Workflow Principles Applied to Multi-Solution Analysis of Dependable Distributed Systems

Francesco Moscato,
Dip. di Ing. dell'Informazione
Seconda Universitá di Napoli
Aversa, Italy
francesco.moscato@unina2.it

Nicola Mazzocca
Dip. di Ing, dell'Informazione
Seconda Universitá di Napoli
Aversa, Italy
nicola.mazzocca@unina2.it

Valeria Vittorini
Dip. di Ing. dell'Informazione
Universitá di Napoli Federico II
Napoli, Italy
vittorin@unina.it

Abstract

Real world dependable distributed systems are often heterogeneous, not only in their physical composition, but also from a modeling and analysis perspective. Indeed, different components may be modeled by using the most suitable modeling formalism and multi-solution strategies may be applied to analyze the resulting multi-formalism model since no single solution method is adequate to solve all submodels. In this paper we present the architecture of an extensible multi-formalism framework for the modeling and design of distributed dependable system. We show that the process needed to solve/analyze a model expressed through different formalisms may be described as it were a business process and executed by means of a workflow engine. We apply the proposed technique to a fault tolerant remote SCADA (Supervisory Control And Data Acquisition) system.

1. Introduction

Modeling and simulation are becoming increasingly important to enable the analysis and design of real distributed dependable systems whose components and aspects cannot be expressed by a single modeling or simulation formalism. To cope with the complexity of such systems multi-formalism approaches are emerging that allow to use different formalisms to model and analyze different subsystems and also to promote models reuse. Indeed, multi-formalism multi-solution approaches are very appealing, but proper modeling methodology and technology must be developed. Some results in this line have been described in the literature. For example SHARPE [11] and SMART [2] are software tools that can be considered a first step towards the development of multi-formalism multi-solution frameworks. Nevertheless, to the best of our knowledge, the only environment that implements an integrated multi-formalism multi-solution approach is the Möbius framework [4] where the semantics of the formalism is expressed by coding a number of predefined C++ methods, adhering to a given Abstract Functional Interface. A recent trend in multi-formalism modeling is to take advantage from meta-modeling (i.e. the process of modeling formalisms) [14], that greatly promotes the availability of CASE tools supporting multi-formalism modeling (e.g. [10], [3], [5], [8]). In particular, AToM^3 [3] implements a modeling approach based on the combination of meta-modeling and graph transformation techniques. This approach allows to analyze the models by translating all components into a common formalism (in [3, 13]). This paper discusses the use of workflow management to automatize the solution process needed to solve the multi-formalism model of a dependable distributed system. We show that such process may be described as it were a business process and executed by means of a workflow engine. The proposed workflow-based solution is part of the OsMoSys (Object-based multi-formaliSm MOdeling of SYStems) project which aims at defining and implementing an extensible multi-formalism multi-solution framework for the modeling and design of distributed dependable systems. This work focuses on multi-solution and applies the proposed workflow-based technique to a fault

Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP'04)
1066-6192/04 $20.00 © 2004 **IEEE**

tolerant remote SCADA (Supervisory Control And Data Acquisition) system of a robotic cell involved in a palletizing process. The rest of the paper is organized as follows. Section 2 briefly describes the overall OsMoSys modeling framework and outlines the architecture of the OsMoSys workflow engine that is currently under development. Section 3 introduces the case study application and its multi-formalism model. In Section 4 the mapping between workflow process and solution process definition is defined and applied to the SCADA system. Section 5 reports some experimental results. Finally, Section 6 contains few closing remarks.

2. Solving multi-formalism models with a multi-solution approach

The work described by the present paper has being developed within a larger research project on multi-formalism modeling and analysis of complex systems. The workflow-based approach firstly introduced by this paper is going to be integrated in the OsMoSys framework. In this section we briefly describe the main ideas behind the OsMoSys multi-formalism multi-solution approach. In particular the modeling methodology is outlined, and the overall OsMoSys architecture is presented.

