SIMTHESysER: a tool generator for the performance evaluation of multiformalism models

[Technical Report]

Enrico Barbierato
Dip. di Informatica, Università
di Torino
corso Svizzera, 185
10129 Torino, Italy
enrico.barbierato
@mfn.unipmn.it

Marco Gribaudo
Dip. di Elettronica e
Informazione, Politecnico di
Milano
via Ponzio 34/5
20133 Milano, Italy
gribaudo@elet.polimi.it

Mauro Iacono
Dip. di studi Europei e
Mediterranei, Seconda
Università degli Studi di Napoli
Belvedere Reale di San
Leucio
81100 Caserta, Italy
mauro.iacono@unina2.it

ABSTRACT
This article presents the architecture of SIMTHESysER, a new extendable multiformalism performance evaluation tool generator. SIMTHESysER relies on the main paradigm denoting multiformalism modeling techniques, that is providing the means to define models composed of entities described in different modeling languages. SIMTHESysER defines multiformalism models by separating the properties and relationships between their elements from the description of how they interact and evolve.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous;
D.2.8 [Software Engineering]: Metrics—complexity measures, performance measures

General Terms
Theory

1. INTRODUCTION
The advantages resulting from the adoption of the most suitable language for a certain part of the system are twofold: from the modeler point of view, it is possible to model each subsystem using the most appropriate language; from the solution point of view, the right (combination of) formalism(s) results in a proper mapping of the model concepts onto the solvers primitives.

SIMTHESys (Structured Infrastructure for Multiformalism modeling and Testing of Heterogeneous formalisms and Extensions for SYStems [17]) is a framework for the definition and solution of multiformalism models. It is based on the generation of specific solvers, starting from the rules contained in the formalisms definitions. Both formalisms and models are written in XML language.

However, up to now the methodology has not been supported by a specific tool. Although several tools with similar goals exist (see Mobius [9], Sharpe [19], SMART [7], the DEDS toolbox [5] and OsMoSys [13]), the innovation of SIMTHESysER is that it reduces the quantity of code required to specify the syntax and the semantics of a new modeling language. This capability is obtained by allowing the user to identify the effects that are important for the evolution of a model, and describe them in appropriate programming language.

The paper is organized as follows: Section 2 presents the architecture of the tool, while Section 3 clarifies its scope. A realistic case-study is presented in Section 4. Comparisons with related works and conclusions are given in Section 5 and Section 6 respectively.

2. ARCHITECTURE
This section presents the architecture of the tools, together with some of the notation used in the paper. The focus of the tool concerns the support of rapid formalisms development by exploiting the fact that formalisms relying on the same basic solution methods (i.e. simulation or mapping to equivalent continuous-time Markov chain) share common solving engines.

2.1 Notations
All the models considered are based on the SIMTHESys metamodel [17]. The basic part of a formalism is the Element, that defines all the atomic elements of a formalism (i.e. in a Stochastic Petri Net, the elements are the places, the transitions, and the arcs). Formalism Elements are used to define entire formalisms, and can contain other elements. An element is characterized by Properties and Behaviors. The former allows to associate values of given types to the elements of a formalism. A property value can be a constant (i.e. a label associated to an element), a variable (i.e. the number of tokens in a place) or used to store the computation of a performance index (i.e. throughput of a transition).
A behavior defines an action that the element performs. An element complies with several Interfaces, each specifying a set of behaviors that the element should define. Behavioral Interfaces allow the creation of families of elements that share similar features, abstracting the characteristics of the evolution of the model caused by a specific primitive. Modeling primitives can interact with each other using these abstractions, without the need of knowing the specific formalism to which they belong. Finally, Solver Interfaces and Solver Helper Interfaces allow the interaction with the solution engines that eventually compute the results of the analysis. The former provide the starting point for the solution of a model, while the latter allow the formalism primitives to call specific features of a solution engine. Fig. 1 shows an UML representation of the relationships between the entities of the architecture. You may refer to [17] for a complete description of the SIMTHESys framework.

