Exploiting multiformalism models for testing and performance evaluation in SIMTHESys

E.Barbierato
Dipartimento di Informatica, Università degli Studi di Torino, Italy
enrico.barbierato@mfn.unipmn.it

M.Gribaudo
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
gribaudo@elet.polimi.it

M.Iacono
Dipartimento di studi Europei e Mediterranei, Seconda Università degli Studi di Napoli, Italy
mauro.iacono@unina2.it

ABSTRACT

SIMTHESys is a framework for the design of multiformalism performance evaluation models. The modeler can create new formalisms by specifying both the syntax and the dynamic behavior of their atomic elements. Even if other approaches address the same issue, the proposed methodology relies on fewer assumptions, opening new possibilities that allow to consider new types of composition and interaction between formalisms.

In this direction, this paper shows how four formalisms belonging to three different classes can interact together in a single environment. The multiformalism proposed is composed of two standard performance evaluation formalisms, a reliability formalism and a verification formalism. The potential of this approach is demonstrated by analyzing a model of an e-government process.

Keywords

Multi-formalism modeling, Tools for performance evaluation.

1. INTRODUCTION

Multiformalism is a well established approach to modeling complex systems: Mobius [22], Sharpe [24], SMART [5], OsMoSys [26] are some examples of frameworks and methodologies supporting it. The use of multiformalism modeling techniques is based on the integration of components defined using several different modeling languages (formalisms). The peculiarities of this approach are twofold: from the modeler's point of view, allowing different subsystems to be modeled by different formalisms favors the choice of a more familiar or more appropriate language, lowering the learning curve or matching the user ideal abstraction; from the point of view of the solution, choosing the right combination of formalisms means a proper mapping of model concepts onto solvers primitives, that better fits the problem.

VALUETOOLS 2011, May 16-20, Paris, France

DOI 10.4108/icst.valuetools.2011.245727

This paper shows how multiformalism can be exploited to enrich modeling when combining performance evaluation with model verification techniques. The following combination of formalisms belonging to three different classes is considered:

1. *Performance evaluation*, considered by supporting both Stochastic Petri Nets (SPN) [16] and Finite Capacity Queueing Networks; (TFCQN) [13];
2. *Reliability* through the use of Fault Trees [21]
3. *Verification* by means of an ad-hoc extension of Finite State Automata [2].

The paper is organized as follows: following a works overview about multiformalisms in Section 2, SIMTHESys is briefly introduced in Section 3; the proposed multiformalism is presented in Section 4 and solving engines are described in Section 5. Section 6 shows an application of the approach to the evaluation of an e-government system; finally, some conclusions are drawn in Section 7.

2. RELATED WORKS

Multiformalism approaches in performance and performability evaluation are widely witnessed by the existence of frameworks like Sharpe [22] [24], SMART [5] and the DEDS toolbox [3], or Mobius [6] [9] [23] [7] [8] and OsMoSys [11] [26] [18] [25] [12], the last two being the most similar experiences with respect to SIMTHESys [15]. In this field both the concepts of multiformalism and multisolution are defined: multiformalism is the ability of a framework or a methodology to support the definition and the analysis of models that are built by simultaneously using different formal languages, while multisolution refers to the integration of existing evaluation tools (synthesized dedicated softwares, composable solution engines or stand alone solutions) to obtain relevant indexes on a complex model.

Multiformalism is a consolidated practice also in other fields of modeling: for example in verification research, a branch named 'integrated formal methods' deals with the coupling of software formal analysis techniques to exploit the benefits of combinations. Z variants and process algebras combination are given in [10], [17], [19] and extensions to include other formalisms are presented in [14], [20]. Some more examples of combinations are provided in [4], that also suggests many other useful references on the topic. In the example used in this paper we will couple performance oriented

formal methods with a verification automata-like language, that is newly designed though is inspired to [2], [1], and a reliability formalism based on Repairable Fault Trees [21].

3. SIMTHESYS OVERVIEW

In [15] the SIMTHESys approach is introduced[1].
SIMTHESys is a novel framework for the definition and solution of multiformalism models based on the generation of complex solvers, that are automatically obtained by combining general solution engines, according to the rules resulting from formalisms definitions. Formalisms are defined by the explicit specification of both syntax and semantics of all their atomic components.

The key advantages of this approach can be described as the rapid prototyping of new formalisms and solution techniques, the capability of deploying new solvers without modifying the existing ones and finally its open architecture, i.e. the possibility to define new interfaces that can be used to characterize different classes of formalisms.

In the current example we have considered formalisms that describe systems characterized by exponential transitions. This has been accomplished by defining appropriate interfaces that must be implemented by both the class of given models and related solvers. It is possible to implement other kind of formalisms such as non-Markovian Fault Trees or Bayesian Networks by defining other types of interfaces which consider probabilities and distributions.