2.1. The Modeling Methodology

The OsMoSys modeling methodology stemmed from the need to support compositionality both within a single formalism and among different formalisms. The relevant aspects of the methodology are the possibility of defining any graph based formalism by using a meta-language called *metaformalism*, and the possibility of organizing a model as a hierarchy of sub-models. Formalisms definitions, called *metaclasses*, can be organized in a hierarchy, so that different *dialects* (or extensions) stemming from a parent formalism can be easily derived in a consistent way. An instance of a model class is said to be a *model object*. In the practice, libraries of parametric sub-models could be constructed, related with a specific application domain or problem class, from which sub-models can be extracted and composed in different configurations to model different scenarios.

A metaclass can be used both to express the syntax of well known formalisms such as Generalized Stochastic Petri Nets (GSPN), Queuing Networks (QN), and Fault Trees(FTs) , or to define the syntax of new formalisms that could be used to express sub-models composition operators and to compose several sub-models expressed in different formalisms, allowing to build multi-formalism models. This type of metaclasses have been called *bridge metaclasses*. Here we will use a bridge metaclass to build the final model of the SCADA system in Section 4.

2.2. Overall OsMoSys Architecture

The OsMoSys framework has been designed to effectively support the solution and the analysis of the multi-formalism models built according to the modeling methodology briefly described in Section 2.1. The solution we propose to achieve multi-solution relies on a three-tier architecture depicted in Fig 1. A graphical user interface (DrawNET++) runs on the client and provides the support to develop the multi-formalism models. The middle-tier server is a workflow engine that enacts the process required to solve a multi-formalism model. The applications(on server side) are legacy simulation/analysis tools or other tools involved in the solution process. The legacy simultaion/analysis tools are integrated into the framework by the definition of proper Adapter modules.

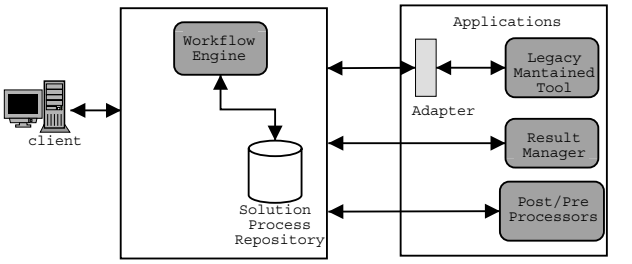

Figure 1. OsMoSys Conceptual Architecture

2.3. DrawNET++

The OsMoSys framework includes DrawNET++ [7, 6], a configurable GUI for graph-based formalisms used to define multi-formalism models. DrawNET++ supports the multi-formalism modeling methodology we have mentioned before and generates reusable models that can be used to build libraries of models.

2.4. The Workflow Engine and the Process Definition Meta-Model

In order to analyze multi-formalism models proper solution processes must be defined and applied through the OsMoSys framework. To cope with the problem of multi-solution when analyzing/simulating multi-formalism models and to automate the execution of the solution processes, a mapping between workflow processes and solution processes is introduced. The workflow engine is in charge of enacting the solution processes according to their specifications. A workflow process definition is defined in [15].
In the same way we define **a solution process definition**

in the OsMoSys framework as a representation of the solution process needed to solve/analyze a multi-formalism model. It consists of a networks of activities too, that may refer to solvers or tools (i.e. Applications) in order to perform, even concurrently, some tasks of the solution process in the multi-solution environment. In Fig. 2 an UML diagram is shown representing the main components of a solution process definition and their relationships. The Process Definition entity in the diagram provides information that applies to the other entities and information about the administration of the process, such as time limits, execution priority etc. The other entities, corresponding to the entities of a workflow process definition meta-model [15], are: *Participants; Applications; Activities; Transitions* and *Relevant Data*.

