Dipartimento di Informatica
Università del Piemonte Orientale "A. Avogadro"
Viale Teresa Michel 11, 15121 Alessandria
http://www.di.unipmn.it

Performance analysis of partially symmetric SWNs: efficiency characterization through some case studies

Authors:S. Baarir, C. Dutheillet, M. Beccuti, G. Franceschinis and S. Haddad
({souheib.baarir,claude.dutheillet}@lip6.fr,beccuti@di.unito.it, giuliana@mfn.unipmn.it,haddad@lsv.ens-cachan.fr)

TECHNICAL REPORT TR-INF-2009-07-06-UNIPMN
(July 2009)

The University of Piemonte Orientale Department of Computer Science Research Technical Reports are available via WWW at URL `http://www.di.unipmn.it/`.
Plain-text abstracts organized by year are available in the directory

Recent Titles from the TR-INF-UNIPMN Technical Report Series

2009-05 *SAN models of communication scenarios inside the Electrical Power System*, Codetta-Raiteri, D., Nai, R., July 2009.

2009-04 *On-line Product Conguration using Fuzzy Retrieval and J2EE Technology*, Portinale, L., Galandrino, M., May 2009.

2009-03 *GSPN Semantics for Continuous Time Bayesian Networks with Immediate Nodes*, Portinale, L., Codetta-Raiteri, D., March 2009.

2009-02 *The TAAROA Project Specification*, Anglano, C., Canonico, M., Guazzone, M., Zola, M., February 2009.

2009-01 *Knowledge-Free Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids*, Anglano, C., Canonico, M., February 2009.

2008-09 *Case-based management of exceptions to business processes: an approach exploiting prototypes*, Montani, S., December 2008.

2008-08 *The ShareGrid Portal: an easy way to submit jobs on computational Grids*, Anglano, C., Canonico, M., Guazzone, M., October 2008.

2008-07 *BuzzChecker: Exploiting the Web to Better Understand Society*, Furini, M., Montangero, S., July 2008.

2008-06 *Low-Memory Adaptive Prefix Coding*, Gagie, T., Nekrich, Y., July 2008.

2008-05 *Non deterministic Repairable Fault Trees for computing optimal repair strategy*, Beccuti, M., Codetta-Raiteri, D., Franceschinis, G., July 2008.

2008-04 *Reliability and QoS Analysis of the Italian GARR network*, Bobbio, A., Terruggia, R., June 2008.

2008-03 *Mean Field Methods in performance analysis*, Gribaudo, M., Telek, M., Bobbio, A., March 2008.

2008-02 *Move-to-Front, Distance Coding, and Inversion Frequencies Revisited*, Gagie, T., Manzini, G., March 2008.

2008-01 *Space-Conscious Data Indexing and Compression in a Streaming Model*, Ferragina, P., Gagie, T., Manzini, G., February 2008.

2007-05 *Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids: a Knowledge-Free Approach*, Canonico, M., Anglano, C., December 2007.

2007-04 *Verifying the Conformance of Agents with Multiparty Protocols*, Giordano, L., Martelli, A., November 2007.

2007-03 *A fuzzy approach to similarity in Case-Based Reasoning suitable to SQL implementation*, Portinale, L., Montani, S., October 2007.

2007-02 *Space-conscious compression*, Gagie, T., Manzini, G., June 2007.

2007-01 *Markov Decision Petri Net and Markov Decision Well-formed Net Formalisms*, Beccuti, M., Franceschinis, G., Haddad, S., February 2007.

2006-03 *New challenges in network reliability analysis*, Bobbio, A., Ferraris, C., Terruggia, R., November 2006.

Contents

Performance analysis of partially symmetric SWNs: efficiency characterization through some case studies

S. Baarir, C. Dutheillet
LIP6, Université Paris 6
104 avenue du Prsident Kennedy
Paris, France
{souheib.baarir,claude.dutheillet}@lip6.fr

M. Beccuti
Dip. di Informatica, Univ. degli Studi di Torino
C.So Svizzera, 185
10149 Torino Italy
beccuti@di.unito.it

G. Franceschinis
Dip. di Informatica, Univ. del Piemonte Orientale
Via Bellini, 25/G
15100 Alessandria, Italy
giuliana@mfn.unipmn.it

S. Haddad
LSV, ENS Cachan, CNRS
61, avenue du Président Wilson
Cachan, France
haddad@lsv.ens-cachan.fr

Abstract

Performance and dependability evaluation of complex systems by means of *dynamic stochastic models* may be impaired by the combinatorial explosion of their state space. Among the possible methods to cope with this problem, symmetry-based ones can be applied to systems including several similar components. However, symmetry-based methods are less effective in case of partially symmetric systems (i.e., systems with mostly symmetric behavior and occasional local asymmetric behavior). To cope with this limitation, two approaches called *Extended Symbolic Reachability Graph* and *Dynamic Symbolic Reachability Graph* have been proposed in literature. In this paper we will discuss the effectiveness and applicative interest of these two methods by means of four model patterns.

1 Introduction

As software systems and hardware architectures are more and more complex, their verification and evaluation become critical issues. Analysis methods are often subject to the problem of combinatorial explosion due to the increasing system complexity.

Several approaches have been undertaken to cope with this problem: decomposition methods take advantage of the modular structure of the system; for performance evaluation, approximate and bounding methods

substitute a simpler system to the original one; diagram decision based methods symbolically manage sets of states rather than representing states explicitly, etc. Among all these methods presented in literature we will focus on the symmetry-based ones, and in particular on the Well-formed Net (WN) formalism and its stochastic extension (SWN) [5].

The (S)WN formalism takes advantage from the system symmetries to cope with *state space explosion problem*, and hence provide efficient solution techniques for both qualitative and quantitative analyzes.

Based on these symmetries, a quotient graph, called *Symbolic Reachability Graph (SRG)* [5], is automatically constructed from an (S)WN. Each node of the SRG, called Symbolic Marking (SM), represents a set of ordinary markings. In case of *highly symmetric systems*, the expected size of the SRG is significantly reduced w.r.t. the one of the ordinary Reachability Graph (RG). However, it is less effective, in case of partially symmetric systems, i.e., systems with mostly symmetric behavior and occasional locally asymmetric behavior. In fact, in the SRG approach asymmetries are always taken into account, even if the asymmetric behavior of the system is local.

