph associated to a labeled system in a maximality-based approach.

Definition 1 Let $N = (S, T, W)$ be a Petri net, the marking of N is a function $M: S \rightarrow \mathbb{N} \times 2^{\mathbb{N} \times T \times \mathcal{M}}$. Among others, the marking $M(s)$ of a place $s \in S$ is a pair $(\mathcal{FT}, \mathcal{BT})$ such that $\mathcal{FT} \in \mathbb{N}$ and $\mathcal{BT} \in 2^{\mathbb{N} \times T \times \mathcal{M}}$ denote respectively the number of free tokens and the set (possibly empty) of bound tokens in the place s. In what follows, a Petri net with a marking will be called configuration. $|M(s)|$ denote the total number of tokens in a place s. If $M(s) = (\mathcal{FT}, \mathcal{BT})$ such that $\mathcal{BT} = \{n_1t_1x_1,..,n_mtm_{xm}\}$ then $|M(s)| = \mathcal{FT} + |\mathcal{BT}|$ with $|\mathcal{BT}| = \sum_{i=1}^{m} n_i$ is the cardinal of the bound tokens set in s.

Definition 2 Let (S, T, W) be a Petri net with a marking M:

- The set of maximal event names in M is the set of all event names identifying bound tokens in the marking M. Formally, the function ψ will be used to calculate this set, it can be defined as $\psi(M) = \bigcup_{s \in S} \bigcup_{i=1}^{ms}$ such that $M(s) = (\mathcal{FT}, \mathcal{BT})$ with $\mathcal{BT} = \{(ns_1, ts_1, xs_1), \ldots, (ns_{ms}, ts_{ms}, xs_{ms})\}$.
- Let $N \subset \mathcal{M}$ be a non-empty finite set of event names, *makefree (N, M)* is defined recursively by:

  makefree $(\{x_1,x_2,\ldots x_n\}, M) =$ *makefree* $(\{x_2,\ldots, x_n\}$, *makefree* $(\{x_1\},M))$
  makefree $(\{x, M) = M'$ such that for all $s \in S$, if $M(s) = (\mathcal{FT}, \mathcal{BT})$ then
  - If there is $(n, t, x) \in \mathcal{BT}$ then $M'(s) = (\mathcal{FT}+n, \mathcal{BT} - \{(n, t, x)\})$ (Conversion of n bound tokens identified by the event name x to free tokens).
  - Otherwise, $M'(s) = M(s)$.
- Let t be a transition of T; t is said to be enabled by the marking M iff $|M(s)| \geq W(s, t)$ for all $s \in S$. The set of all transitions enabled by the marking M will be noted *enabled (M)*.
- The marking M is said to be minimal for the firing of the transition t iff $|M(s)| = W(s, t)$ for all $s \in S$.
- Let M_1 and M_2 be two markings of the Petri net (S,T, W). $M_1 \Subset M_2$ iff $\forall s \in S$, if $M_1(s) = (\mathcal{FT}_1, \mathcal{BT}_1)$ and $M_2(s) = (\mathcal{FT}_2, \mathcal{BT}_2)$ then $\mathcal{FT}_1 \leq \mathcal{FT}_2$ and $\mathcal{BT}_1 \Subset \mathcal{BT}_2$ such that the relation $\Subset$ is extended to bound tokens sets as follows:
  - $\mathcal{BT}_1 \Subset \mathcal{BT}_2$ iff $\forall (n_1, t, x) \in \mathcal{BT}_1, \exists (n_2, t, x) \in \mathcal{BT}_2$ such that $n_1 \leq n_2$.
- Let M_1 and M_2 be two markings of the Petri net (S,T,W) such that $M_1 \Subset M_2$. The difference M_2-M_1 is a marking M_3 (M_2 - M_1 = M_3) such that for all $s \in S$, if $M_1(s) = (\mathcal{FT}_1, \mathcal{BT}_1)$ and $M_2(s) = (\mathcal{FT}_2, \mathcal{BT}_2)$ then $M_3(s) = (\mathcal{FT}_3, \mathcal{BT}_3)$ with
  - $\mathcal{FT}_3 = \mathcal{FT}_2 - \mathcal{FT}_1$
  - $\forall (n_1, t, x) \in \mathcal{BT}_1, (n_2, t, x) \in \mathcal{BT}_2$ if $n_1 \neq n_2$ then $(n_2 - n_1, t, x) \in \mathcal{BT}_3$.
- *Min* $(M, t) = \{M' \mid M' \Subset M\}$ and M' is minimal for the firing of t.
- Let $\mathcal{M}$ be a set. The function get: $2^{\mathcal{M}}-\{\emptyset\} \rightarrow \mathcal{M}$ is a function which satisfies $get(E) \in E$ for any $E \in 2^{\mathcal{M}}-\{\emptyset\}$.
- Given a marking M, a transition t and an event name $x \notin \psi(M)$, *occur* $(t, x, M) = M'$ such that $\forall s \in S$, if $M(s) = (\mathcal{FT}, \mathcal{BT})$ then $M'(s) = (\mathcal{FT}, \mathcal{BT}')$ with $\mathcal{BT}' = \mathcal{BT} \cup \{W(t, s), t, x)\}$ if $W(t, s) \neq 0$ and $\mathcal{BT}' = \mathcal{BT}$ otherwise. Hence, M' is the resultant marking from the addition of tokens bound to t to the marking M.