In SIMTHESys a *model* is a description of a system written according to a *formalism*. Models are written to be evaluated by some point of view that depends on the formalism. A formalism is a formal language defined in terms of its *elements*, that form its grammar. For each element a set of attributes (*properties*) and a set of method-like constructs (*behaviors*) are given. Each property is either a constant, a state information (used to define the state structure of the model) or a result (computed during the solution). The set of behaviors of an element describes its semantics (its dynamics or execution policy). In every formalism a special *container element* is defined to represent a (sub)model written in that language. This special element contains global properties and behaviors that identify the entire (sub)model.

Special formalisms, defined as *composition formalisms* are used to interface primitives belonging to different languages. Elements of composition formalisms are designed to semantically connect concepts belonging to different types of models. Such elements are said *hybrid elements* and exploit their own behaviors as a handle to implement the interaction logic. Formalisms and models are described by XML formats, called respectively the "Formalism Description Language" (FDL), and the "Model Description Language" (MDL).

4. DEFINING THE FORMALISMS

The purpose of this work is to propose an approach to test a multiformalism performance oriented model against some conditions. This guarantees that the model is correct from the designer's point of view. The proposed approach should not affect the formalisms chosen to represent the original model and should be based on an additional testing formalism capable to interact with the others. The solver for the resulting formalism composition should be either able to measure performances if an unwanted condition is not met, or to compute specific indices that characterize the unwanted condition otherwise.

Two well defined performance evaluation formalisms, namely SPN and FCQN, and their integration in the SIMTHESys approach were already considered in [15]. In Section 4.1 a brief introduction to the implementation of both formalisms is given. Then two new formalisms are introduced: Fault Trees (FT) for the specification of reliability performance indices (in Section 4.2), and a Testing Formalism (TF) for the verification of the model behavior (in Section 4.3).

Note that though the verification can also be done by using an appropriate SPN model, and FT can be emulated using SPN as described in [21], the use of different formalisms allows a more compact representation, producing models that are easier to read and to maintain.

4.1 Performance evaluation oriented formalisms

SPN is a class of Petri Nets in which firing of enabled transitions consumes a time interval that is described by an exponentially distributed random variable. Since the firing time is exponentially distributed, it can be characterized by a single parameter: the *rate* of the distribution.

Table 1 describes SPN formalism elements[2].

The SPN element uses the behavior `InitEvents` to update the state of the model by checking which of the transitions are enabled, and by scheduling the firing of the corresponding events at the given rate. The behaviors `ComputeStateRewards`, `CountStateRewards`, `SetStateRewards`, `ListImpulseRewards` and `SetImpulseRewards` are used to compute performance indices.

For the sake of brevity, only one behavior is described in Algorithm 1. The `Fire` behavior of the Transition element updates the state of the system when the transition actually fires.

Algorithm 1 `Fire`

```
1: for all a ∈ Arc where a.from = this do
2:     a.Push();
3: end for
4: for all a ∈ Arc where a.to = this do
5:     a.Pull();
6: end for
```

The second performance oriented formalism is FCQN. It is suitable for the analysis of systems in which a number of servers are connected to satisfy requests of customers. Service stations are provided with queues and characterized by the distribution of their service times. Queues are connected in a network, so that a customer that is served in a queue is delivered to another queue. In this work we consider only

[1]See also http://www.dem.unina2.it/simthesys.

[2]The complete FDL description of SPN is available at www.dem.unina2.it/simthesys.

Element	Property	Type	Modifier	Behaviors
SPN	bounded	boolean	computed	InitEvents, ComputeStateRewards, CountStateRewards, SetStateRewards, ListImpulseRewards, SetImpulseReward
Arc	weight from to	integer element element	const const const	IsActive, Push, Pull
Inhibitor Arc	weight from to	integer element element	const const const	IsActive
Place	marking meantokens	integer float	state computed	GetOccupancy, AddOccupancy
Transition	rate throughput	float float	const computed	IsActive, Fire

Table 1: Elements of the SPN SIMTHESys definition

queues with exponentially distributed service times. Moreover, in FCQN every queue has a finite *capacity*, with a maximum number M of waiting customers allowed. In case there is not room for new customers, a queue blocks the server until it is able to receive a customer again (blocking hypothesis). In this work we consider only the Blocking Before Service policy (BBS, the blocked queue processes the customer after the end of the block).
FCQN with BbsSoQueue discipline are defined in SIMTHESys using the elements, properties and behaviors presented in table 2[3].

4.2 Reliability formalism

FT is used as a reliability formalism. A FT describes how a complex fault (Top Event, TE) is generated by the combination of elementary faults (Basic Events, BE) by representing the relationship that components have in a system. In this paper we will implement a repairable variant of FT, in which a BE is a repairable event, to enable the TE of a FT to dynamically evaluate the overall condition of a system. The elements of the FT formalism are presented in table 3[4]. A BE is described by a BasicEvent element. Each BasicEvent has associated two properties that specify an exponential failure time (*break_rate*), and an exponential repair time (*repair_rate*, which can be set to 0 if no repair is allowed). BasicEvents also have a *broken* state property that checks whether a component has failed. Fails propagates up to the TopEvent using connecting arcs (specified by the Arc elements), and can be combined using And, Or and Not elements. All these elements implement a *IsTrue* behavior that is used to check whether the sub-tree that is connected to them describes a faulty situation in a given moment. A FT is here enabled to use an element of a SPN or a FCQN in place of a BE by exploiting SIMTHESys multiformalism characteristics.