In a multi-formalism/multi-solution domain we define a

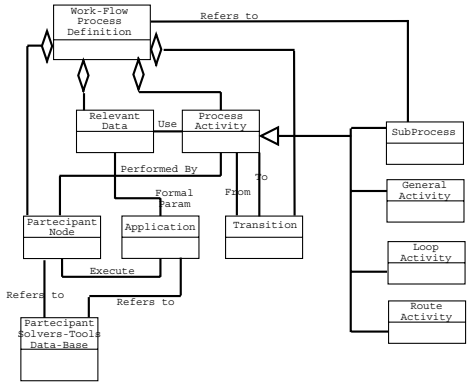

Figure 2. Solution process definition

Participant to be a physical elaborative node where applications are executed. It is possible to create more than one application instance on a participant node, but a single instance refers to only one activity. Activities, instead, comprise logical, self-contained unit of work within the project. A general activity performs a set of tasks, invoking one (or more) application services. Attributes may be defined to specify activity control information and data used as input or output (such as input or output files) parameters. A process definition itself consists of a network of Activity nodes, and it may be represented by a directed graph whose edges, named Transition, are represented by a couple: $(FromActivity, ToActivity)$ An Activity, may define JOIN conditions on incoming transitions and SPLIT conditions on outgoing transitions. A JOIN describes the semantics of an activity with multiple incoming transitions. The SPLIT of an activity describes the semantics with multiple outgoing transitions.

As shown in Figure 2, activities may also be: Route, Loop or Subflow Activities. The semantics of these activities is explained in [15]. The OsMoSyS workflow engine architecture is shown in Fig.3. To manage the solution process, the workflow engine performs the following tasks:

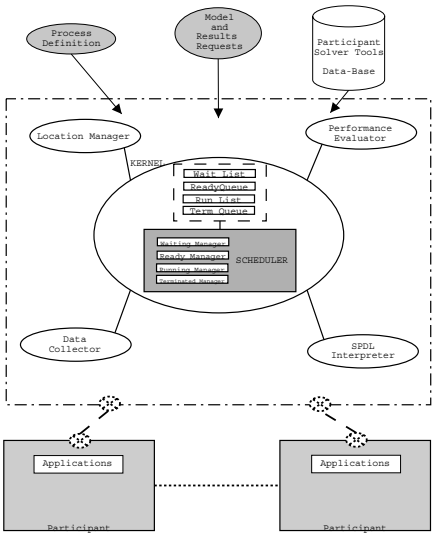

Figure 3. OsMoSys Engine architecture

- Instantiates the applications needed to perform the solution process activities;
- Performs a dynamical load balancing on participant nodes;
- Performs routing of data needed by the applications to accomplish their tasks.

The *Kernel* fetches a process definition (expressed in SPDL), a model definition and its related results request (both express in the DrawNET++ XML format) and performs the solution process. The kernel instances the activity-related applications on participant nodes. The participant is chosen on the basis of the informations provided by the *Performance Evaluator* that dynamically estimates the load of the participant node and other performance parameters.

The *Scheduler* consists of four manager processes, and four lists/queues to manage the activities. The activities wait on the incoming transition condition variables. When a condition variable becomes true, the scheduler wakes up the wait manager to control if one or more waiting activities can be inserted in the *ready queue*.

Activities in the ready queue refer to applications ready to be instantiated to perform the activity work. Once instantiated (the participant physical location is retrieved by the *Location Manager*), a ready application becomes running and it is inserted in the *running list*.

At the state, a prototype implementation of the workflow engine has been developed.

2.5. The Application Programs

The application programs are grouped in the following classes: a) legacy analysis/simulation tools that actually solve the (sub)models; b) adapters needed to integrate the legacy solvers into the framework; c) the Results Manager that is in charge of allowing the visualization of solution results on the GUI by interpreting the requests coming from DrawNET++ and collecting the results to forward to it according to a predefined XML format; d) pre/post processors, i.e. tools that may participate in the solution process (e.g. to translate the XML representation of the DrawNET++ models into a syntax suitable for the specific analysis tool).
The adapters and the Result Manager play a key role in the solution process execution, but their description is out of the aim of this work.

