Element Based Semantics in Multi Formalism Performance Models

M. Iacono[1], M. Gribaudo[1]

[1] Dipartimento di studi Europei e Mediterranei, Seconda Università degli Studi di Napoli, Caserta, Italy
[2] Dipartimento di Elettronica ed Informatica, Politecnico di Milano, Milano, Italy

mauro.iacono@unina2.it gribaud@elet.polimi.it

Abstract—The design and the requirements of modern computer-based systems have reached a complexity level that calls for the use of models for the verification of non functional requirements since the beginning of their design cycle. Such systems are however too complex to be modeled directly in a simple unstructured formal language like Queueing Networks or Petri Nets. SIMTHESys (Structured Infrastructure for Multiformalism modeling and Testing of Heterogeneous formalisms and Extensions for SYStems) is a novel approach to multi formalism compositional modeling, that is based on the possibility of freely specifying the dynamics of the elements of a formal modeling language in an open framework. This is obtained by the application of consolidated metamodeling foundations to the description of models, together with the concept of behavior as a bridge between formalism dynamics and solution techniques. In this paper the main concepts of the SIMTHESys approach are presented, together with a running example of how SIMTHESys copes with performance evaluation of multi formalism models.

I. INTRODUCTION

In recent years common systems like Computer Networks, Multi-core architecture, Web Services, Critical Infrastructure Controllers have reached a complexity level that suggest the use of modeling techniques during their design phase. The same systems are however too complex to be modeled directly in a simple unstructured language like Queueing Networks or Petri Nets. Component Based techniques [1] allow the representation of a large system as a collection of interacting sub systems. At the same time, subdivision of a complex model into components suggests the possibility of expressing each of the sub-systems using a different language. Multi-formalism modeling techniques allow the definition of the components describing the complete model to be specified using the most appropriate formalism.

One of the problems of multi-formalism is that it requires the development of software solution components that are very specific to the particular set of interacting formalisms considered. Usually, the definition of a new formalism similar to an existing one requires a large amount of redesign of the solvers that is not usually straightforward and can result in an unaffordable design and waste of development effort with respect to the benefits. Moreover, a too specific combination of formalisms will very unlikely be reusable for other projects. Thus, a solution component risks to be developed only for a very specific problem.

In this work, we propose a way of encapsulating the evolution policy of a formalism into the definition of its syntax, to simplify the combination and the variation of existing formalisms. The proposed framework decouples the solution technique from the description of the model evolution while keeping the focus on the modeler and his perspective of the model.

The proposed approach is not only capable of considering standard event based modeling languages with exponential distributions, such as Stochastic Petri Nets (SPN) and Queueing Networks (QN), that are introduced in this paper for the sake of simplicity, but is specifically designed to support other types of formalisms too, even non state space based formalisms as Fault Trees (FT) and Bayesian Networks (BN).

II. RELATED WORKS

Multiformalism modeling for performance evaluation of systems is widely present in scientific literature as a solution to cope with heterogeneity and complexity of systems. Early experiences with multi formalism modeling are given by Sharpe [2], in which models are composed by submodels in a fixed set of different formalisms, that are solved by different solvers. In Sharpe submodels interact by exchanging probability distributions to obtain the global result and the solution process is determined by the user. The main references for this paper are Mobius [3] [4] [5] and OsMoSys [6] [7] (that also present a more extended state of the art). Both of them aim to provide a methodology and a tool for extensible multiformalism models design and evaluation and consider model composition and multiple solution methods as the foundation of their model solution process. From the point of view of multiformalism modeling, both approaches use submodel composition to support formalism interaction, but with different premises. In Mobius submodels interact by sharing state variables and by superposing events between submodels; in OsMoSys interactions are defined by using operators, that formally describe the semantics of information exchange between submodels. Extensibility is another common goal of both the research initiatives: Mobius is designed to be extended by third-party formalism that can be integrated in the framework through proper APIs, while OsMoSys exploits a sophisticated object-oriented metamodeling approach to allow users to include new formalisms by specifying their description in terms of elements and constraints. From the point of view of solution processes, Mobius offers a very flexible and complete execution policy for model elements, that is implemented by a framework
of specialized solution algorithms, and allows the automated generation of a custom software that solves the specific model; OsMoSys exploits external existing solvers by encapsulating them in proper software components (namely adapters) and using them to build solution processes in the form of workflows, which structure is generated by examining the model and its operators. The SIMTHESys (Structured Infrastructure for Multiformalism modeling and Testing of Heterogeneous formalisms and Extensions for SYStems) approach is based on the possibility of defining the semantics of formalism elements in the same description of the syntax of the formalism. This allows for a great flexibility in the specification and solution of multiformalism models and for supporting extension and experimentation of new formalisms and variants.