4.3 Verification formalism

The verification formalism is named *Test* and it is composed of the elements presented in Table 4[5].

[3]The complete FDL description of FCQN is available at www.dem.unina2.it/simthesys.
[4]The complete FDL description of FT is available at www.dem.unina2.it/simthesys.
[5]The complete FDL description of Test is available at www.dem.unina2.it/simthesys.

This formalism describes a sequence of conditions that should occur in order to progress to a certain state by using an automaton-like style. The state represents a situation where the modeler wants to check the probability occurrence. Such a situation is described by the Sat elements in the formalism; the sequence of conditions is represented by a sequence of State elements, that are connected by Next elements and Check elements. Next elements are interconnection arcs that join one State element to one Check element and viceversa. Check elements act as guards on the model, and cause state changes in the automaton when a given condition is satisfied. Conditions can be verified over elements of other formalisms, by exploiting SIMTHESys multiformalism characteristics: here the state of the TE of a FT, the marking of a place in a SPN and the length of a queue in a FCQN will be used. The Test element of the Test formalism has a state property *currentState* (that is preset to the initial state of the automaton), as summarized in Table 4. This property is used to track the state of the Test formalism. As seen for the other formalisms, Test implements `InitEvents` (to initialize the model) and five reward behaviors. It implements `ExpEventModel`, to use the exponential events solving engine), defines the *PerformTest* behavior to offer to the solvers a way to invoke the check of the model.

The Test formalism evolution depends on the *GetNext* behavior of the State element, invoked by the solver. The behavior uses the *IsTrue* behavior of the Next arcs to verify if some of the conditions (specified by the connections of the Check elements) are verified. If this is true, the formalism uses the *GetDestination* behavior to update the current state. Note that we require that only one check can be valid at a given time. If more than one check is valid, the Test formalism will evolve following the specification associated to the arc that came first in the XML file.

4.4 Bridging the three worlds

A composition formalism Tester is used to allow the interaction between the verification, the reliability and the performance oriented submodels. This multiformalism introduces five elements that can be grouped into three categories, according to the formalisms they interact with. The semantics of the first category (PN place -> FCQN queue) is analogous to the semantics of a SPN arc. When a place is marked with a number of tokens equal to the weight of the arc, the

Element	Property	Type	Modifier	Behaviors
TFCQN				InitEvents, ComputeStateRewards, CountStateRewards, SetStateRewards, ListImpulseRewards, SetImpulseReward
Arc	from to weight	element element float	const const const	HasSpace, Push IsActive, Pull
BbsSoQueue	length meanlength capacity rate throughput	integer float integer float float	status computed const const computed	IsActive, AddOccupancy, Fire, CanSend, CanAccept

Table 2: Elements of the TFCQN SIMTHESys definition

Element	Property	Type	Modifier	Behaviors
FT				InitEvents, ComputeStateRewards, CountStateRewards, SetStateRewards, ListImpulseRewards, SetImpulseReward
TopEvent	prob	float	computed	IsTrue
BasicEvent	prob broken break_rate repair_rate	float boolean float float	computed state const const	IsTrue, Fire, GetRate
Or				isTrue
And				isTrue
Not				isTrue
Arc	from to	element element	const const	IsTrue

Table 3: Elements of the FT SIMTHESys definition

Element	Property	Type	Modifier	Behaviors
Test	currentState	string	state	PerformTest, ComputeStateRewards, InitEvents, CountStateRewards SetStateRewards, ListImpulseRewards SetImpulseReward
State	prob	float	computed	GetNext
Sat	prob	float	computed	
Next	from to	element element	const const	IsTrue, GetDestination
Check				IsTrue

Table 4: Elements of the TEST SIMTHESys definition

Element	Property	Type	Modifier	Behaviors
compose				InitEvents, ComputeStateRewards, CountStateRewards, SetStateRewards, ListImpulseRewards, SetImpulseReward, PerformTest
Arc	from to	element element	const const	IsActive, HasSpace Push, Pull
CheckTrue	from to	element element	const const	IsTrue
CheckFalse	from to	element element	const const	IsTrue
CheckGE	weight from to	integer element element	const const const	IsTrue
CheckLT	weight from to	integer element element	const const const	IsTrue

Table 5: Elements of the composing formalism

queue is enabled and processes the requests. The semantics of the second (FCQN queue or PN place -> FT gate) is analogous to the semantics of a normal FT arc that connects an event to a gate. It feeds the gate with the result of a check over the number of units waiting in the queue or the number of tokens in a place. The semantics of the third (FT event -> TF check node) checks if the current probability of the top event satisfies the specified condition and activates consequently the check node.