3. Case Study Application: A fault tolerant remote SCADA system for a robotic cell

3.1. System Description

The case study application is a fault tolerant remote SCADA system for a robotic cell which performs a palletizing process. A PLC locally supervises the process. Fig. 4 shows the configuration of the fault tolerant SCADA system of the described robotic cell installed at the University of Naples. It consists of three general purpose PCs: two

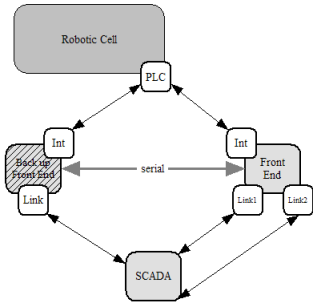

Figure 4. SCADA architecture

of them have a module interface *Int* towards the PLC onboard and act as primary and back-up front-end to the cell respectively; the third one works as remote SCADA: it demands to the local front-ends the interaction with the PLC. The remote SCADA communicates with the primary front-end over two (redundant) network links (*link1* and *link2* in Fig. 4), and with the back-up front-end over just one link (simply denoted *link* in Fig. 4).
The (active) front-end interface card driver reads and writes status and control variables from and to the PLC memory using a polling protocol. The variables are stored in the front-end shared memory accessed by the PLC driver and other front-end processes.
Remote SCADA reads from front-end the process status performing control and configuration operations if needed and visualizes a graphical representation of the process. In this paper three different classes of failure are considered when modeling the system:

link failures: If both If both *link1* and *link2* fail or back up *link* fails.

machine failures: If one of the machines fails.

timing failures: If deadline expirations of (soft) real-time occur while processes are executed on the machines.

3.2. System Model

Analyzing, tuning and recognizing bottlenecks of the described system is a very hard task: it is necessary to model the hardware and software behavior of highly heterogeneous components such as different tasks behaviors, operating system scheduling policies, communication protocols, concurrency among tasks, IPC mechanisms and so forth. A multi-formalism approach allows to cope with the complexity of the problem and model both qualitative and quantitative characteristics of each component of the systems by using the appropriate formalism to analyze each subsystem. The goals of our analysis are to evaluate bootlenecks and the availability of the whole system and to provide the parameters (number of control and configuration threads and the width of polling interval) to be used in the system tuning in order to guarantee that the required real-time deadlines will be met. At this aim is necessary to develop and integrate both a performability and an availability model of the system.

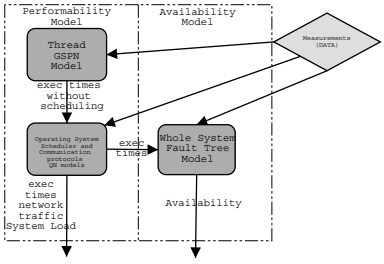

Figure 5. Modeling Strategy

As shown in Fig. 5, the performability model is used to evaluate: 1) the distribution of the response time of each thread, 2) the network traffic, and 3) the system load. Three GSPN models are used to describe the threads on the two front-end and the remote SCADA. Since PNs do not provide native mechanisms to represent queuing effects, two QN-based models are developed to represent the operating system scheduling and the communication protocols behaviors (such as TCP/IP). The service times in the QN models are obtained by solving the GSPN models.
The availability model instead is developed by using the FT formalism.

GSPN models. The GSPN model of a front-end describes four types of tasks: (1) read and write operations on PLC memory; (2) send and receive operations (to and from the remote SCADA); (3) fault prediction tasks; and (4) reconfiguration task after a fault detection. Furthermore, the GSPN model of the remote SCADA describes: (1) threads responsible for the processing of the status variables and the alarm variables to update the process graphical representation on remote SCADA (*GUI threads*); (2) communication of control and configuration signals to the active front-end; (3) requests of information about the status of the palletizing process to the active front-end; and (4) reconfiguration tasks required to reset the processes after a fault detection. For brevity's sake in the following we only report the part of the GSPN models of the front-end the points **(1)** and **(2)** of the front-end model.
The model of two concurrent tasks of the front-end is reported in Fig. 6. The first task performs the send and receive operations towards the remote SCADA system and interacts with the second task (PLC interface card driver) through read and write operations on the front-end shared memory. On the left of Fig.6 is modeled the protocol used by the first task to access to the shared memory. The middle part of the model in Fig.6 describes read-write contests in a shared memory managed by a semaphore (P_{sem}). Transitions $T_{\_read\_PLC}$ and $T_{\_write\_PLC}$ represent read and write operations from and to the PLC memory respectively. Transitions $T_{\_prep\_pkt}$ and T_{send_pkt} represent the operations of sending data and feedback information to the remote SCADA.
Notice that read-write conflicts affects the execution time of each thread. In Fig. 7 the model of two concurrent thread on the remote SCADA is reported . The first thread(on the top of the figure) sends requests to get the process status variables from the front-end and stores them in the remote SCADA shared memory. The second thread(on the bottom of the figure) reads from this shared memory and upgrades the graphical representation of the pallettizing process on the remote SCADA Graphical User Interface.