III. Operational Maximality Semantics for Petri Nets with Aggregation of Transitions

Usually, marking graph is generated by the operational maximality for Petri nets, thus we keep the same basic definitions, but to achieve our goal we must change the semantics of the function *Min*. In this case, a minimal marking

for the firing of a transition *t* is considered as an element of the set *Min (M, t)* only if for each place of this marking, bound tokens are only taken in the case when the free part does not satisfy the pre-condition of this transition. Therefore, we can ensure that a transition *t* will be executed sequentially after a transition *t´* if it cannot be executed independently with this same transition *t´.*

Formally, *Min (M, t)* is the set of markings $M' \in M$ such that for any state *s* where *M(s) =* $(\mathcal{FT}, \mathcal{BT})$, *M'(s)* is defined as follows:

$$M'(s) = \begin{cases} (w(s,t), \emptyset) & if \mathcal{FT} \geqslant w(s,t) \\ (\mathcal{FT}, \mathcal{BT}') & otherwise \end{cases}$$

With $\mathcal{BT}' \in \mathcal{BT}$ and $|\mathcal{BT}'| = w(s, t) - \mathcal{FT}$

IV. SEQUENTIAL GENERATION OF MARKING GRAPH

In Algorithm 1 we have three sets (L, V, and T), where L contains the set of configurations not yet explored, V contains the set of configurations that have been already developed and T contains the MLTS.

Algorithm 1

```
Begin
L :={C0}
V := ∅;
T := ∅;
While L != ∅ Do
    config := pop(L) ;
    push(config, V) ;
    For all enabled Transition Do   /*Petri net transitions*/
       // Exhaustive Development
       For eash new configuration s′ Do
         check if there exist s′ in L or in V
         If  not exist  s′ Then
             push(s′, L};
             create new transition(s,edge,s′)
             T := T  ∪ {(s,edge,s′)};
         Else
             create new transition(s,edge,si);
             T := T  ∪ {(s,edge, si)};
         endif
       endfor
    endfor
endwhile
End
```

In order to generate the MLTS we initialize the set L by the initial configuration and by applying the maximality operational semantics for Petri net we get a new configuration to be developed. Thus we add a transition (s, edge, s′) that does not already exist, to T for each new configuration [7].