Figure 1: SIMTHESysER metaformalism (M2)

2.2 DrawNET
SIMTHESysER has been integrated with DrawNET ([14]), a framework that supports the design and solution of models expressed in a graph-based formalism. Specifically, it provides a way to generate a specific file format, a Graphical User Interface (GUI), and an integrated modeling environment starting from a description of the primitives used by a formalism. DrawNET models provide the capability to compose different submodels, which conform to different formalisms (multi-formalism). The models are interfaced to existing solution components to complete their analysis. DrawNET uses the Data Definition Language (DDL), an XML based language that describes formalisms and models (refer to [14] for more details). Finally, note that this integration extends the functionality of DrawNET by providing the capability to solve the specified models in addition to the creation of the GUI.

2.3 SIMTHESysER workflow
Fig. 2 represents the architecture of the tool by showing the complete workflow of the solution of a model in a given formalism.

The input given to SIMTHESysER is the description of the formalism, while the output is a tool, based on the DrawNET GUI, that can be used to design and analyze models computing the required performance indexes. The tool can generate several solution engines for the same formalisms, which can then be selected by the user throughout the GUI.

Figure 2: SIMTHESysER architecture and workflow

Formalisms are described by Formalism Description Language (FDL) documents. To define a new formalism, the user creates an FDL file that declares all its modelling primitives. The FDL documents are provided to the FDL Analyzer, that builds the final solver and its GUI in several steps. First it generates the model parser (called MDL parser), that can read models in the specified formalism. Then the Solver Generation Facility links the model parser to the selected Solving Engines to produce a set of solver functionalities. Generated solvers are inserted in the DrawNET GUI, to produce the user interface. Models are designed with the GUI and saved in a specific format, called Model Description Language (MDL), where elements of the provided formalism are instantiated. Using the GUI, the user can save the models in the MDL format, and invoke the solvers on it, producing the required solutions.

2.4 Solution engines
Formalisms can be grouped into families. SIMTHESysER considers two formalism families (exponential events based and exponential and immediate events based) and offers six different solution engines (called Se, Sq, Cg, Cg, and Ce). Formalisms belonging to these families include SPN, Generalized Stochastic Petri Nets (GSPN), Markov Chains (MC), Queuing Networks (QN), Finite Capacity Queuing Networks (FQCN), Fault Trees (FT) and custom multi formalisms based on their composition.

Among the solution engines, Se and Sq evaluate performance indexes by simulation. The other four solution engines translate the evolution of the corresponding models to Continuous Time Markov Chains (CTMCs). Cg and Cg deal with steady state analysis, while Ce and Cg provide a transient solution. Se, Ce, and Cg consider only exponential events family (i.e. SPNs), and the others support both exponential and immediate transitions (i.e. GSPNs). Ongoing work is focusing on the development of a new formalism family, and a new solution engine based on simulation to support non exponential delays (Sn). Solution engines and formalisms families capable of considering phase-type distributions, and exploiting state space compression techniques (such as Multivalued Decision Diagrams [8]) are planned.

Performance indexes are defined in the FDL file as specific types of properties. For example, in a Petri Net they cor-
respond to the mean number of tokens in a place and the throughput of transitions. In a similar manner, other measures can be defined for other formalisms: for example, it is possible to define the fault probability for a FT.

All the previously mentioned engines work by taking snapshots of the state of the model. Each snapshot stores all the properties containing state information, and then the solution engine can use them to back track to a previous state. Each formalism must define a behavior that stores all the scheduled events into a list. The Simulator reschedules all the events after each firing then repeats the analysis for a fixed number of runs. Each run is executed until a global maximum time is reached; statistics are collected only after a transient time of fixed length. All these parameters are constants defined by the modeler. Initially a snapshot of the first state is taken, and then after each run has finished, the snapshot is used to start a new simulation from the same initial state. The execution of each run calls a behavior to find all the enabled events, and then draws exponentially distributed samples for each one of them.

The numerical solution solver generates the CTMC that describes the stochastic process equivalent to the model. Starting from the initial state, the solver calls the behavior that computes all the enabled events, then it builds a transition graph by executing each enabled event. It checks if the snapshot of the obtained state has already been encountered (if not, it adds a new state) and backtracks to the previous event; the process is repeated until all the states have been visited. The generator matrix $C$ of the underlying Markov chain is next computed from the transition graph. Finally, the solver computes the steady state solution vector $\pi$ of the Markov chain by computing $\pi C = 0$, and normalizing the solution such that all the components of $\pi$ sums up to 1.