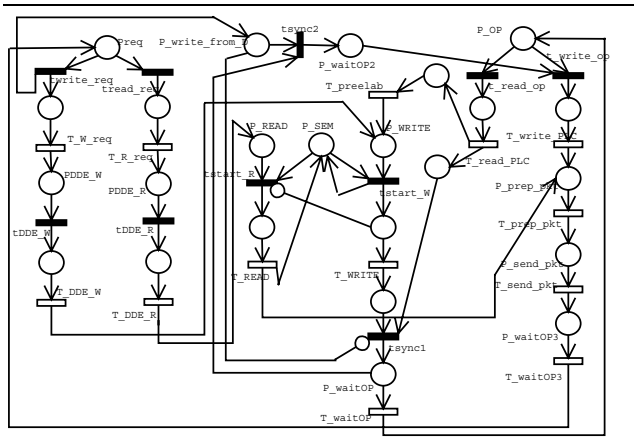

Figure 6. GSPN model of two front-end threads

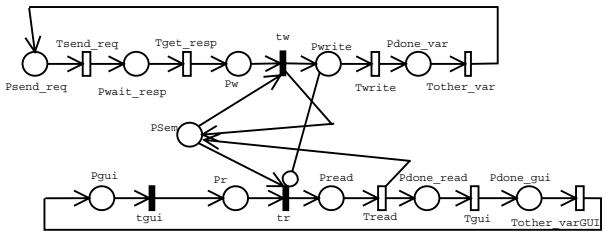

Figure 7. GSPN model of two SCADA threads

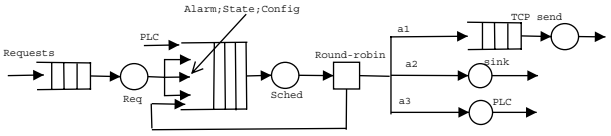

Figure 8. Front End QN model

QN models. The QN model of the front-end is reported in Fig. 8. The remote SCADA sends requests to the front-end to : a) get process alarm variables; b) get process status variables; c) perform control and management operation on the process. These three kind of requests are managed by different threads that are scheduled by the operating system together with the PLC interface card driver. The scheduling policy used by operating system is a preeemtive round-robin one (represented in Fig.8 by the rectangle at the left of server *Sched*). In Fig.8 a_k $k \in \{1, 2, 3\}$ represents the percentage of scheduled operation requiring : (a_1) send opera-

tions towards remote SCADA; (a_2) tasks that do not ask for further operations; (a_3) PLC operations. Communication-related queues (*Req and TCPsend*) are modeled by a **M/M/1** queue. A time slice q is associated to the round-robin preemptive scheduler queue. In the following :

- n_i*q+h_i (h<q) is the time units used by thread *i* to accomplish its task;
- t_0 is the service time of scheduled activities requiring a whole time slice (all (sub)task preempted);
- $t_{s_i}, i \in 1, ..ntask$ the service time of scheduled activities requiring less than a time slice to terminate;
- p_i , $i \in 0, ..ntask$ the relative frequency of each (sub)task.

It is possible to prove [9] that our model can be studied as an **M/G/1** queue with :

$$t_s = \sum_{i>=0} t_{s_i} * p_i; \qquad \sigma_s^2 = \sum_{i>=0} (t_{s_i} - t_s) p_i; \qquad C_s^2 = \frac{\sigma_s^2}{t_s^2} \tag{1}$$

The proof is omitted owing the lack of space.