The five elements of the Tester formalism are: Arc, CheckGE and CheckLT, CheckTrue and CheckFalse. All these elements are arcs that connect primitives belonging to the other four formalisms. Their properties and behaviors are summarized in Table 6. The Arc primitive interconnects elements of the two performance evaluation formalisms. CheckGE and CheckLT connect an element from a performance oriented submodel to a Check of a Test submodel, and they respectively signal that the occupancy of a queue or the marking of a PN place is either greater or equal, or less than, an integer value (the weight of the arc); CheckGE and CheckLT arcs can also interconnect an element from a performance oriented submodel to a FT gate. Finally CheckTrue and CheckFalse interconnect the TopEvent of a FT to a Check in the Test submodel. The condition in this case is verified if the TopEvent of the FT is true or false.

All the primitives belonging to the four submodel formalisms plus the composition formalism are graphically represented in Figure 1. Primitives belonging to the same formalism are grouped as enclosed by dashed contours. The bridge formalism in the middle of the figure also encloses the four Formalism Elements from the other formalisms, represented with dotted contours connected to their respective formalism representations, to symbolize its bridge nature.

5. PERFORMANCE EVALUATION ORIENTED SOLVERS

Two solution engines have been developed for the exponential event based formalisms presented in Section 4 that solve models using discrete event simulation or generation of a Continuous Time Markov Chain (CTMC). The obtained

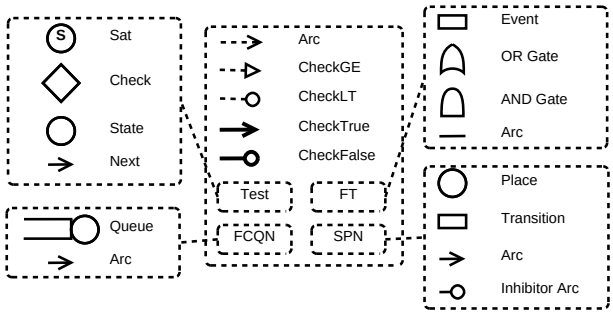

Figure 1: Elements of the formalisms and the multiformalism

CTMC can then be analyzed both in steady state and in transient. All solvers rely on the possibility of taking a *snapshot* of the state of the model (by storing all the properties with the *status* modifier), and then back track to it. Also all algorithms implement the `Schedule` behavior by storing all the scheduled events into a list. In the current implementation none of the solution components is optimized yet: currently their purpose is to show that they can be implemented easily and that they can solve a large variety of formalisms without having to re-design new solvers.

5.1 Stochastic Simulation

The simulator relies on the fact that, since all distributions are exponential, it is sufficient to reschedule all the events after each firing. The simulator repeats the analysis for a fixed number N of runs. Each run is executed until a global time T_{max} is reached and statistics are collected only after a transient time of fixed length T_{trans}. A snapshot of the initial state is taken, and after each run has finished, the snapshot is used to start a new simulation from the same initial state. The execution of each run simply calls the `InitEvents` behavior to find all the enabled events, and then draws an exponentially distributed sample for each of them. The event with the shortest sample is executed, and time is advanced accordingly until T_{max} is reached. At the end of all simulation runs, statistics are collected and returned to the

Element	Property	Type	Modifier	Behaviors
Tester	TestModel	string	computed	InitEvents, ComputeStateRewards CountStateRewards, SetStateRewards ListImpulseRewards, SetImpulseRewards
Arc	from	element	const	IsTrue
	to	element	const	
CheckLT	from	element	const	IsTrue
	to	element	const	
	Weight	element	const	
CheckGE	from	element	const	IsTrue
	to	element	const	
	Weight	element	const	

Table 6: Elements of the Tester SIMTHESys definition

model using the `SetStateRewards` and `SetImpulseReward` behaviors.

5.2 Numerical Analysis

The numerical solution solver generates the CTMC (Continuous Time Markov Chain) that describes the stochastic process equivalent to the model. Starting from the initial state, the solver calls the `InitEvents` behavior to compute all the enabled events. The solver then builds a transition graph, executing each enabled event. Finally, it checks if the snapshot of the obtained state has already been encountered (if not, it adds a new state), and backtracks to the initial event. The process is repeated until all the states have been visited. Snapshots are used both to reset the properties to a previously encountered state, and to backtrack to the current state whenever an enabled event is considered. Event firing rates are used to label the arcs of the transition graph. Each time a new state is encountered, the solver computes the reward vector associated to that state: for state rewards it simply accounts for the values returned by the `ComputeStateRewards` behavior. For the impulse rewards, it considers the reward value multiplied by the rate of the corresponding enabled event. The generator matrix C of the underlaying Markov chain is then computed from the transition graph. The solver calculates the steady state solution vector π of the Markov chain by computing $\pi C = 0$, and normalizing the solution such that all the components of π sums up to 1. Finally, a transient analysis is performed to compute $\pi(\tau)$ (where τ represents the time) using randomization. Performance indices are then computed by multiplying π (or $\pi(\tau)$) times the reward vectors, and are stored back into the model using the `SetStateRewards` and `SetImpulseReward` behaviors.