To cope with this limitation, in [7] a more compact structure called *Extended SRG (ESRG)* has been proposed. The idea is to group into Extended SMs (ESM) sets of (partially) similar SMs. In this representation, in case of *locally symmetric behavior*, the set of SMs captured in an ESM are kept implicit and represented by a unique symbolic Symmetric Representation (SR).

Unfortunately, the derivation of a CTMC from the ESRG is not straightforward: in the general case, the SMs aggregation suggested in the ESRG approach does not satisfy the (strong or exact) lumpability condition as on the SRG. Hence, the ESRG structure has to be refined according to the desired lumpability condition. In [3, 6] an efficient refinement algorithm is proposed. It is based on the Paige and Tarjan partition refinement algorithm and exploits the information contained in the ESRG.

Another approach, called *Dynamic SRG (DSRG)*, was proposed in [1, 2]. It relies on a separate representation of the system asymmetries in a so called *control automaton*. The DSRG is then obtained by synchronizing the transitions of a (symmetric) SWN with the corresponding control automaton. It is worth noting that DSRG satisfies the exact lumpability condition by construction, so that it does not need further refinement. The criteria for states aggregation applied by DSRG method is such that the set of states represented by different aggregates may have a non null intersection.

The presentation of the two methods is out of the scope of this document. This report is organized as follows: in Sec. 2 we discuss how to compare the two approaches, while in Sec. 3 significant case studies are presented and analyzed. We conclude in Sec. 4.

2 Expected efficiency of the behavior of the methods

It is not easy to characterize the kind of models for which ESRG and DSRG lead to a relevant reduction of the state space size. There are least two problems related with this characterization:

- The "degree" of asymmetry of the system is not directly related with the number of asymmetrical transitions of the system but it depends on when they occur in the dynamics of the system. This was already experienced by the qualitative methods for partially symmetrical systems.
- The extent of the propagation of asymmetry is difficult to forecast due to the constraints associated with the lumpability conditions.

In the light of these two problems and with the help of numerous models that we have tested since the development of these methods, we can suggest some model patterns for which our algorithms perform

efficiently. The behavior of these systems can be described as an infinite loop where every round of the loop consists of:

- a synchronization point where the consequences of the past asymmetrical behavior are completely forgotten: this corresponds to reaching one or more "regeneration states" w.r.t. any past of asymmetrical behavior.
- a symmetrical behavior phase and an asymmetrical behavior phase that may partially overlap.

We compare the proposed methods w.r.t. different criteria. The size of the lumped chain is often smaller when applying the ESRG method as it is based on the minimization of a MC w.r.t. lumpability. However the ESRG method requires to explicitly develop "asymmetrical" set of states during the refinement process (that may then be aggregated in the final structure) thus facing the problem of a peak in memory usage, contrary to the DSRG method. Moreover another peak in memory usage may also be experienced during the ESRG generation; in fact the ESRG implementation maintains explicitly the eventualities already reached of a saturated ESM expect if an its eventualities will be reachable.

The memory peak in the refinement process can be reduced by discarding all the eventualities in a block and rebuilding them if needed, while the ESRG memory peak cannot be avoided, so that it may prevent the termination of the algorithm.

On one hand, when a model is efficiently handled by the methods, the final sizes are of the same magnitude order. On the other hand, when the asymmetry of the model propagates throughout the state space, it may yield a combinatorial explosion and the size of the lumped chain of the DSRG method may become bigger than the original one. This cannot happen by construction with the ESRG method. Notice that some artificial partitioning added to DSRG method could also avoid this problem; however in this case, it is better to apply the other methods. Finally, the DSRG method is parametrized in the following sense: as lumping is based on labels the modeler can freely change the numerical values associated with labels without need to recompute the graph associated with the lumped chain; only the numerical values have to be updated. In order for the ESRG method to support such a parametrization the refinement phase should be adapted to work on transition labels instead of rate values.

In the next section we check whether these assumptions agree with the experiments.

3 Case studies

In this section we present the following four "model patterns": readers-writers, client-server, workflow and cluster computing. For each pattern we discuss in details the obtained experiment results and we highlight the model parameters that have a more significant impact on the reduction factor (i.e $\frac{SRG}{ESRG_e}$, $\frac{SRG}{ESRG_s}$ and $\frac{SRG}{DSRG}$, where SRG, $ESRG_e$, $ESRG_s$ and $DSRG$ denote the number of states of the SRG, the refined ESRG w.r.t. exact lumpability, the refined ESRG w.r.t. strong lumpability, and $DSRG$ respectively.

3.1 The readers-writers pattern

The readers-writers pattern models a database which is accessed by users for read or write operations. Read operation may be simultaneous while write operation are mutually exclusive w.r.t any operation. Thus as soon as a write request is queued, it waits for end of any currently ongoing operation before performing its transaction without concurrent accesses.

Read operation duration has wless variability than writes. Thus we have chosen to represent their distribution by an Erlang distribution while the duration of write operation has two possible exponential distributions depending on the class of users : ordinary ones that perform unitary updates and administrators that perform a batch of updates.

The net that models this pattern is shown in Fig. 1. A token in place *Think* represents a user performing some local activity, while a token in a place *WaitChoice* represents an accepted request. A request is received and accepted by the database (transition *Start*), if there are not pending write operations (inhibitor arc between place *ReadyWrite* to transition *Think*).

After that, the request type is randomly chosen by the firing of transitions *ChoiceRead* or *ChoiceWrite*. If the operation is a read, then it can be immediately executed (transitions *Read* and *EndRead*), otherwise it must wait until all the accepted read operations finish (inhibitor arc between place *ReadyWrite* and transitions *WriteAsMaster* and *WriteAsNormal*). Moreover the guards associated with transitions *WriteAsMaster* and *WriteAsNormal* ensure that the write task duration is chosen according to the user type.

We can observe that this pattern follows the general abstract schema described in the previous section. The synchronization points occur after every writing. Then there is a (possibly empty) phase of readings which corresponds to the symmetrical part of the behavior (submodel N_1). The asymmetrical part of the behavior (submodel N_2) starts at the beginning of a writing since only in that case the kind of user matters. Observe that there is no overlapping between the two phases since the readings are ended before the writing starts.

The results for this model pattern are very good as witnessed by table 1. The experiment parameters are (in order) the number of stages of the Erlang distribution, the number of administrators and the number of ordinary users (see the first column).