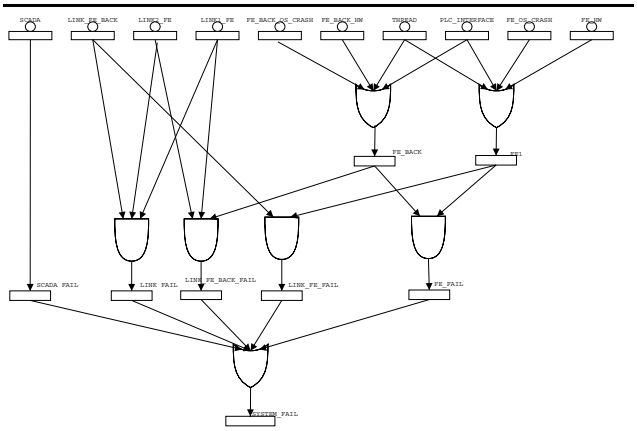

Figure 9. Fault Tree of the whole system

Fault Tree model. Finally, the fault tree model of the whole system is reported in Fig.9. The rectangles with a circle on the top (*basic events*) represent system component faults and are characterized by an occurrence probability. Basic Events are combined by using OR and AND gates. The results of these operations are simply called *events* and represents subsystem faults. The latest composition level have only one event (*top event*) and represents the whole system fault probability. At system level, in Fig.9, there a fault occurs if at least one of subsystem of the previous level faults. The following faults are considered in Fig.9:

a) hardware and operating system machine (*FE_HW, FE_OS_CRASH, PLC_INTERFACE, FE_BACK_HW, FE_BACK_OS_CRASH*);
b) Link (*LINK_FE_BACK, LINK1_FE, LINK2_FE*);
c) Thread (*THREAD*) due to three consecutive deadline overcomings.
Notice that the fault probability of the basic event at point c) is obtained from the performability model reported in Fig.5.

4. Solution Process Definition

Modeling the SCADA system by the OsMoSys framework requires that the GSPN, QN and FT models described in Section 3 are built by using the DrawNET++ GUI. Since the models of the system components are general enough to be reused it is useful to create new model classes from whom model objects can be instantiated and used as building blocks in different configurations. As an example, Fig. 10 shows the DrawNET++ GSPN model class which has the same structure of the model in Fig. 6. The

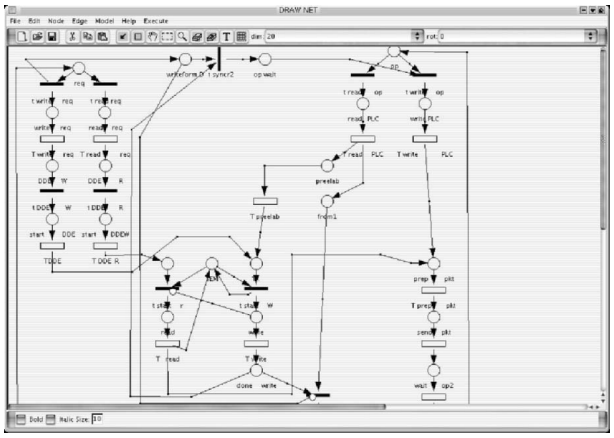

Figure 10. OsMoSys GSPN front end model

final model of the SCADA system depicted in Fig 11 is built by using a bridge metaclass, in which composition operators are defined. In Fig 11 the nodes represented by squares are sub-models and the nodes represented by circles or diamonds are instances of two different types of connection operators. The final model is obtained by integrating the GSPN and QN models by the operator instances named $operator1$, $operator2$ and $operator3$ and by realizing a communication between the performability model and the availability model through the operator instances named $operator4$ and $operator5$. Indeed, the operation needed to compose a GSPN model and a QN model is a data exchange. Instead the operation needed to integrate

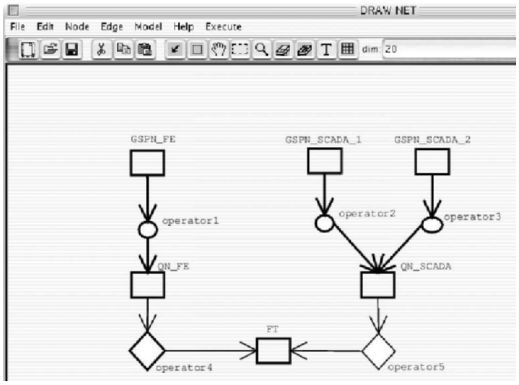

Figure 11. OsMoSys system model

the availability model is more complex. It consists of a data exchange after the solution of the associated QN model and the evaluation of a joint probability starting from the QN model results.