5.3 Solving Testing formalism

Solving the testing automaton requires the accumulation of probability of the Sat event(s) in a model. Due to the probabilistic nature of the approach, instead of developing a separate dedicated engine it has been decided to exploit the existing exponential events engines. At every state change, the conditions are evaluated and the probability of the Sat state is updated, according to the evolution of the events detected by the Check elements. The behaviors of the elements describe how to call the solving engines in order to get to the final result. The elements of the Tester formalism transfer the condition verified by the performance model to the testing submodel, achieving the final effect of verifying the given model.

6. EXAMPLE

An e-Government system that serves citizens' requests is composed by a number of different units. Every request is registered, examined and then dispatched to the appropriate processing unit. Some units are completely automated: the others need a human supervisor to authorize the results of the service. All processed requests are later checked and verified to decide if their processing is concluded or they generate another request that is sent back to registration. Human supervisors are not always available and are not exclusively in charge of a single unit. The system has to be monitored against the verification of certain conditions in case a service quality measure level is not satisfied.

The system is modeled by a multiformalism model composed of: a FCQN fragment, describing the flow of requests through units; a SPN fragment, describing the behaviour of the human supervisors; a FT fragment, that computes the quality level of the service (based on the blocking probabilities of the working units); a TF fragment, that verifies the undesired condition. Five versions of the model are presented in Fig. 2 and Fig. 3, in each of which the (a) part is common and the (b) parts describe the different implementations. The (a) and (b) parts have elements with the same names (in gray) that represent the same element.

The FCQN fragment is composed by a registering unit R, four working units W1-W4 and a control unit C. R dispatches requests to W1-W4 with probability p1, p2, p3 and p4 respectively. All requests processed by W1-W4 are sent to C, that in turn sends requests back to R for further operations with probability pR.

In the SPN fragment places EnableW1 and EnableW2 represent the activation of W1 and W2 when marked, since they represent the presence of the supervisor. Available supervisors are constituted of tokens in the `Here` place. Periodically supervisors start working at W1 and W2 (transitions StartW1 and StartW2) and, after some time they finish their working shift (transitions StopW1 and StopW2).

The FT fragment is composed by an AND gate A1 and an OR gate O1 that feed events EA1 and EO1, which in turn

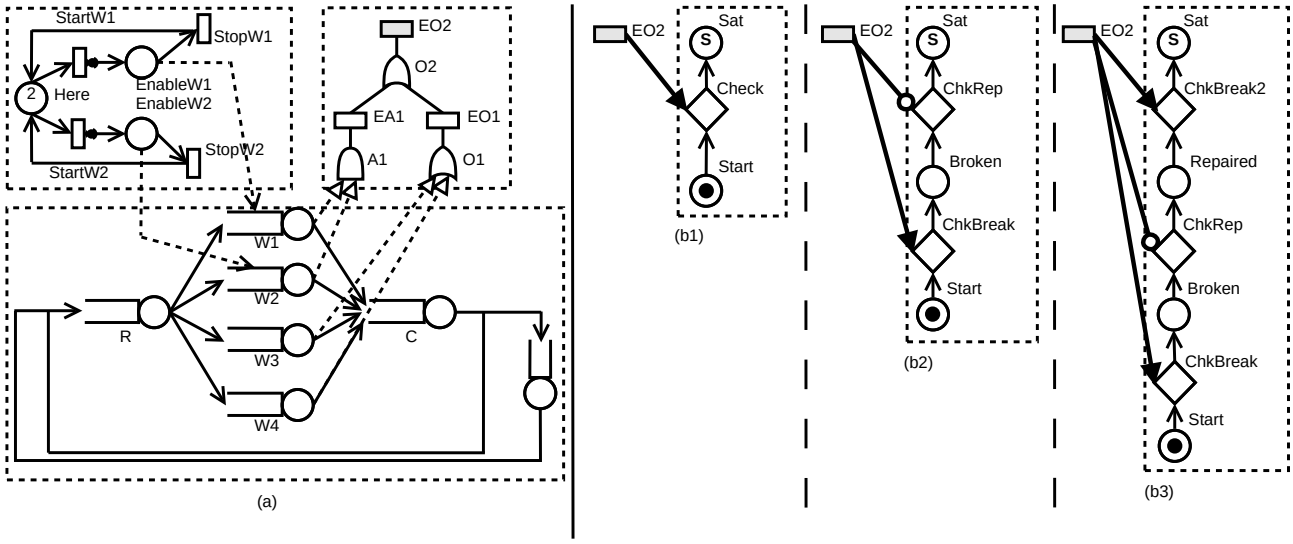

Figure 2: System model (1-3)