Moreover the columns labeled "St." represents the number of constructed states for each structure. The columns labeled "Peak" contain the total number of intermediate states stored to obtain the final structure (only for ESRG and RESRGs). Finally the ratio columns show the reduction factor w.r.t the SRG obtained using these three methods.

Observe that the parameter that has more impact on the reduction factor is the number of system users (ordinary and administrator users), so that the reduction factor is increasing w.r.t to this parameter. For instance, the experiment with 13 users leads to a reduction factor of approximately 30 whatever the method.

3.2 The client-server pattern

The client-server pattern, in Fig. 2, is composed of a finite number of terminals and a Remote Terminal Server (RTS). In subnet N_1, the initial marking of place *Clients* corresponds to the number of terminals. Via a terminal, a client tries to open a connection with the RTS. This connection is accepted if the maximum load of the RTS has not been reached yet, then it is authenticated. The maximum load is given by the initial marking of place *MaxReq*. The authentication is performed within subnet N_2. Variable x associated with transition *AuthOk*, memorizes the *user class* of the client.

Once authenticated, a client asks for a service that can be *non-critical* (e.g. a read transaction) or *critical* (e.g. a write transaction). Non-critical services can be simultaneously handled (inside subnet N_3) while a critical service must be performed in mutual exclusion with any other service. The system ensures a weak priority for non-critical services based on a *wave* mechanism. The wave consists of the clients currently accepted by the RTS. Once a client chooses a critical service (transition *ChCs*) accepted by the RTS (transition *AccCs*), no more clients can join the wave (inhibitor arc from place *Wave* to transition *AccR*). Critical services are performed only when there are no more clients in the authentication stage and

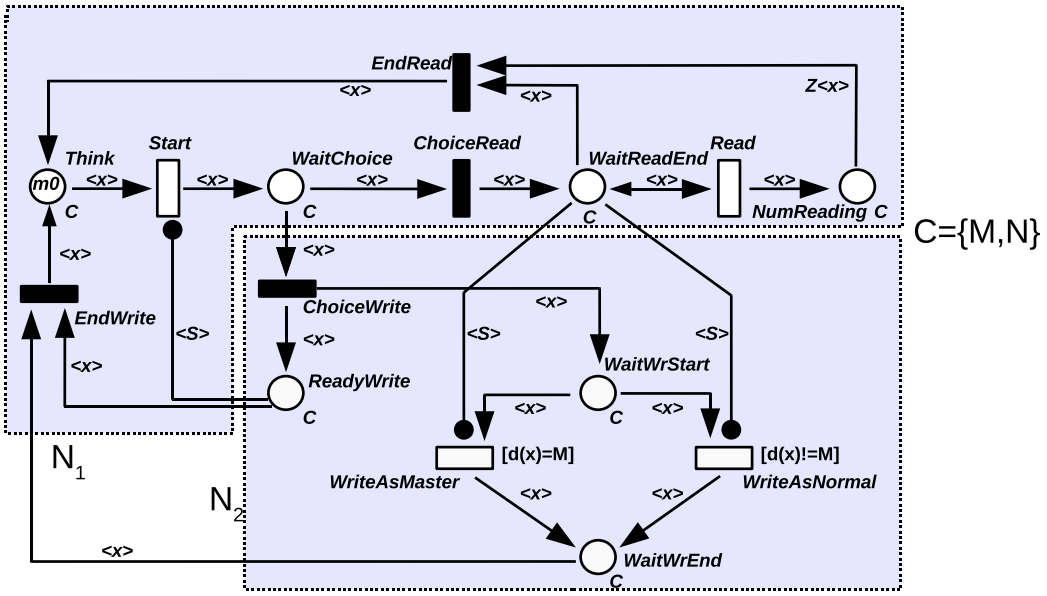

Figure 1: SWN model implementing the Reader-Writer pattern.

	SRG	ESRG		RESRG (Exact)			RESRG (Strong)			DSRG	
Z, |M|, |N|	St.	St. (Esm.+Ev.)	Peak	St.	Peak	ratio	St.	Peak	ratio	St.	ratio
6,1,3	1.535	491+3	141	492	492	3,12	491	491	3,12	491	3,12
6,1,6	21.338	4.951+3	1.263	4.952	4.952	4,30	4.951	4.951	4,30	4.951	4,30
6,1,8	75.968	15.731+3	3.625	15.732	15.732	4,82	15.731	15.731	4,82	5.732	4,82
6,1,10	216.218	41.407+3	8.682	41.407	41.407	5,22	41.406	41.406	5,22	41.406	5,22
6,2,6	97.358	9.076+3	5.387	9.077	9.077	10,72	9.076	9.078	10,72	9.076	10,72
6,2,8	344.192	26.027+3	15.716	26.028	26.028	13,22	26.027	26.027	13,22	26.028	10,72
6,2,10	974.976	63.701+3	37.896	63.702	63.702	15,30	63.701	63.701	15,30	63.701	15,30
6,3,6	321.638	15.731+4	16.641	15.732	15.732	20,44	15.731	15.731	20,44	15.731	20,44
6,3,8	1.131.671	41.406+4	49.369	41.407	41.407	22,92	41.406	41.406	22,92	41.406	22,92
6,3,10	3.195.194	95.201+4	120.085	95.202	95.202	33,56	95.201	95.201	33,56	95.202	33,56
8,1,6	77.816	16732+3	3.667	16.733	16.733	4,65	16.732	16.733	4.65	16.732	4,65
10,1,6	228.230	46.476+3	8.985	46.477	46.477	4,91	46.476	46.476	4,91	46.476	4,91
12,1,6	574.394	112.269+3	19.427	112.270	112.270	5,11	112.269	112.269	5,11	112.269	5,11

Table 1: Results for the Reader-Writer pattern in Fig. 1.

in a non-critical service execution. Place *NbReq* controls this requirement. When the last critical service of the wave completes, a new wave can start.

Subnet N_3 models the handling of a non-critical service. A service identity (variable i) is attached to the two parallel tasks that perform the service in order for them to correctly synchronize at the end (transition *eNCs*).

For efficiency reasons, during a wave the RTS accepts a limited number of different concurrent user classes (initial marking of place *MaxQueues*) in the critical services. This management is modeled by subnet N_4. A critical service request related to a user class not yet in competition (i.e., without its color in place *Queues*) is rejected (transition *Rej*) if the maximum number of concurrent user classes has been already reached.