4.1. Definition of the Fault-Tolerant remote SCADA System Solution Process

In this Section we describe the solution process used to solve the multi-formalism model of the SCADA system. The graphical formalism used in Fig.12 to describe the

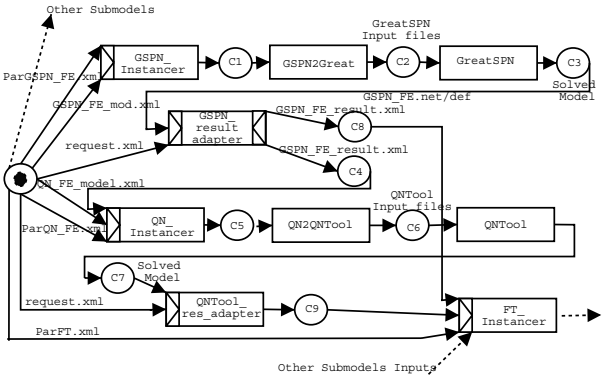

Figure 12. Remote SCADA solution process

workflow process is defined in [12]. A Square Box represents an activity, while arrows represents transitions. Circle are used to store token generated by transition activations as they were arcs of a PN, where activities may be treated as transitions. In Fig. 12 only a part of the whole workflow process is represented because of lack of space. At the Start Point, the XML model description is passed to the process activities. Here we use the GreatSPN [1] to solve the GSPN models and Sharpe [11] to solve the FT model. A preprocessor application translates the XML representation of the GSPN models to the GreatSPN format. It is instanced by the GSPN2Great activity. The activity *GreatSPN* invokes the tool to solve the GSPN sub-model. Then the GreatSPN adapter ($GSPN\_result\_adapter$) is invoked to translate the solution provided by the tool in the OsMoSys XML syntax. The obtained results ($GSPN\_FE\_result.xml$) are used to solve the QN sub-model: a post-processor instantiated by the $QN\_Instancer$ activity inserts the parameters calculated by solving the GSPN sub-model into the QN sub-model and the preprocessor instantiated by the activity QN2QNTool translates the XML format of the resulting DrawNET++ model to the input format of the QN solver. The activity QNTool invokes the tool to solve the front-end QN model. The QN solver adapter $QNTool\_res\_adapter$ translates the solution in the XML format used as input data for the Fault Tree sub-model. The workflow described in

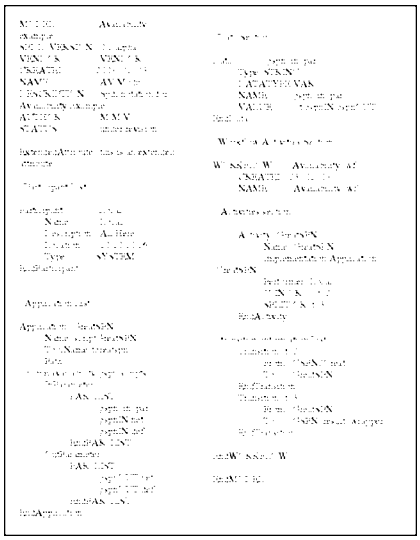

Figure 13. SPDL example

Fig. 12 is expresses the procedural aspect of a solution process, that is fully specified by means of SPDL, a proper language based on the XML Process Definition Language (XPDL) [16]. Fig. 13 presents the SPDL section (pruned of all XML tag) of the SCADA solution process which defines the workflow reported in Fig. 12.

5. Experimental Results

Table 5 reports some (partial) results obtained by solving the multi-formalism model in Section 4. For brevity we only report the results obtained by solving the performability model.