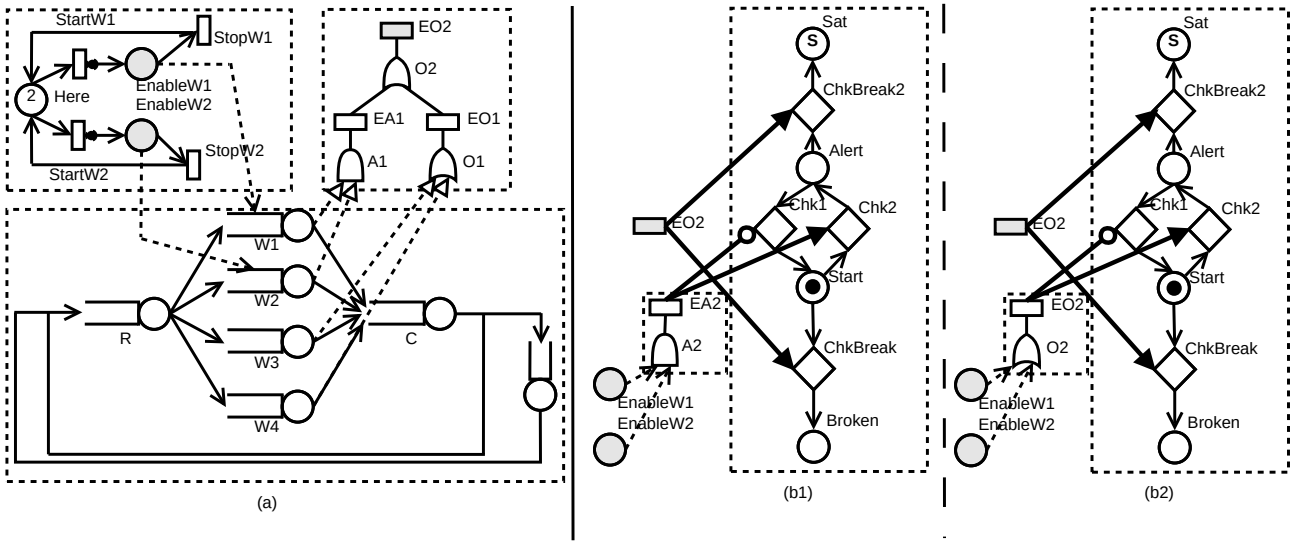

Figure 3: System model (4-5)

feed the OR gate O2 and finally the top event EO2. A1 ensures that, being W1 and W2 conditioned by the presence of human supervisors, their contribution to the overall quality metric (that combines queue full probabilities) is weighted as a whole, while O1 evaluates the feed of the automated part of the system, in which W3 and W4 are independent. O2 finally couples the two contributions as desired.

In Fig. 2 the model is completed with three different versions of a TF fragment. The first (b1) is composed by an initial State, a Check intended to verify a critical condition against the top event EO2 and a final Sat signaling that the critical condition has been reached. The second (b2) also considers the case in which the critical condition ceases by means of a Check that verifies its absence against EO2, and the third (b3) analogously verifies a second arise of the same problem with a third Check element cheching EO2. In Fig. 3 the model is completed with two other FT fragments, that check by FT fragments the condition of both resources working (b1) and at least one resource working (b2) in the SPN fragment, to evaluate the influence of these situations on the consequences of a critical condition, by the same TF fragment. The TF fragment is composed by two Check elements that account for the variations of the number of the working resources (Chk1 and Chk2) and two Check elements that verify the arise of the critical condition represented by EO2 (ChkBreak and ChkBreak2). If EO2 becomes true when the TF fragment is in the Alert state (less than 2 resources at work (b1) and no resource at work (b2)) the fragment activates the absorbing state Sat representing the monitored target condition, otherwise it activates the Broken state.

The average service times for W1, W2, W3 and W4 are respectively 15, 30, 2 and 1 minutes, while the probabilities that requests arrive to each queue are 0.1, 0.2, 0.3 and 0.4 and their maximum length is 2. The average service times for R and C is 0.1 seconds. R and C are unbounded. The probability that a request processed by C is sent back to R is 0.1. Average firing times for StartW1, StartW2, StopW1 and StopW2 are respectively 120, 30, 240 and 60 minutes. All service and firing times are exponentially distributed. The queue without name has service time 1 and is unbounded.

The results obtained on the three models in Fig. 2 have been resumed in Fig. 4, that shows the probability of the three absorbing states Sat, obtained with a transient analysis (currently, the results can be validated by performing the same analysis using other engines).

The figure shows the probability with which the system is in one of the three absorbing states (and the initial state) at a certain time. Fig. 5 shows the probability distribution function of the time at which the system reaches the critical condition for the first ('1st Failure') and second ('2nd Failure') time and at which the first critical condition is solved ('Repair'). The three curves suggest that '2nd Failure' is a most critical reference for the overall evaluation of the system, since it happens after a 'Repair' with a short delay, due to the buffer effect of the queues.