V. DISTRIBUTED GENERATION OF MLTS WITH AGGREGATION OF TRANSITIONS

Algorithm 2

```
Data : initial configuration C0;
Variables :
NT_configi : the list of configurations not yet explored
T_configi : the list of configurations explored
Si : the list of state the MLTS
Ti : the list of transition the MLTS
Begin
(a) If hash(C0) = i Then
    NT_configi.push(C0)
Endif
While (shutdown signal not received) Do
   (b)While NT_configi is not empty Do
         conf := pop(NT_configi) ;
       push(conf, T_configi) ;
       For eash  t in Transition Do  /*t is a Petri net transition*/
          If enabled(t) Then
            MinAggreg(conf,t) /*using aggregation of transition t*/
            Exhaustive development /* SOS of Maximality */
            For each new configuration new_conf Do
            (c)If hash(new_conf) = i then
             check if there exist new_conf in NT_configi or
                  in T_configi
                  If new_conf not exist Then
                    push(new_conf,NT_configi};
                    create new transition (conf,edge, new_conf)
                    Si = Si ∪ {new_conf};
                    Ti := Ti  + (conf,edge,new_conf);
                  Else
                    create new transition (conf,edge,confi);
                    Ti := Ti + (conf,edge,confi);
                  Endif
                Else
                  create new transition(conf,edge,new_conf);
                  Ti := Ti + (conf,edge,new_conf);
                  SEND(new_conf,hash(new_conf)) ;
            Endfor
          Endif
       Endfor
     Endwhile
     (d) If RECEIVE(conf_d) Then
          check if there exist conf_d in NT_configi or in T_configi
          If config_d not exist Then
              push(conf_d, NT_configi};
          Endif
       Endif
     (e)If RECEIVE( terminate) Then
          If NT_configi = ∅  Then
              Procedure_Terminisation();
     (f) If isInitiator and NT_configi = ∅  Then
          Init_Terminisation();
Endwhile
End.
```

Figure 1:Marked Petri net

We take an example to better explain the algorithm. We suppose the Petri net in "Fig. 1" from which we generate the distributed MLTS.

We distribute the generated MLTS through three nodes of workstations network *N=3*. Referring to maximality operation semantics [10] the configuration for this Petri net is C_0= *(<1, Ø><1, Ø><0, Ø>)*

For each node *i* we initiate $NT\_config_i = Ø$, $T\_config_i = Ø$. $S_i = Ø$, $T_i = Ø$. In the first step (a) in the algorithm we apply the hash function defined as:

*hash (new_conf.toString())=31*MD5(new_conf.toString()) mod 3* to the initial configuration ,thus we get the initiator node that inserts C_0 into NT_config_i then we start the process of distribute generation. The results in our case are:

C_0*.toString () = "1, {} 1, {} 0, {}",*

md5("1,{}1,{}0,{}") = 4ef58d8d06188dbd58bf39f5877c7285, hash(C_0*.toString()) = 0;* so the node 0 will be the initiator and NT_config_0 *={ (<1, Ø><1, Ø><0, Ø>)}*, T_config_0*= Ø.*

In the step (b) in algorithm 2 from the initial configuration we get two new configurations "Fig. 2" because t_0 and t_1 are both enabled, we apply the hash function we get:

C_1*= (<0, Ø><1,1t0x><0, Ø>).*

hash ("0, {} 1,1t0x 0, {} ") =1

C_2*= (<1, Ø><0, Ø ><0, 1t1x >)*

hash ("1,{}0,{}0,1t1x") =2

Hence we send each new configuration to its corresponding node (step c), then:

NT_config_1*= {(<0, Ø><1,1t0x><0, Ø>)}*

T_config_1*= Ø.*

NT_config_2*= {(<1, Ø><0, Ø ><0, 1t1x>)}*

T_config_2*= Ø.*

- The node number one receives the new configuration *(<0, Ø><1, 1t0x><0, Ø>)* step (d) and start computation. In the first iteration (b) the transition t_1 is enabled so two new configurations can be generated each one from either the bound token or the free token but by applying the reduction by aggregation of transition t_1 one configuration must be kept "Fig. 3",in this case the algorithm keeps *(<0,Ø><0,1t0x><0,1t1x>)* then calculates its corresponding *hash* : *hash("0,{}0,1t0x0,1t1y")=1*. The value corresponds to the node itself so the configuration won't be sent. The node must update its variables:

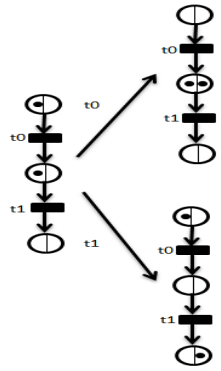

Figure 2: New markings by firing two transitions

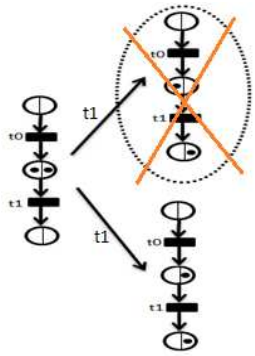

Figure 3: Free tokens and bound tokens in a marking

NT_config_1*= {(<0, Ø><0,1t0x ><0, 1t1y>)}*

T_config_1*= {(<0, Ø><1,1t0x><0, Ø>)}.*

- The node number two also start computation in parallel with the node number one and in the first iteration (b) we can generate from the configuration received the new configuration *(<0, Ø><0,1t0y><0, 1t1x>)* where*: hash(0,{}0,1t0y0,1t1x)=0* and thus will be send to node 0.