III. THE SIMTHESYS APPROACH

SIMTHESys allows users to design models, by using one or more existing formalisms exploiting their own characteristics, or to design (new) formalisms, to cope with special needs or to evaluate the effects of new ideas. In the first case, SIMTHESys offers a sound way to specify models by hierarchical composition of reusable model components: it gives the modeler a flexible environment for the evaluation of different possible formalism concepts, by simply specifying their elements and respective behaviors in terms of calls to proper behavioral libraries. The specification of both the elements of a model and their behavior allows to decouple the problem of creating new formalisms (or formalism variants) with a desired, specific semantics from the problem of solving a model written in that formalism. The general idea behind SIMTHESys is the management of custom formalism behaviors by means of a middle layer that exploits behavioral libraries provided by solvers to generate a model description suitable for non-specialized solution algorithms. A user willing to specify a new formalism can describe the effects of events over model elements, that will result in the generation of what is needed to solve the desired models, e.g. a state space or a simulation trace for SPN depending on the desired solution technique.

Multiformalism is obtained by allowing to consider heterogeneous models. This can be implemented by proper composition formalisms. Such formalisms define hybrid elements which behaviors are designed for the interaction with elements of different formalisms, and encapsulate in their behaviors the interaction logic. The described behaviors-based approach allows to specify complex interactions between heterogeneous models in a compact, atomic way that is transparent to simple modelers: it appears as a feature of a special formalism that can be applied to own models without any further knowledge of behavior design logic. Conversely, the possibility of specifying in a formalism (FDL) the interaction between different languages offers a unique tool for the design of multiformalism models without forcing a formalism designer to redesign related solvers. How this is obtained is described in the next section.

IV. BEHAVIORAL FACILITIES

Behavioral Facilities (BF) are the cornerstone of the flexibility characteristics of SIMTHESys. Formalisms can be grouped in families, that share some common basic aspects (totally or partially). While differences in formalisms are the key for fitness for different applications, these common aspects allow for the definition of core solving engines, independent from formalisms, around which the differences can be seen as a middle layer made to obtain the desired semantics. E.g. SPN are fit for modeling concurrent timed systems while QN are suitable for schedulers and services, but both of them share the stochastic time characteristics that can be analyzed by different techniques; and Generalized Stochastic Petri Nets (GSPN) differ from SPN because of the existence of immediate (non timed) transitions, so that both of them describe concurrent timed systems but they need different solution engines.

The role of BF is to act as the middle layer between models (as described by formalisms) and solvers, with the aim of decoupling related problems and allowing an easy mapping of different (and new) formalisms on common solving engines, without losing the natural relations between them, and opening the way to the introduction of new engines. BF abstract primitives needed by solvers to build their data structure in support of solution algorithms and present them to the formalism developer with a familiar aspect, filling the semantic gap between complex formal concepts and simple low-level solver calls.

BF can be considered as libraries dedicated to translate concepts typical of a family of formalisms in more general descriptions of the evolution of a model. A behavioral facility can be used by more formalisms (or formalism variants) to compose the behavior of a single formalism element, described as a script of behavioral library primitives. This approach isolates common aspects of the evolution of formalisms in atomic scripting commands that resembles the specific operators defined to operate in a specific family of formalisms, thus easing the learning curve of formalism developers. BF mainly provide:

- **behavioral interfaces**: a number of predefined abstract behaviors that formalisms must implement to obtain specific services from BF;
- **solver interfaces**: libraries that enable the access to solving engines;
- **FDL analyzer**: examines FDL files to generate what needed to integrate the FDL information with the other facilities and build a reusable interpreter that can execute and solve MDL documents based on the formalism.

Besides the advantages already cited, this approach enables a straightforward implementation of complex integrations of different formalisms. Actually hybrid elements of composition formalisms are cross-formalism entities that benefit from native behaviors of different BF, by which the formalism designer is free to concentrate on the semantics of the interaction rather than on the technical problems of implementing it. BF enable a constructional approach to the development of
new formalisms, because they can be layered to obtain more complex behaviors for more abstract formalisms, or coupled as in the case of compositional formalisms, or experimented and consolidated in case of success by creating new BF; but also a deconstructional approach, because their mediation allows the design of solution engines as interpreters of elementary behaviors that analyze complex behaviors as aggregates.

V. Example

To show the use of SIMTHESys, consider the case of a server that has two computing facilities, both of which are subject to faults that are recovered when needed. The computing facilities have buffers for incoming requests, and are designed so that in case of unavailability of free positions in the buffer they stop accepting requests. Requests are managed by a frontend that dispatches them to one of the computing facilities. If the first computing facility does not manage to process a request before a timeout, the request is redirected to the second computing facility and the first will start processing the next request. In order to evaluate the performances of the system, here a multiformalism model is proposed in which simple submodels designed in SPN and FCQN formalisms interact. SPN is a variant of Petri Nets in which transitions are characterized by a stochastic timed behavior, with an exponentially distributed firing time, and consequently an active transition will consume the required tokens after an exponential delay. FCQN are a variant of queueing networks in which a full queue has the effect of blocking the activities of another queue that is sending a processed customer to it.