### 3. TOOL SCOPE

SIMTHESysER main goals aim at simplifying the definition of new variants of existing formalisms, and allowing the creation of new formalisms without the development of specific solver tools. The main users of the tool are thus the formalism developers. However other categories of users can benefit from our proposal: model developers can exploit more favorable modeling primitives in their modeling activity; model users can use existing models by simply changing their parameters. The main functionalities, including their dependencies and their relations with the user categories are shown in the UML use case diagram in Fig. 3.

![Figure 3: SIMTHESysER use case diagram](image)

The aspects regarding the efficiency of the solving engines or their optimization, like solving stiff Markov models, models with infinite state spaces or the analysis of the performance of automatic generated solvers vs manually designed solvers, have not been addressed yet. However, the tool has been already experimented in several previous works, where models of moderate size (i.e. $10^5$ states) have been solved without any problem. We are currently working on more efficient solution engines: thanks to the architecture presented in Section 2.3, as soon as they will be available, they will replace the existing solvers, allowing all the formalisms and models developed up to that date to benefit from the solver improvements. The tool is currently available and freely downloadable from the SIMTHESys web site.

### 4. CASE STUDY

To show an example of the possible uses of the tool, this section describes a simplified version of a model of the distributed registration system designed for tracking detailed data about transformation and distribution in the agri-food sector [4]. Data transactions are supported by a communication protocol that implements the well known two-phase commit (2PC, see [18]) schema. The use of a careful exception handling based design can improve the availability of the system. An evaluation of the availability of the system, that is improved by the use of exception handling, is the goal of the study.

The system is composed by three subsystems: an Official Registry (OR), managed by a governmental authority, open to public consultation; a Company Registry (CR), private and managed by a company or a third party, that publishes only the non-confidential part of the stored complete information about the production; and a Tracking Manager (TM), that registers the operations performed on a site. Every registration operation by a TM, operated as a distributed 2PC transition, involves a CR for internal information computing and a OR for the related certification issues.

A multiformalism availability model of the system is shown in Fig. 4. The model has been written using a combined formalism that includes GSPN extended with exception handling to capture the transactional operations, and FT to evaluate the influence of faults. Faults in the TM are neglected. The formalism includes common primitives for immediate and timed transitions, places and related arcs (GSPN), AND and OR gates and events and related arcs (FT). Some unusual symbols are also used: exception triggering elements (marked with an E in a square), catch elements (ovals), and interformalism arcs. Exception triggering elements behave normally, but throw an exception interrupting their usual behavior according to a deterministic (hardware failure, restore in Fig. 4) or a probabilistic (1st phase in Fig. 4). Catch elements act on a submodel (OR, TM) changing the conditions of their internal elements when the related exception is triggered. Interformalism arcs connect element of different formalisms (e.g. if one of the CPUko places in Fig. 4 is marked, the arc transforms this condition into a fault input for the corresponding OR gate).

For a more detailed presentation and analysis of exception related elements see [4].

OR and CR are modeled by submodels that present the same

[1]www.dem.unina2.it/simthesys: see also for examples and references.
structure. Such submodels are composed by two parts. The left part is modelled by a GSPN that implements the server component of the 2PC protocol. It includes the possibility that a transaction may fail and raise a restore exception in the first phase. The right part of the submodel, is composed by three GSPNs that represent the working condition of three critical repairable components (HD1, HD2 and CPU), and a FT that throws an exception whenever the combined effect of component failures result in a system fault. TM is modeled by a GSPN that implements the client part of the 2PC protocol which throws a restore exception to handle a failure in the protocol due to a fault in TM or OR. UM represents user requests by a simple GSPN. The INTERCONNECTION NETWORK allows the proper interconnection between OR, CR and TM, and it is modeled with a proper GSPN.