The results obtained on the two models in Fig. 3 have been resumed in Fig. 6, that shows the relation between the prob-

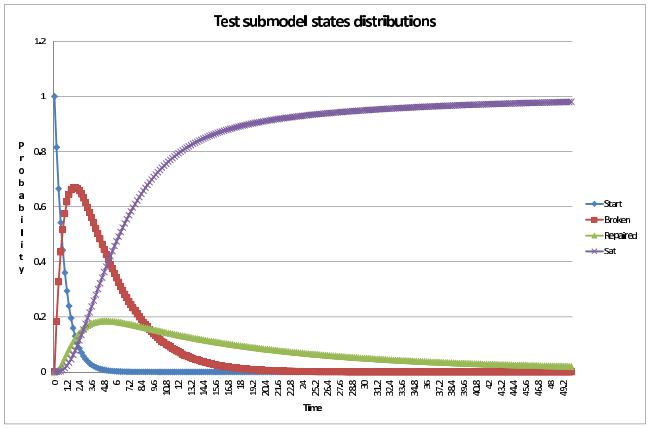

Figure 4: Absorbing states distribution (1-3)

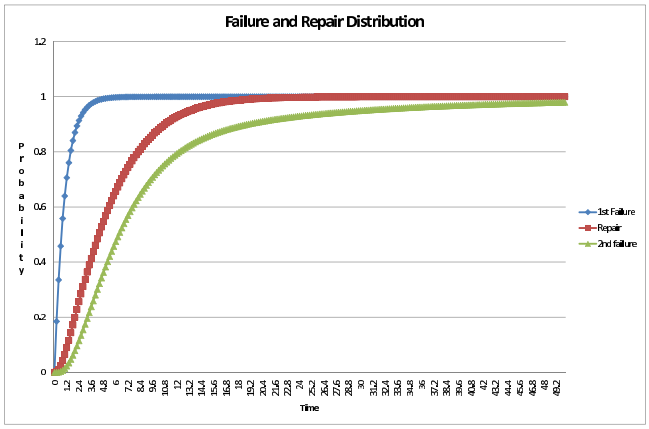

Figure 5: 1st Fault, Repair and 2nf Fault time distributions

ability of the critical condition and the ratio between the average working and non working time of workers in the SPN fragment, obtained with a steady state analysis, in the two cases of no resource at work ('Sat (OR)') or less than two resources at work ('Sat (AND)'). In this case average firing times for Start1, Stop1, Start2 and Stop2 variate so that for each couple Start+Stop is constant and Start/Stop has the value on the horizontal axis. The figure also shows the same relation considering the arise of the critical conditions with at least one resource at work ('Broken (OR)') or with two resources at work ('Broken (AND)'). The figure shows the obvious bigger tolerance of the OR situation, but also a substantially unexpected indifference of the probability with respect of work/pause ratios from 0.1 to 0.9, that suggests a proper resource management: workers assigned to this system can be also used for other tasks, assumed that approximately 10% of their time is allocated to this one. Moreover, at least with the given parameters, it is clear that the critical condition will happen with a non negligible probability, that suggests a revision of system dimensioning regardless of human resources. Due to the scope of this paper, this point is not investigated any further.

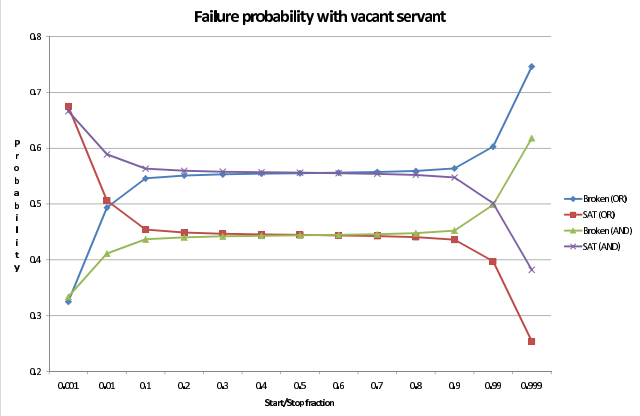

Figure 6: Critical condition probability vs human resource usage

Despite the fact that SIMTHESys solving engines are still experimental and no optimization has been applied yet in their design, the generated solver can obtain the desired solution in a few tenths of seconds even if the maximum number of generated states is approximately 80000.

7. CONCLUSIONS AND FUTURE WORKS

The main contribution of this paper with respect of [15] has been the presentation of a multiformalism environment that includes four different formalisms of three different categories, as well as an e-government case study. Even if ways of mixing Petri Nets and Fault Trees have already been considered in literature, this is to the best of our knowledge the first paper in which Markovian based models, logical specification and complex test expressions defined by special automata are joined in a single environment.

The way in which formalisms have been combined is open to new extensions: for example adding new Markovian based formalisms (such as Process Algebras), or reliability languages (such as Reliability Graphs) will require a small extra effort. The same reasoning applies also to the introduction of new optimized solution components that for example might exploit MDD (Model Driven Development) or other symbolical data structure to encode the state space and increase the solution efficiency.

The proposed methodology is supported by a tool called *SIMTHESysER* that can be freely downloaded from the project web-page [6].

8. REFERENCES

[1] Christel Baier, Lucia Cloth, Boudewijn Haverkort, Matthias Kuntz, and Markus Siegle. Model checking action- and state-labelled markov chains. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 701–, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Christel Baier, Lucia Cloth, Boudewijn R. Haverkort, Matthias Kuntz, and Markus Siegle. Model checking markov chains with actions and state labels. *IEEE Trans. Softw. Eng.*, 33:209–224, April 2007.

[3] Falko Bause, Peter Buchholz, and Peter Kemper. A toolbox for functional and quantitative analysis of deds. In *Proceedings of the 10th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, TOOLS '98, pages 356–359, London, UK, 1998. Springer-Verlag.