A critical service is divided into two sequential stages: a preprocessing step that can be performed concurrently and a main step that is performed in mutual execution (see subnet N_6). If a priority rule is applied then the requests access the critical section following the order of the user classes. Observe that in this case the first critical service that has achieved its preprocessing step must wait if it does not belong to the highest priority user class in competition (see subnet N_5).

The priority among user classes is managed in N_5 using a *swap* mechanism based on the asymmetric *guarded* ($[y < x]$) transition *Swap*. It ensures that place *Elected* ends up by containing the highest user class of the remaining critical services requests. In order to guarantee that *Swap* is always performed before allowing the next critical section entry, transition *Swap* is given the highest priority (this is denoted $prio_3$).

Observe that this pattern can be interpreted as a refinement of the previous pattern when we consider the critical services as write processes and the non-critical as read processes. First the synchronization step occurs at the end of a wave which includes multiple critical services. Furthermore the overlapping of the asymmetrical part of the behavior and the symmetrical one is more important since the preprocessing step of critical service is symmetrical whereas the entrance in the critical section is asymmetrically managed. The propagation of asymmetry depends on two factors: the maximal load of the RTS and the maximum number of allowed simultaneous user classes.

Let us examine in more details table 2. The first column shows the experiment parameters: *Local*, *GC*, *LC*, *Prio*.

- *Local*: representing the numbers of terminals, would affect only the "front-end" behavior of the system (outside the server). The increase of the number of terminals does not affect the internal activities of the server
- $GC(\leq Local)$: representing the maximum number of users allowed to simultaneously access to the server. This parameter has crucial impact on the global behavior of the server. Actually, its value affects the symmetric part as well as the asymmetric one.
- *Prio*: represents the cardinality of the color class C. Since there is a bijection between this cardinality and the number of different priorities, we observe a strong dependency between the value of this parameter and the efficiency of the different approaches.
- $LC(\leq Prio)$: the maximum number of user classes allowed to access, simultaneously, the critical part. The value of this parameter will affect, essentially, the behavior of the asymmetric part (with some side effect on the symmetric part).

The other columns have the same meaning of the columns in the previous table.

We notice here the significant reduction achieved by our approaches w.r.t. the SRG one. For instance, in case 8,5,3,5 the ESRG reduction factor (both strong and exact) is $45,53$, while the DSRG reduction is

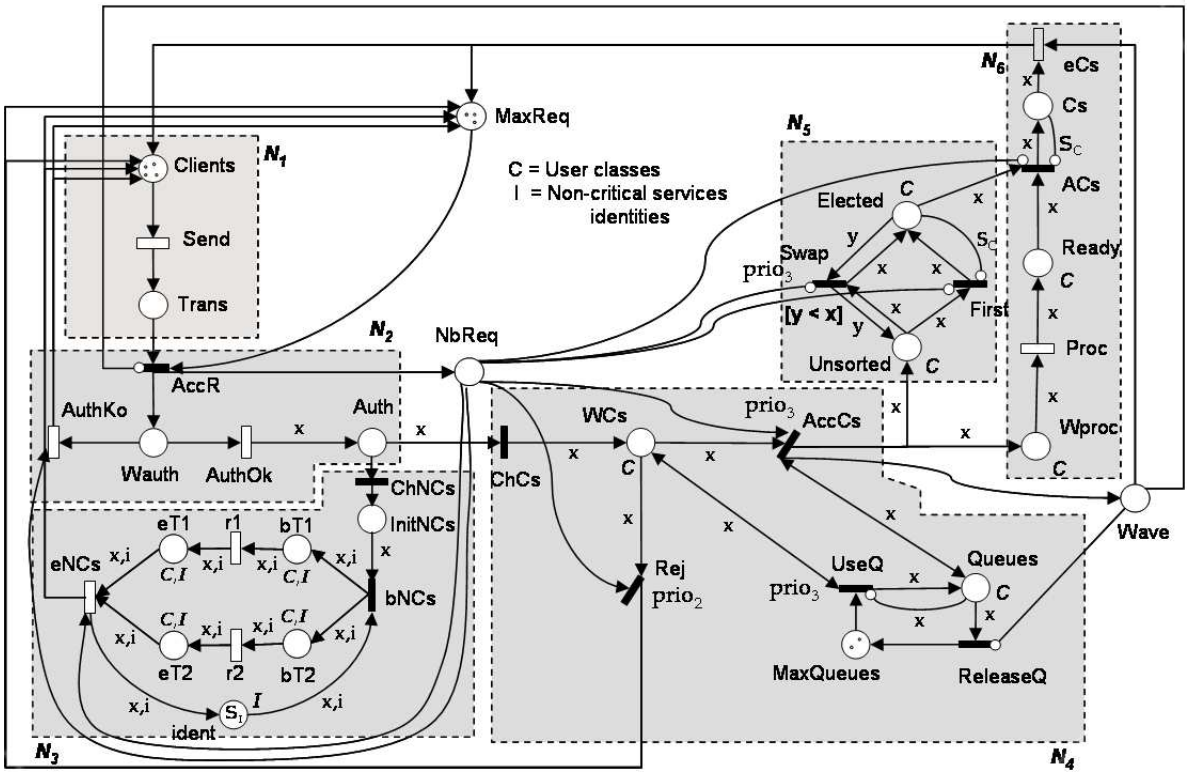

Figure 2: SWN model implements the Clients-Server pattern with critical section.

	SRG	ESRG		RESRG (Exact)			RESRG (Strong)			DSRG	
Local,GC LC,Prio	St.	St. (Esm.+Ev.)	Peak	St.	Peak	ratio	St.	Peak	ratio	St.	ratio
3,3,2,5	19.108	981+470	4.050	998	1.727	19,14	998	1.727	19,14	1.217	15,70
3,3,2,8	72.772	981+1.316	17.028	998	3.185	72,91	998	3.185	72,91	1.436	50,67
3,3,3,3	4.778	1.028+179	846	1.064	1.128	4,49	1.060	1.228	4,49	1.142	4,18
3,3,3,5	19.918	1.028+850	4.080	1.064	2.326	18,71	1.060	2.326	18,71	1.613	12,34
3,3,3,8	77.308	1.028+3.444	17.196	1.064	6.498	72,93	1.060	6.498	72,63	2.765	27,95
8,5,3,3	496.618	90.429+4.522	59.187	92.600	96.088	5,36	92.224	96.088	5,36	94.593	5,25
8,5,3,5	4.788.499	108.205+28.040	691.390	110.376	159.104	45,53	110.000	159.104	45,53	134.553	35,58
8,5,3,8	-	out of memory	-	-	-	-	-	-	-	193.401	-

Table 2: Results for the clients-server pattern

27, 95. Moreover, we remark that in this model the computation peaks of the two approaches based on the ESRG are important. For instance, for 8,5,3,8 the ESRG cannot be computed due to its computation peak. Hence the final aggregation obtained by the two approaches based on the ESRG is better, but the number of real ESMs and eventualities stored during the computation is greater than the number of states of the DSRG.