NT_config_2*={}*

T_config_2*={(<1, Ø><0, Ø ><0, 1t1y>)}*

NT_config_0*={(<0, Ø><0,1t0y><0, 1t1x>) }*

T_config_0*={(<1, Ø><1, Ø ><0, Ø>)}*

Since NT_config_1 and NT_config_0 are non-empty, the node zero and node one continues working and node 2 wait for receiving new configurations. From the configuration *(<0, Ø><0, 1t0y><0, 1t1x>)* located in node 0 and the configuration *(<0, Ø><0, 1t0x ><0, 1t1y>)* located in node 1 we obtain the configuration *(<0, Ø><0, Ø><0, 1t1x 1t0y>).* The two nodes zero and one get the same hash value : *hash(0,{}0,{}0,1t1x1t0y)=2* .

The node 2 receives this configuration which is no more enabled and thus no new configuration can be generated. Each node must update its variables:

$NT\_config_0=\{\}$

$T\_config_0=\{(<1, \emptyset><1, \emptyset><0, \emptyset),(<0, \emptyset><0,1t0y><0, 1t1x>))\}$

$NT\_config_1=\{\ \}$

$T\_config_1=\{(<0, \emptyset><1,1t0x><0, \emptyset>),(<0, \emptyset><0,1t0x><0,1t1y>)\ \}.$

$NT\_config_2=\{\}$

$T\_config_2=\{(<1,\emptyset><0,\emptyset><0,1t1y>),(<0, \emptyset><0, \emptyset><0,1t1x\ 1t0y>)\}$

All lists not yet explored are empty; the termination is thus detected by the initiator node which is the node 0 in this case. The obtained MLTS is represented in "Fig. 4".

VI. DISTRIBUTED TERMINATION DETECTION

The principle of the termination detection algorithm used in Algorithm 2 is done in [12] by a two-wave algorithm. All machines have processed all their configuration and no more messages are in transit in the queues. Here, we have used a simple technique when all lists are empty, the termination is detected.

For more information about distributed termination detection we refer the reader to [13].

VII. IMPLEMENTATION AND EXPERIMENTATION

To assess the feasibility of our approach, we have developed an environment (D-STEM-PNet) that generates either a centralized or a distributed reduced MLTS by the aggregation of transitions presented in the algorithms 1 and 2. This environment has a graphical editor to draw and edit Petri nets **"**Fig. 5" and a result viewer "Fig. 6, 7" that produces a dot file type [14].

We have used java as programming language to implement all the pieces of the environment. For the distributed algorithm we executed the implementation on a network of 7 pcs 2.4 Ghz Pentium with 256 MBytes of RAM connected with 100 Mbps Ethernet. For the communication we have used jade environment [15] so each node is presented by an agent.

Case study :

In order to illustrate the interest of the proposed approach, we study in this section an example of processes synchronization, namely ticket reservation system illustrated in "Fig. 5".

The algorithm generates 20076 states without aggregation of transitions dispersed over 7 nodes as shown in "Fig.7" and 10999 with aggregation "Fig. 8" hence the ratio of reduction in percent by each node is shown in "Table. I". The ratio reduction of the entire graph is about 45%. Thus the aggregation of transitions gives important results. In addition, the graph is distributed over nodes and by this we limit the combinatorial explosion problem.