To simplify the presentation of the example, in this paper queue service times are considered exponentially distributed, with no loss of generality of the approach. FCQN can perfectly model the computing facilities and the frontend of the system, while SPN can be used to describe the fault/repair process and the timeout, assuming the exponential distribution of the processes. The integration of the two formalisms can be done by deciding and defining what elements are needed in the composition formalism and what semantics they implement to let elements from the two formalisms interact. The choice of SPN and FCQN, besides their fitness to the application, offer a further advantage for the purposes of this paper because it makes relatively simple to define and present the composition formalism and to analyze the effects of the interaction semantics, and at the same time it allows to show the possibilities given by this approach. The design choices led to the definition of arc elements that implement four interactions (resumed in fig. 1):

SPN place to FCQN queue (see fig. 1 a)): this interaction can be considered as an extension of enabling arcs in PNs. The behavior of this arc enables the execution semantics of the queue when the place is marked (blocking it as the successor was a full queue otherwise);

SPN transition to FCQN queue (see fig. 1 b)): this interaction can be considered as an extension of token production in an output place in PNs. The behavior of this arc produces a customer in the queue when the transition fires, if the queue is not full: otherwise blocks the firing until there is a free place in the queue for a customer;

FCQN queue to SPN place (see fig. 1 a)): this interaction is similar to the previous one. The behavior of this arc produces a token in the place every time processing of a customer in the queue finishes;

FCQN queue to SPN transition (see fig. 1 b)): this interaction is an extension of transition enabling in PNs. Every time a customer enters the server of the queue, the transition is enabled, while the end of processing disables it; firing of the transition consumes a customer from the queue.

The BF imports the descriptions of SPN and FCQN and defines a formalism element that uses the same solver and implements the same interfaces by behaviors ‘InitEvents’, ‘computeStateRewards’, ‘countStateRewards’, ‘setStateRewards’, ‘listImpulseRewards’ and ‘setImpulseReward’, the details of which are omitted. This composition formalism only has one element, ‘Arc’, that is designed to obtain the four desired interactions. This element implements all the interfaces that are needed to operate on all elements of both formalisms by offering the ‘Push’, ‘Pull’, ‘isActive’ and ‘hasSpace’ behaviors. A simple reimplementation of the key behaviors defined in SPN and FCQN is all that is needed to obtain a SYMTHESys formalism capable of interfacing these two formalisms. Given this formalism, it is simple to model the example as in fig. 2 (which MDL description is omitted for the sake of space).

The model is composed by 6 submodels: Main Server and Serv 2 are FCQN and describe the main subsystem, made by the frontend and the first computing facility, and the

Fig. 2. Model of the example system

The model is composed by 6 submodels: Main Server and Serv 2 are FCQN and describe the main subsystem, made by the frontend and the first computing facility, and the
second computing facility; Load, Trf, Serv1 B/R and Serv1 B/R are SPN and respectively represent the load generation, the watchdog timer that transfers a request from the first to the second computing facility and the model of faults and repairs of Main Server and Serv2. Main Server is composed by two queues, the first of which represent the dispatcher that accepts requests from the outside world and passes them to the computing facilities represented by the second queue. In Serv1 B/R (Serv2 B/R) the rate of T11 (T21) represents the MTTF (mean time to failure) of Main Server (Serv2) while the rate of T10 (T20) is its MTTR (mean time to repair). Load represents the arrival process of requests (arrivals happens according to the rate of T00). Submodels are connected by the hybrid arcs defined in PN-FCQN. T00 produces customers for Q41, P11 (P21) enables Q40 (Q50), while T30 transfers the customer that is being served in Q40 to Q50 when firing (exploiting the all-exponential hypothesis). Q40 and Q50 are connected to P00 to allow a steady state evaluation of the model. Whenever the main server is not able to process a customer within the expiration time of the Trf watchdog, due to a fault or not, a customer is generated into Serv2, that will serve it in a time that depends both on Q50 rate and the effects of Serv2 B/R.

The model has been evaluated to compute throughputs, mean lengths and mean number of tokens in the elements with a number of items N in the system that variate from 1 to 20. Two engines have been used, one based on exact solution of the markov chain generated by the system and one based on its simulation, which gave the same results. Results are plotted in fig. 3: the right scale concerns the number of states; T00(T), Q41(T), Q40(T) and Q50(T) appear as superposed because they are very similar.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper a novel approach to multiformalism modeling has been presented. The SIMTHESys approach is based on the use of the concept of behavior to define formalism semantics. This allows an easy definition of new formalisms, as well as the definition of multiformalism models. This work presents the first results of a very promising research activity and many open questions still offer many directions of investigation. The next steps will concern the study of alternative behaviors in multiformalism applications and their comparison, the implementation of more natural and easy to use BF for the development of elements behaviors and the introduction of formalism and elements inheritance for the extension of formalisms and the application to different real world problems, such the study of wireless sensor networks (WSN). Moreover, a dedicated scripting language for an easier, less programming-oriented description of behaviors will be developed. Future works concern the extension of the framework towards model transformations applications and inclusion of existing solvers.

ACKNOWLEDGMENTS

This work has been partially supported by MIUR fund through the PRIN 2007 project under grant 2007J4SKYP.

REFERENCES