3.3 A workflow pattern

A workflow is a set of tasks organized according to a model that describes the triggering conditions for every task. It is usually described by operators like sequential flow, parallel flow, choice flow, etc. that are easily modeled by an ordinary Petri net. In addition, with every task is associated a set of agents which are allowed to perform it. A job is an instance of a workflow and we consider simultaneous executions of jobs with the same workflow.

Our experiment is based on the fixed control flow shown in the SWN model in Fig. 3. The total number of tasks in the system is given by the initial marking of place *Idle* while the total number of agents for each type is given by the initial marking of colored place *Agents*. Moreover, the initial marking of places *T1*, *T2* and *T3* represents the maximum number of simultaneous *accepted tasks* for each task type, while the initial marking of places *ExcT1*, *ExcT2* and *ExcT3* represents the maximum number of simultaneous *executions* for each task type.

The asymmetry is due to the set of agents which are divided in groups $\{G_i\}_{i\in\{1,2,3\}}$, according to their authorization levels: an agent $g \in G_i$ has less authorizations than an agent $g' \in G_j$ with $j > i$. We specify for every task T an execution threshold $i(T)$: an agent in G_i is allowed to perform task T if $i \geq i(T)$.

The execution cost of a task depends both on the execution time and on its execution threshold. Thus the WFN model enforces a policy that aims at minimizing the execution time of special tasks with an high execution threshold. So with every execution threshold we associate a maximum number of simultaneous executions. Then, when a task is triggered, it is queued and later on it is executed. Most of the tasks can be executed by every agent, we call it *normal tasks*; the other ones are called *special tasks*. This is granted by the guards on the transitions *StartTask2* and *StartTask3*, so that *StartTask2* can fire if there is at least one available agent with $G_i \geq 2$, while *StartTask3* if there is at least one available agent with $G_i = 3$.

A task is executed (transition *StartTask1* or *StartTask2* or *StartTask3*) if the following conditions are fulfilled:

- there is no running or waiting special task with higher execution threshold. This is ensured by the test arcs with multiplicity k connecting T_j with $StartTask_i$ and $j > i$.
- there are only running tasks with the same execution threshold. This is ensured by the test arcs with multiplicity a_i connecting $ExcT_i$ with $StartTask_j$ and $j > i$.
- the maximum number of simultaneous executions corresponding to its execution threshold is not reached. This is ensured by the arcs connecting $ExcT_i$ with $StartTask_i$.

Finally the completion of a task i is modeled by transition $EndTask_i$.

The symmetrical behavior corresponds to a wave of executing normal tasks (submodel N_1) whereas the asymmetrical behavior corresponds to a wave of special task executions with the same execution threshold (submodel N_2). The synchronization steps occur after every wave execution. Observe here that between two synchronization steps there is either a symmetrical behavior or an asymmetrical one but not both.

Let us examine in more details Table 3, that shows some experiments performed on this model for different values of its parameters: the number of total tasks (K), the maximum number of simultaneous executions

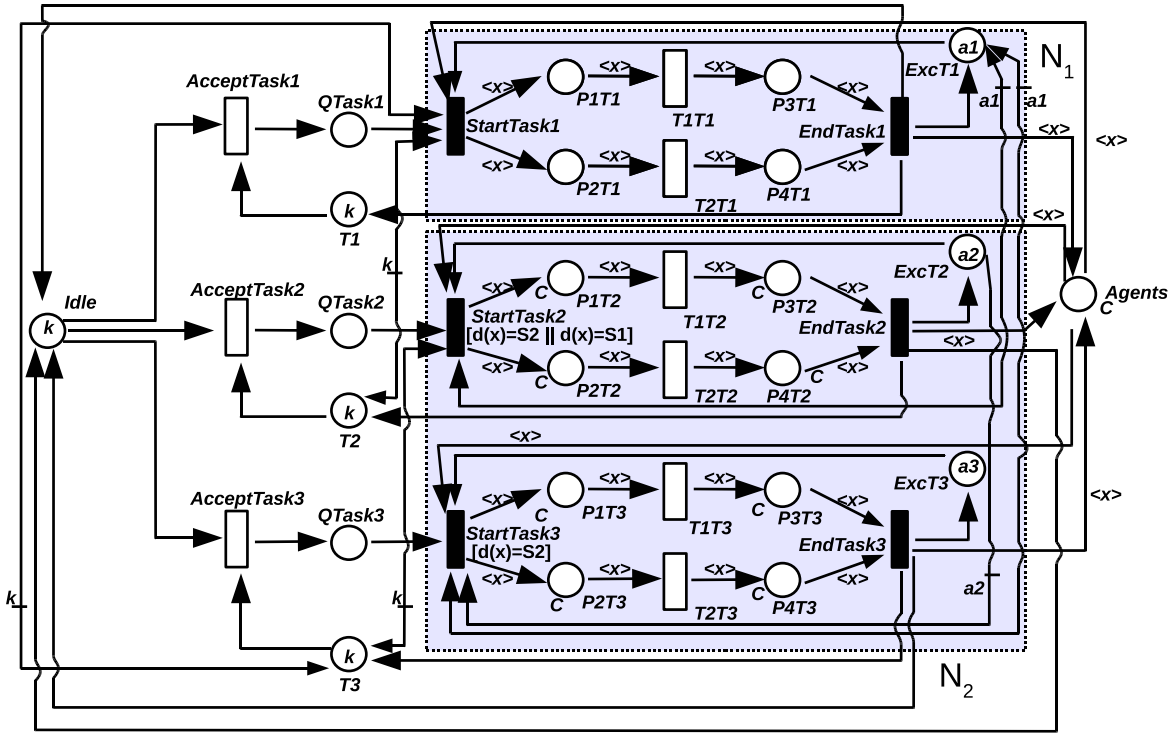

Figure 3: SWN of the Workflow pattern.