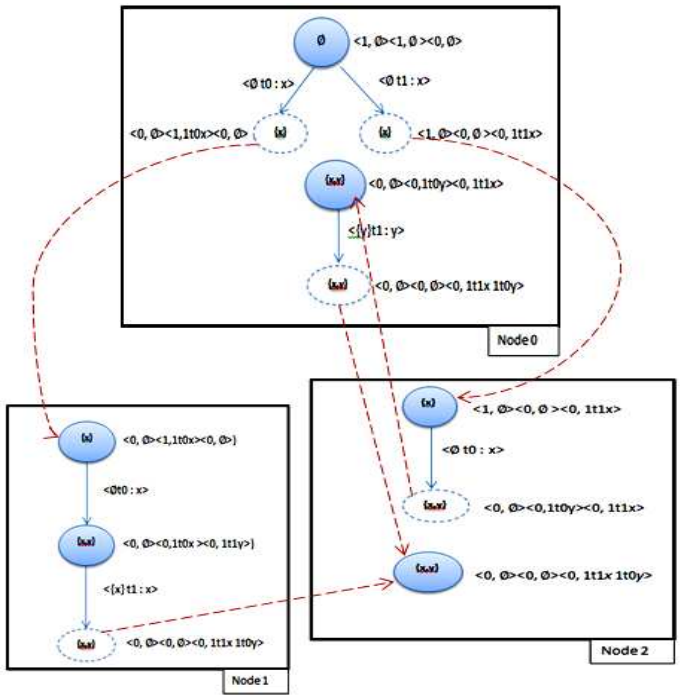

Figure 4: A distributed MLTS over 3 nodes

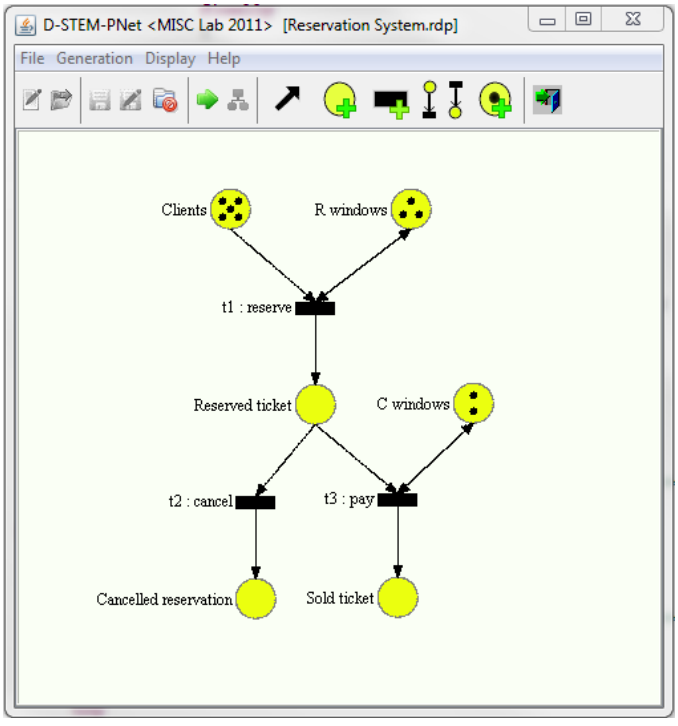

Figure 5: graphical tool for editing Petri nets

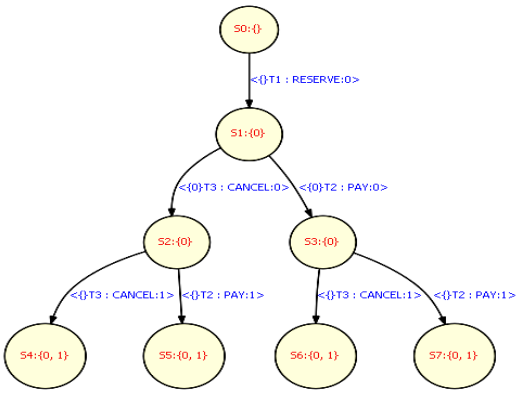

Figure 6: graph viwer tool

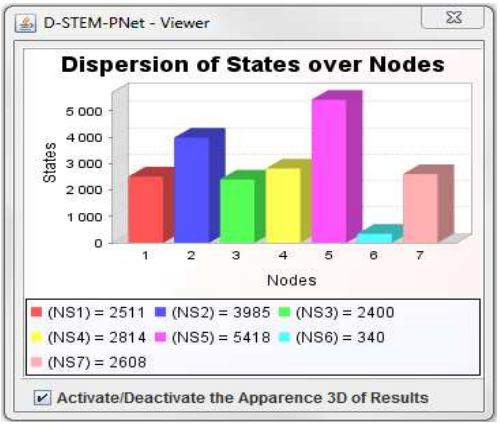

Figure 7. Distribution of states over 7 nodes without aggregation

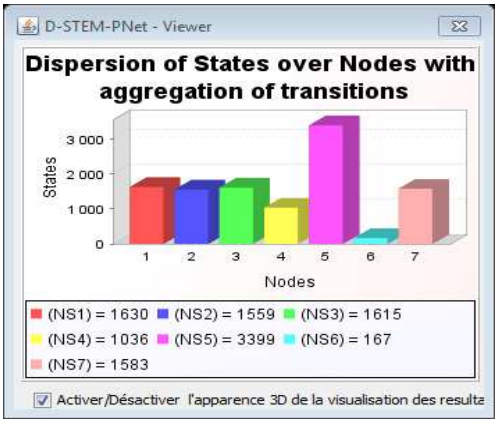

Figure 8. Distribution of states over 7 nodes with aggregation

TABLE I. TABLE I. RATE RESULTS

Nodes	Before reduction	After reduction	Reduction rate %
1	2511	1630	35
2	3985	1559	61
3	2400	1615	33
4	2814	1036	63
5	5418	3399	37
6	340	167	51
7	2608	1593	39
Total	20076	10999	45

VIII. CONCLUSION

In this work we proposed a distributed algorithm for generating a distributed MLTS in order to push the limit of size to be able to handle spaces states increasingly of large sizes. We have also applied the aggregation of transitions to reduce the size in each node as well. This work gives significant results in reducing the graph and distributing it over nodes in order to fight against the combinatorial problem, however using a hash function for distributing the states may lead to huge communication between the nodes and less connectivity between states, which pushes us to try another methods for reducing the graph such as using a distributed maximality bissimulation and try to take another direction as well to increase the connectivity between states.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. ACM Transactions on Programming Languages and Systems, 16(5) :1512–1542, septembre 1994.

[2] D. E. Saidouni and A. Benamira, La Alpha-Reduction à la Volé Des Systémes de Transitions etiquetés Maximales, Master thesis, LIRE Laboratory, University of Mentouri, Constantine, Algeria, 2004.

[3] P. GodeFroid, Using Partial Orders to Improve Automatic Verification Methods, in Proceedings of CAV'90, volume 3, pages321-340, ACM, DIMACS, 1990

[4] P. Godefroid and P. Wolper. A Partial Approach to Model Checking, in Proceedings 6th Symp. On Logic in Computer Science, volume 531,pages 406-415, Amsterdam 1991.