[4] D. Bjørner, C. W. George, A. E. Haxthausen, C. K. Madsen, S. Holmslykke, and M. Penicka. "uml-ising" formal techniques. In *Proceedings of INT'2004 - Integration of Software Specification Techniques for Applications in Engineering*, Lecture Notes in Computer Science, pages 423 – 450. Springer, 2004. Invited paper.

[5] G. Ciardo, R. L. Jones, III, A. S. Miner, and R. I. Siminiceanu. Logic and stochastic modeling with smart. *Perform. Eval.*, 63:578–608, June 2006.

[6] Graham Clark, Tod Courtney, David Daly, Dan Deavours, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick Webster. The mobius modeling tool. In *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 241–, Washington, DC, USA, 2001. IEEE Computer Society.

[7] Tod Courtney, Salem Derisavi, Shravan Gaonkar, Mark Griffith, Vinh Lam, Michael McQuinn, Eric Rozier, and William H. Sanders. The mobius modeling environment: Recent extensions - 2005. *Quantitative Evaluation of Systems, International Conference on*, 0:259–260, 2005.

[8] Tod Courtney, Shravan Gaonkar, Ken Keefe, Eric Rozier, and William H. Sanders. Mobius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *DSN*, pages 353–358. IEEE, 2009.

[9] Daniel D. Deavours, Graham Clark, Tod Courtney, David Daly, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick G. Webster. The mobius framework and its implementation.

[10] Clemens Fischer. Csp-oz: a combination of object-z

[6] http://www.dem.unina2.it/simthesys

and csp. In *Proceedings of the IFIP TC6 WG6.1 international workshop on Formal methods for open object-based distributed systems*, pages 423–438, London, UK, UK, 1997. Chapman & Hall, Ltd.

[11] F. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. Towards an object based multi-formalism multi-solution modeling approach. In *Proc. of the Second International Workshop on Modelling of Objects, Components, and Agents (MOCA'02), Aarhus, Denmark, August 26-27, 2002 / Daniel Moldt (Ed.)*, pages 47–66. Technical Report DAIMI PB-561, August 2002.

[12] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, F. Moscato, and V. Vittorini. Interfaces and binding in component based development of formal models. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 44:1–44:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[13] Marco Gribaudo and Matteo Sereno. Gspn semantics for queueing networks with blocking. In *Proceedings of the 6th International Workshop on Petri Nets and Performance Models*, pages 26–, Washington, DC, USA, 1997. IEEE Computer Society.

[14] Jochen Hoenicke and Ernst-Rüdiger Olderog. Csp-oz-dc: a combination of specification techniques for processes, data and time. *Nordic J. of Computing*, 9:301–334, December 2002.

[15] Mauro Iacono and Marco Gribaudo. Element based semantics in multi formalism performance models. In *MASCOTS*, pages 413–416. IEEE, 2010.

[16] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and Giuseppe Conte. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[17] Brendan Mahony and Jin Song Dong. Blending object-z and timed csp: an introduction to tcoz. In *Proceedings of the 20th international conference on Software engineering*, ICSE '98, pages 95–104, Washington, DC, USA, 1998. IEEE Computer Society.

[18] F. Moscato, F. Flammini, G. Di Lorenzo, V. Vittorini, S. Marrone, and M. Iacono. The software architecture of the osmosys multisolution framework. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, pages 1–10, 2007.

[19] Alexandre Mota and Augusto Sampaio. Model-checking csp-z: strategy, tool support and industrial application. *Sci. Comput. Program.*, 40:59–96, May 2001.

[20] Richard F. Paige. Heterogeneous notations for pure formal method integration. *Formal Asp. Comput.*, 10(3):233–242, 1998.

[21] Daniele Codetta Raiteri, Mauro Iacono, Giuliana Franceschinis, and Valeria Vittorini. Repairable fault tree for the automatic evaluation of repair policies. In *DSN*, pages 659–668. IEEE Computer Society, 2004.

[22] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems; An Example-based Approach Using the SHARPE Software Package*. Kluwer Academic Publisher, 1996.

[23] William H. Sanders, Tod Courtney, Daniel Deavours, David Daly, Salem Derisavi, and Vinh Lam. Multi-formalism and multi-solution-method modeling frameworks: The mobius approach.

[24] Kishor S. Trivedi. Sharpe 2002: Symbolic hierarchical automated reliability and performance evaluator. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, page 544, Washington, DC, USA, 2002. IEEE Computer Society.

[25] V. Vittorini, G. Franceschinis, M. Gribaudo, M. Iacono, and N. Mazzocca. Drawnet++: Model objects to support performance analysis and simulation of complex systems. In *Proc. of the 12th Int. Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (TOOLS 2002)*, London, UK, April 2002.

[26] Valeria Vittorini, Mauro Iacono, Nicola Mazzocca, and Giuliana Franceschinis. The osmosys approach to multi-formalism modeling of systems. *Software and System Modeling*, 3(1):68–81, 2004.