	SRG	ESRG		RESRG (Exact)			RESRG (Strong)			DSRG	
K,a1,a2,a3, $\|G_1\|$,$\|G_2\|+\|G_3\|$	St.	St. (Esm.+Ev.)	Peak	St.	Peak	ratio	St.	Peak	ratio	St.	ratio
5-4-2-1-2-2	3.550	1.003+274	682	1.003	1.244	3,53	1.003	1.244	3,53	1.003	3,53
10-4-2-1-2-2	40.060	9.643+2.199	5.447	9.643	11.444	4,15	9.643	11.444	4.15	9.643	4,15
15-4-2-1-2-2	151.045	34.933+6.614	17.417	34.933	40.869	4,32	34.933	40.869	4,32	34.933	4,32
5-4-3-2-2-4	7.994	1.429+1044	1.878	2.437	2.440	3,28	1.429	2.440	5,59	1.429	5,59
10-4-3-2-2-4	108.789	13.889+10.504	17.708	24.147	24,175	4,50	13.889	24,175	7,82	13.899	7,82
15-4-3-2-2-4	431.134	50.399+38.289	63.413	87.932	88.010	4,90	50.399	88.010	8,55	50.399	8,55
15-4-2-1-3-2	200.467	34.933+6.614	23.483	34.933	40.869	5,73	34.933	40.869	5,73	34.933	5,73
15-6-2-1-4-2	383.108	55.497+6.814	40.776	55.497	61.433	6,90	55.497	61.433	6,90	55.497	6,90
15-8-2-1-6-2	655.095	74.951+7.023	62.540	74.951	80.887	8,74	74.951	80.887	8,74	80.887	8,74
15-10-2-1-8-2	885.630	89.075+7.467	70.031	89.075	95.011	9,94	89.075	95.011	9,94	95.011	9,94
15-12-2-1-10-2	1.026.425	96.401+9.453	71.517	96.401	102.337	10,64	96.401	102.337	10,64	102.337	10,64
15-15-2-1-13-2	1.086.911	98.556+10.345	71.573	98.556	104.492	11,02	98.556	104.492	11,02	98.556	11,02

Table 3: Results for the workflow pattern

for each threshold (a_i), the number of normal agents ($|G_1|$) and the number of special agents ($|G_2| + |G_3|$). We observe that both these methods reach the same reduction factor; moreover the parameters that have more impact on the reduction factor are the number of total tasks and the maximum number of simultaneous executions for normal tasks. The reduction factor is increasing w.r.t to these two parameters.

3.4 A cluster computing pattern

This pattern corresponds to a finite set of machines grouped in clusters. Each cluster has *a master* machine and a set of *slave* machines. Every machine can fail while being idle or working. While idle, the failing of a slave machine means its removing from the cluster, for (local) updating/maintenance reasons. It is put back in its environment as soon as it is reconfigured. Instead, if it fails while working because of a hardware/software problem then its last stable state is saved. Afterwards it is repaired and finally restarted.

The failing of a master has a different consequence: the whole cluster is no longer reachable. Actually, even if the slave machines of the cluster are not in a fail state, the absence of a master makes the cluster in an unstable state. In such a case, the cluster becomes unavailable until the recovery of the master machine.

The system presents a symmetrical behavior until the first failure. However after this failure we cannot identify synchronization steps. Thus the results are poor as detailed in table 4.

The SWN model implementing the cluster computing pattern is shown in Fig. 4. It is divided in four submodels: N_1, N_2, N_3 and N_4. N_1 models the jobs submission to the computing system. A submitted job marks place *AJobs*. The corresponding token is moved to place *SJobs*, when a machine m is assigned to its execution (firing of transition *AssMach*).

N_2, models the machine states and transitions between them: available machines are in place *AvMach*, failed machines in place *FMach*, and frozen machines in place *Freeze*. Initially, all machines are available. When a machine fails (transition *FailedIdleMach*), it is moved to *FMach*. If it is a master machine, then all slaves of its cluster are also moved to *Freeze* (transition *RemMach*). Once the master is recovered (transition *Recover*), its frozen slave machines are put back in *AvMach*.

Subnet N_3 represents the behavior after a normal exit: the firing of transition *EndWork* models the end of a submitted job without failure.

Subnet N_4 models the behavior in case of failing while working: if it is a slave machine then it is

	SRG	ESRG		RESRG (Exact)			RESRG (Strong)			DSRG	
|Job|, |Cl|, |M|	St.	St. (Esm.+Ev.)	Peak	St.	Peak	ratio	St.	Peak	ratio	St.	ratio
2,2,2,2	1.803	220+1.601	1.634	897	1.803	2,10	625	1.795	2,84	2.008	0,9
5,2,2,3	3.776	408+3.318	3.352	1.790	3.776	2,10	1.254	3.703	3,01	3.646	1,03
5,2,2,4	6.295	641+5.468	5.330	2.790	6.295	2,25	1.861	6.080	3,38	5.771	1,09
5,2,2.5	9.002	900 +7.642	7.202	3.712	9.002	2,42	2.474	8.507	3,63	7.565	1,18
5,2,5,2	15.369	910+14.457	15.198	7.625	15.369	2,01	6.752	15.350	2,27	15.343	1
5,2,5,3	35.246	1.779+33.415	34.788	17.293	35.246	2,03	13.117	35.171	2,68	35.037	1
5,2,5,4	66.413	2.971+63.226	65.162	32.217	66.413	2,06	20.602	66.168	3,22	65.575	1,01
5,2,5,5	109.118	4.483+103.947	105.744	52.039	109.118	2,09	35.032	108.486	3,11	106.374	1,02
16,1,8,2	734	459+327	575	733	741	1	707	730	1,03	731	1
16,2,4,2	8.797	627+8.168	8.626	8.615	8.797	1,02	4.106	8.778	2,14	9.105	0,96
16,1,16,2	2.418	1.547+971	2.124	2.409	2.433	1,00	2.381	2.414	1,01	2.415	1,00
16,2,8,2	52.713	2.077+50.634	52.542	51.919	52.713	1,00	22.406	52.694	2,35	53.010	0,99

Table 4: Results for the cluster computing pattern

repaired and restarted (transition *RestartSlave*). In case of a failed master machine, it is moved (transition *MasterFail*) to the fail state (place *FMach*), and all working machines of the same cluster are moved to a frozen state, by use of transitions *RetWorMach* and *RetEndMach*. In this last case, all served jobs of the cluster must be rerun. They are put back in place *AJobs*.