[5] R. Drechsler, et B. Becker. Binary Decision Diagrams: Theory and Implementation. Kluwer Academic Publisher, 1998.

[6] R.E. Bryant. Graph-based algorithm for boolean function Manipulation. IEEE Transactions on Computer Science, 37: 77-121, 1986.

[7] D. E. Saidouni. Semantique de Maximalité: Application Au Rafinement D'actions Dans LOTOS. PhD thesis, LAAS, Université Paul Sabatier, Toulouse (Mars 1996).

[8] J. P. Courtiat and D. E. Saidouni. Relating maximality based semantics to action refinement in process algebras. In " D. Hogrefe and S. Leue, Editors, IFIP TC/WG6.1, 7th Int. Cof of Formal Description Techniques(FORTE'94)", pages 293- 308. Chapman Hall (1995).

[9] H. Garavel, R. Mateescu, and I. Smarandache. Parallel state space Construction for model-checking. In Proc. 8th Inter. SPIN Workshop, volume LNCS 2057, pages 217-234. Springer, 2001.

[10] D. E. Saïdouni, N. Belala, and M. Bouneb. Aggregation of transitions in marking graph generation based on maximality semantics for Petri nets.In Proceedings of the Second International Workshop on Veri?cation andEvaluation of Computer and Communication Systems (VECoS'2008), Uni-versity of Leeds, UK. eWiC Series, The British Computer Society (BCS),July, 2-3rd 2008. ISSN: 1477-9358..

[11] D. E. Saïdouni, N. Belala, and M. Bouneb. Maximality-based structu-ral operational semantics for Petri nets. In Proceedings of INTELLIGENT SYSTEMS AND AUTOMATION: 2nd Mediterranean Conferenceon Intelligent Systems and Automation (CISA'09), Zarzis, Tunisia, volume1107, pages 269–274. American Institute of Physics, March 23th-25th 2009.

[12] F. Mattern. "Algorithms for Distributed Termination Detection. Distributed Computing", 1987.

[13] J. Misra and K. M. Chandy, Termination detection of diffusing Computations in communication sequential processes, ACM Transactions on Programming Languages and Systems, January, 1982, 37-42.

[14] http://jade.tilab.com/ ,2011

[15] http://www.graphviz.org/ ,2011