Table 1: Some Experimental Results

Parameter	*Model*	*Measured on System*	*Variation*
$t_{sAL,sST,sC}$	3.27msecs	3.39msecs	3.8%
t_{sGUI}	20.03msecs	20.83msecs	4.0%
t_r	15.89msecs	16.62msecs	4.6%
Deadline expiration Probability(t_r)	0.004	—-	—

The values of $t_{sAL,sST,sC}$ represents the mean values of services times for the scheduler *Sched* in Fig.8 of the remote SCADA (Alarm, State and Config) requests. These values are the outputs of the GSPN submodel reported in Fig.6. Likewise $t_{sAL,sST,sC}$, t_{sGUI} represents the mean value of service times for GUI operations for remote SCADA and is obtained by solving the model in Fig.7. Finally t_r represents the mean value of the response time to requests sent by the remote SCADA to the active front-end. These requests are scheduled under (soft)real-time constraints: the deadline for a request operation is set at 30 msecs. The probability of a deadline expiration for a request operation is evaluated on the basis of the distribution of the requests response time obtained by solving the submodel *QN_SCADA* in Fig.11.

6. Conclusions and Future Work

In this paper we have discussed the application of workflow management principles to the multi-solution problem which arises when using a multi-formalism approach to the modeling of distributed dependable systems. We propose to express the solution process needed to solve a multi-formalism model by using workflow concepts and technologies. The main advantages of the proposed solution are the flexibility and the extensibility of the framework, that allows to integrate a wide number of modeling formalism and analysis/simulation tools, and the possibility of defining libraries of reusable models. At the state, the OsMoSys architecture only addresses the WfMC Interface 1 (Process Definition Interchange) [16]. Some works will be done to make it compliant with Interface 2 (Client Application Interface) and Interface 3 (Invoked Application Interface).

References

[1] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. Greatspn 1.7: Graphical editor and analyzer for timed and stochastic petri nets. *Performance Evaluation*, 924(1-2):47–68, November 1995.

[2] G. Ciardo, R. Jones, A. Miner, and R. Siminiceanu. Smart: Stochastic model analyzer for reliability and timing. *Proc. of the International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 29–34, September 2001.

[3] J. de Lara and H. Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. *Proc. of the European Joint Conference on Theory And Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE), LNCS*, (2306):174–100, April 2002.

[4] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Risavi, J. Doyle, W. Sanders, and P. Webster. The mobius framework and its implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.

[5] E. Engstrom and J. Krueger. Building and rapidly evolving domain-specific tools with dome. *Proc. IEEE International Symposium on Computer-Aided Control Systems Design*, pages 83–88, 2000.

[6] G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. Drawnet++ : Model objects to support performance analysis and simulation of complex systems. *Proc. of Performance TOOLS*, (2324):233–238, April 2002.

[7] M. Gribaudo and A. Valente. Framework for graph-based formalisms. *Proc. of the first International Conference on Software Engineering Applied to Networking and Parallel Distributed Computing*, pages 233–236, May 2000.

[8] G. Karsai, G. Nordstrom, A. Ledeczi, and J. Sztipanovits. Specifying graphical modeling systems using constraint-based meta models. *Proc. IEEE International Symposium on Computer-Aided Control System Design*, pages 89–94, 2000.

[9] F. Moscato. Realizzazione e validazione di un sistema remoto per il controllo di una cella robotizzata (in italian). *Master's thesis, Universita' di Napoli Federico II*, May 2002.

[10] M. Remelhe. Simulation and visualization support for user-defined formalisms using meta-modeling and hierarchical formalism transformation. *Proc. IEEE International Conference on Control Applications*, pages 750–755, 2001.

[11] R. Sahner, K. Trivedi, and A. Puliafito. Performance and reliability analysis of computer systems, an example-based approach using the sharpe software package. *Kluwer Academic*, November 1995.

[12] W. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, System and Computers*, 8:21–66, 1998.

[13] H. Vangheluwe. Devs as a common denominator for multi-formalism hybrid systems modelling. *Proc. IEEE International Symposium on Computer-Aided Control System Design*, pages 129–134, 2000.

[14] H. Vangheluwe, J. de Lara, and P. Mosterman. An introduction to multi-paradigm modelling and simulation. *Proc. of the AI, Simulation and Planning in High Autonomy Systems Conference*, pages 9–20, April 2002.

[15] WFMC. *Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011)*. 1999.

[16] WFMC. *Workflow Management Coalition: Interface 1 - Workflow Process Definition Interchange - XML Process Definition Language (XPDL), (WFMC-TC-1025)*. 2002.