Several experiments have been performed on this SWN model for different values of the system parameters: the number of jobs ($|Job|$), the number of clusters ($|Cl|$) and the number of machines per cluster ($|M|$), but the best result obtained for the reduction factor is less than 3.

Observe that then number of ESM and eventualities stored during the generation of the ESRG and during the refinement steps is close to the $|SRG|$.

4 Conclusion

Four model patterns, namely readers-writers, client-server, workflow and cluster computing have been presented to show the ESRG and DSRG methods effectiveness and their applicative interest.

The experiment results show that in systems matching the a pattern proposed in section 2 the two approaches reach an high aggregation level. For instance, the first three case studies matching with the proposed application pattern lead to an high reduction factor, while in the last one the RESRG and the DSRG size is closed to the SRG one.

Moreover, we have to highlight that in our experiments the size of the lumped chain is generally smaller with the ESRG method; however the ESRG method requires to explicitly develop "asymmetrical" set of states during the refinement process thus facing the problem of a peak in memory usage, contrary to the DSRG method. For instance in the case 8,5,3,8 of the client-server pattern the ESRG cannot be computed due to its computation peak.

References

[1] S. Baarir, C. Dutheillet, S. Haddad, and J-M. Ilié. On the use of exact lumpability in partially symmetrical Well-formed Nets. In *Proc. of Int. Conf. on the Quantitative Evaluation of Systems (QEST)*, pages 23–32,

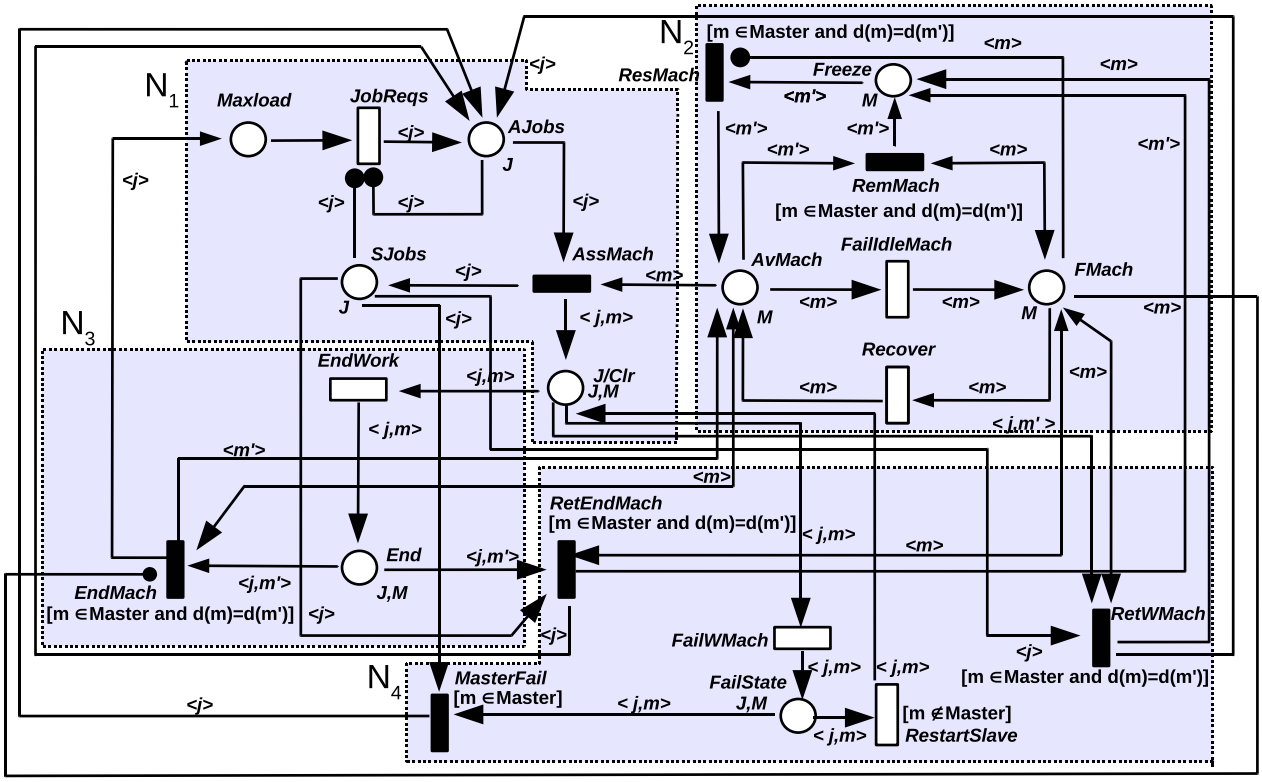

Figure 4: SWN implementing the cluster computing pattern.

Torino - Italy, September 2005. IEEE C.S. press.

[2] Soheib Baarir. *Exploitation des symétries partielles pour la vérification et l'évaluation de performances des systémes finis.* PhD thesis, Laboratoire d'Informatique de Paris 6 (LIP6), 2007.

[3] Marco Beccuti. *Modeling and analysis of probabilistic systems.Formalisms and efficient algorithms.* PhD thesis, Universitá degli Studi di Torino in "cooperation" with Universitè Paris Dauphine, 2008.

[4] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, Nov. 1993.

[5] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured Petri nets. *Theoretical Computer Science*, 176(1-2):39–65, 1997.

[6] M. Beccuti, S. Baarir, G. Franceschinis, and J-M. Iliè. Efficient lumpability check in partially symmetric systems. In *QEST*, pages 211–220, Riverside, CA, USA, September 2006. IEEE Computer Society.

[7] S.Haddad, J.M Ilié, M. Taghelit, and B. Zouari. Symbolic Reachability Graph and Partial Symmetries. *Proc. of Int. Conf. on Application and Theory of Petri Nets. Turin, Italy.*, 935:238–251, June 1995.

A A brief introductin on to Stochastich Well-formed Net

The Well-formed Net (WN) formalism [4] was inspired by the Colored Petri Net (CPN) formalism and has the same modeling power of CPNs (but unlike CPNs it includes transition priorities and inhibitor arcs). However its color annotations syntax is peculiar: it was defined with the aim of developing efficient analysis techniques able to automatically exploit the behavioral symmetries embedded in the model.