**Evaluation results.** The model has been analyzed using different parameters. Results were computed by the SIMTHESys analytical solving engine. The rates have been chosen to match typical values collected from various literature. The parameters marked by a star have been scaled to study the system performance under different rates. In particular all the marked parameters have been multiplied by the same factor in the different experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>preparemsg, commitmsg</td>
<td>5.00000E−01</td>
</tr>
<tr>
<td>recvok1, recvok2</td>
<td>5.00000E−01</td>
</tr>
<tr>
<td>recvack1, recvack2</td>
<td>5.00000E−01</td>
</tr>
<tr>
<td>firstPhase, secondPhase</td>
<td>2.00000E−01</td>
</tr>
<tr>
<td>CPU Fault</td>
<td>3.12500E−09</td>
</tr>
<tr>
<td>HDD Fault</td>
<td>8.33000E−09</td>
</tr>
<tr>
<td>CPU Repair</td>
<td>5.55000E−04</td>
</tr>
<tr>
<td>HDD Repair</td>
<td>2.77000E−04</td>
</tr>
<tr>
<td>Restore</td>
<td>9.25000E−05</td>
</tr>
<tr>
<td>tUsers</td>
<td>2.50000E−01</td>
</tr>
</tbody>
</table>

Table 1: Parameters of the agri-food model

Figure 5 shows the throughput of the system considering a model M1 without faults and a model M2 were software and hardware exceptions can occur during the first phase (Malformed req.). Both the models uses the parameters in Table 1 though for M1 all the rates of the fault transitions and the probability of the exception throwing nOk have been set to zero.

![Figure 5: System throughput](image)

Figure 6 shows the relative distance between the throughput of tUser in models M1 and M2. This value has been calculated as \((T_{M1} - T_{M2})/T_{M2}\) where \(T_{M1}\) and \(T_{M2}\) represent...
the system throughput in the best case (i.e. no faults occurring) and in the worst case (i.e. both software and hardware faults occurring).

5. RELATED WORKS
Regarding previous research on the SYMTEHSys methodology ([17], [3], [12] and [4]), this paper presents the tool that supports the framework.

SIMTHESysER provides a more general modelling environment with respect of Mobius [9] at a price of a reduced solution efficiency. The Abstract Functiona Interface (AFI) [11] that defines all the possible evolution is not required in SIMTHESysER. In a certain sense, Mobius AFI, can be considered as a specific set of Solver Interfaces and Solver Helper Interfaces in the SYMTEHSys methodology.

Concerning Sharpe [19] and SMART [7], SIMTHESysER doesn’t currently provide optimized solution engines as they do. However, its set of existing solution engines is planned to be expanded in order to provide solutions that exploit some of the basic principles on which such tools are based.

Regarding OsMoSyS [13] and SPE-ED [1], SIMTHESysER focuses on a primitive level interaction among the elements of the formalisms, and not only on the multisolution obtained by mixing existing solvers throughout the definition of a workflow. Such techniques can also be included into SIMTHESysER by the creation of a suitable set of behaviors. This capability is currently under study.

Similarly to AToM³ [10], SIMTHESys formalisms and models are described as graphs based on a metaformalism, and the integration with DrawNET similarly allows to visually manipulate models described in specified formalisms. Both tools greatly consider the relevance of the operational semantics of formalisms, AToM³ for simulators generation and SIMTHESysER for multiformalism solvers generation. However, AToM³ privileges model transformations, while SIMTHESys relies on extensibility through the behaviors.

6. APPLICATIONS AND CONCLUSIONS
Literature and practice offer a wide range of modeling formalisms, either general (as Petri Nets, Fault Trees, Queueing Networks) or for special domains, and some examples of formalisms combination techniques (multiformalism) that exploit the possibility of using different formal languages to specify different portions of a model, to be eventually solved by integrating different existing performance evaluation tools (multisolution). However every tool is still limited in some aspect, and the topic is still a research issue.

SIMTHESys is an original approach to the definition and the evaluation of multiformalism performability oriented formal models. The main innovation in SIMTHESys is the possibility of specifying all aspects of a formalism in a single document by an object-oriented XML based description, including multiformalism interactions. This is made possible thanks to the use of behaviors, simplifying the description of the semantic of a formalism.

SIMTHESysER is a valuable support for the rapid generation of multiformalism performance evaluation tools and a valid help for researchers and practitioners willing to exploit flexibility in their models. Since the tool is relatively new (the first version was made available in September 2010, and in previous SIMTHESys works results were obtained manually), no literature that uses results computed using SIMTHESysER is yet available.

Future work includes the capability of editing an FDL file directly from DrawNET (currently files of this sort are created externally with an XML-enhanced text editor), the study of new model families (such as Fluid Models [16] or Markovian Agents [15]), and the introduction of new solution engines based on Product Form Solutions [2] and Mean Field Analysis [6].

7. REFERENCES