Definition 1 *A Well-formed Net is a ten-tuple*

$$\mathcal{N} = \langle P, T, \mathcal{C}, cd, I, O, H, \phi, prio, \omega, m_0 \rangle$$

P and T are the place and transition [1] sets; transition input, output and inhibitor arcs are defined by I, O, H, which define also their color annotations (called arc expressions); m_0 is the initial (colored) marking. *prio* defines the transition priorities (assigning a priority level $prio(t) \in \mathbb{N}$ to each transition). The other elements correspond to the model color annotations, described next.

Basic colour classes. $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a set of pairwise disjoint *basic colour classes.* C_i is a finite not empty set and it can be partitioned into n_i disjoint subsets $C_{i,j}, j = 1, \ldots, n_i$ called *static subclasses*; $|C_i|$ denotes the cardinality n_i of the partition. A basic colour class C_i may be (circularly) ordered, the order is induced by a successor function: the successor of element c is denoted $!c$.

Color domain. cd defines the color domains of places and transitions, which are Cartesian products of basic colour classes. The colour domain of a node k is denoted $cd(k) = C_1^{e_1} \times C_2^{e_2} \times \ldots \times C_n^{e_n}$, where $e_i \in \mathbb{N}$ is the number of occurrences of class C_i in $cd(k)$ and its value depends on the considered node.

Elementary Function. I, O and H are defined on $T \times P$ and can map a pair t, p to an empty function, meaning that there is no input, output or inhibitor arc connecting t and p, or to an arc function $f : cd(t) \rightarrow Bag(cd(p))$, defined next.

Let us consider the colour domain $cd = C_1^{e_1} \times C_2^{e_2} \times \ldots \times C_n^{e_n}$: an *elementary colour function* is a linear mapping from cd to $Bag(C_i)$ (for some $i \in 1, \ldots, n$) chosen among the following functions:

[1] T can be partitioned into two subsets of timed and immediate transitions

- the projection denoted X_i^l defined as: $X_i^l(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n}, \ldots) \mapsto c_i^l$
- the successor denoted $!X_i^l$ defined as: $!X_i^l(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n} \ldots) \mapsto !c_i^l$
- the diffusion function (also called synchronization function, depending if it annotates an output or an input arc), which is constant, denoted S_i and defined as follows: $S_i(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n} \ldots) \mapsto \sum_{\forall c \in C_i} c$

Notice that in practice the symbols X_i^l used above to denote the projection function are substituted by names of transition variables (representing the transition parameters) in the models; each variable has a type C_i. The variable-based notation makes the model more readable, since variables can be given meaningful names. The diffusion(synchronization) function can be restricted to a static subclass, denoted $S_{i,j}$ and defined as follows: $S_{i,l}(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n} \ldots) \mapsto \sum_{\forall c \in C_{i,l}} c$

Due to the linear property of the the elementary functions, they can be just defined on the single elements of the domain. Abusing notation, the elementary functions and their linear extension are usually denoted in the same way.

Class function. A colour function f on class C_i, also called C_i class-function, is a linear combination of elementary functions (with same domain and codomain):

$$f_i = \sum_j \alpha_j \cdot X_i^j + \sum_{q=1}^{|C_i|} \beta_q \cdot S_{i,q} + \sum_j \gamma_j \cdot !X_i^j$$

The coefficients $\beta_q \in \mathbb{N}$, $\alpha_j, \gamma_j \in \mathbb{Z}$ must satisfy the following constraint: if f_i^- and f_i^+ are respectively the multisets of elements with negative and positive coefficients in the formula above (so that $f_i = f_i^+ - f_i^-$), then it must hold $f_i^- \subseteq f_i^+$.

Arc function. An arc function F on an input, output or inhibitor arc, connecting transition t and place p, is a sum so defined:

$$F \;=\; \sum_k \lambda_k \cdot \bigotimes_{i=1}^{n} \bigotimes_{j=1}^{e_i} f_i^{j,k}$$

where $f_{i,j}^k : cd(t) \to Bag(C_i)$, $\lambda_k \in \mathbb{N}$ and e_i is the number of occurrences of class C_i in $cd(p)$. The symbol $\bigotimes$ denotes the Cartesian product quantifier, in the text we shall also use the alternative representation $\langle f_1^1, f_1^2, \ldots f_n^{e_n'} \rangle$, briefly called *function tuple* (or simply *tuple*).

Transition and colour function guards. A *guard* is a Boolean expression defined on a transition colour domain whose basic terms are *standard predicates*. Standard predicates allow to compare colour elements from the same colour class C_i, and can take the following form:
- $[X_i^j = X_i^k](c)$, it evaluates to *true* iff the j^{th} component of type C_i in c is equal to the k^{th} component of same type;
- $[d(X_i^j) = C_{i,h})](c)$, it evaluates to *true* iff the j^{th} component of type C_i in c belongs to $C_{i,h}$;
- $[d(X_i^j) = d(X_i^k)](c)$, it evaluates to *true* iff the j^{th} and k^{th} components of type C_i in c belong to the same static subclass.

In WNs the ϕ function associates a guard with each transition, the default guard is $[true]$. Moreover, guards may be employed in arc expressions. Let F be a function tuple and g a guard, then the guarded tuple is defined as

$$F\,[g] \stackrel{def}{=} \text{ if } g(c) \text{ then } F(c) \text{ else } \emptyset$$

A guarded arc function is then a linear combination of guarded tuples.

The transition instance weights ω. The fuction ω is defined as a set of functions $\omega_t : cd(t) \rightarrow \mathbb{R}$ expressed in the following form:

$\omega(t)=$
{
case $cond_1 : r_1$
case $cond_2 : r_2$
...
case $cond_n : r_n$
default: $r_{default}$
}

where $cond_i$ is a boolean expression comprising standard predicates on the transition color instance and predicates on the marking (defined in terms of static subclasses) so that the firing rate of a transition instance can depend only on the static subclasses of the objects assigned to the transition parameters, and not on the assigned objects themselves.

Definition 2 (Marking) *A marking m is expressed as a distribution of colored tokens in the all places.*

Definition 3 (Transition instance) *A transition instance $\langle t, c \rangle$ in m is a binding c of the transition variables to the objects in the appropriate color class.*

The evolution of an WN system is defined through a firing rule applied to a given transition instance $\langle t, c \rangle$. The new marking obtained by the transition instance firing satisfies: $\forall p \in P, m'(p) = m(p) - I(p,t)[c] + O(p,t)[c]$.