Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

MODEL-DRIVEN PERFORMANCE PREDICTION OF HLA-BASED DISTRIBUTED SIMULATION SYSTEMS

Daniele Gianni

European Space Agency
Keplerlaan, 1
N-2200 AG, Noordwijk, The Netherlands

Paolo Bocciarelli

University of Rome TorVergata
Via del Politecnico, 1
I-00133 Rome, Italy

Andrea D'Ambrogio

University of Rome TorVergata
Via del Politecnico, 1
I-00133 Rome, Italy

ABSTRACT

Performance models offer a convenient tool to assess design alternatives and predict the execution time of distributed simulation (DS) systems at design time, before system implementation. Currently, performance models are to be manually developed and the related extra effort often becomes the limiting factor for their cost- and time-effective use. In this paper, we aim to reduce this extra effort with the introduction of a model-driven method for the automated building of performance models whose evaluation provides a prediction about of the execution time of a distributed simulation system. As such, the method contributes to bring software performance engineering techniques into the distributed simulation system lifecycle. In particular, we show how the SysML-based specification of the system to be simulated and the design documents of the DS system can be used to derive the topology and the parameters of a performance model specified according to the Extended Queueing Network formalism.

1 INTRODUCTION

Computer simulation is a key tool for systems engineering in many domains as it provides engineers with virtual representations that can be used to predict and validate the design alternatives at design time, as well as to analyze system behavior at operation time. However, the ever growing complexity of modern systems may often require computational capabilities that are not available through local simulation approaches. These capabilities can be often achieved by Distributed Simulation (DS) approaches, which overcome the limitations of the local approaches by running loosely coupled partitions of the simulators over an internetworked set of hosts. However, DS approaches require substantial technical know-how and a considerable development effort for implementing the DS system (Gianni et al. 2011). Moreover, a DS may not necessarily yield reduced execution times with respect to a local simulation, as the synchronizations and communications over the network can become the bottleneck of the DS system. Before investing in the development of a DS system, it is therefore convenient to predict the DS performance and evaluate whether the DS approach can lead to reduced execution times.

Predictive performance engineering methodologies have been introduced to estimate the execution time of a DS system, thus supporting the evaluation of design alternatives and minimizing the risk that the DS system will not meet the expected performance (Chu-Cheow et al. 1999; Perumalla et al. 2005; Ewald

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

et al. 2006; Gianni et al. 2010). However, barriers to the wide adoption of these methodologies still exist. Specifically, considerable performance engineering expertise is still needed as well as a manual effort to derive the performance models from the simulated system and DS system design documentation. Lowering these barriers, and make these predictive methodologies cost- and time-effective, is therefore a key issue to bring performance engineering practices into the DS system lifecycle. These barriers can be mitigated by reducing the required know-how—for example automating the derivation of performance models—and by limiting the production of extra artifacts—for example reducing the design effort of the DS system by reusing artifacts produced by engineering processes of the system to be simulated.

In this paper, we aim to achieve above mentioned objectives by introducing a model-driven method for the derivation of predictive DS system performance models from SysML models of the system to be simulated and from UML models of the DS system. As such, the method relieves simulation engineers from the need of acquiring performance engineering know-how and from the effort related to the production and validation of the performance model. The method is based on performance models specified in the Extended Queuing Network (EQN) formalism (Boolch et al. 2006), and is originated from our previous work in the areas of model-driven simulation engineering (Bocciarelli et al. 2012) and simulation performance engineering (Gianni et al. 2010). The method consists of a sequence of steps that, starting from models of both the system to be simulated and the DS system, derive the topology and the parameters of an EQN model representing the time-performance of the DS system. The EQN performance model is then automatically implemented and executed to obtain predictions of the execution time of the DS system.

The paper is structured as follows. The background section recalls notions and concepts upon which the methodology is defined. The method section outlines the steps for the derivation of the EQN performance model. Finally, the example section presents the derivation of a predictive EQN performance model for a simplified manufacturing system and the model validation through comparison with a simulator built on top of the HLA (High Level Architecture) distributed simulation technology (IEEE 2000).

2 BACKGROUND

The paper contribution is defined using concepts and principles from Model-driven engineering (MDE) (Schmidt 2006), and in particular from Model-driven Architecture (MDA) (OMG 2003) and Model-driven Performance engineering (MDPE) (Balsamo et al. 2004), as illustrated in the following two subsections, respectively.

2.1 Overview of MDE and MDA

MDA is the most prominent implementation of MDE principles, which define an approach for developing systems by use of formal abstract models that are decoupled from the peculiarities of the implementation platforms, and the related technical complexity. The MDE approach is based on a set of iterative model transformations between different levels of abstractions, from the abstract specification down to the implementation into a specific platform. In line with this definition, MDA defines a set of standards that support both model definition and model transformations (model-to-model and model-to-text, depending on the abstraction level). Specifically, MDA consists of the following main standards: *Meta Object Facility (MOF)*—for specifying technology neutral metamodels (i.e., models used to describe other models) (OMG 2004), *XML Metadata Interchange (XMI)*—for serializing MOF metamodels/models into XML (OMG 2007) and *Query/View/Transformation (QVT)*—for specifying model transformations (OMG 2008).

2.2 Model-driven Performance Engineering

MDPE deals with the use of MDE principles and standards in the software performance engineering context, with a specific focus on MDA standards. Software performance engineering methods are applied to introduce the so-called lifecycle validation of software systems, i.e., the ability to predict the non-functional behavior of the system before its implementation. The main idea behind MDPE is to exploit

the system models available at specification/design time not only to apply MDE-based system development approaches but also to automatically derive the performance models that provide the required predictions in terms of efficiency, reliability and/or performability. The performance model building activity, which can be time-consuming and error-prone if carried out manually, is carried out by specifying and executing automated model transformations that take as input the standard (e.g. UML) software design models and yield as output the corresponding prediction models specified by use of a given formalism (e.g., queueing-based models, petri nets, process algebras, etc.). This provides an integrated approach that can be effortlessly used to integrate lifecycle validation into MDE-based system development processes. A comprehensive survey of MDPE methods can be found in (Balsamo et al. 2004), while specific contributions related to the automated generation of queueing-based performance models can be found in (D'Ambrogio et. al 2007; Bocciarelli et. al 2008).

Specifically, in this paper both the UML and the Performance Model (PM) are instances of their respective metamodels, which are defined using MOF constructs. At metamodel level, model transformations are defined using QVT for the derivation of a PM from a UML model. Both these models are serialized in XMI documents by means of pre-defined XMI rules, which with XMI schemas, allow the validation of the respective XMI documents.

3 METHOD FOR PERFORMANCE PREDICTION OF DS SYSTEMS

The proposed method takes into account the following types of models:

- *system model*: the model of the system to be simulated by use of a DS approach, specified as a SysML model;
- *DS model*: the design model of the DS system to be implemented, specified as a UML model;
- *performance model*: the performance model of the DS system, specified as an EQN model (topology and parameters).

The method introduces a set of steps for the derivation of a performance model from both the system model and the simulation model.

Figure **1** illustrates the proposed method, which consists of the following steps:

1) system model partitioning: a manual step that partitions the system model to identify which SysML blocks are to be transformed into simulation components of the DS model (Bocciarelli et al. 2012);
2) DS model building: a manual, but potentially automated, step that takes as input the partitioned system model and derives the DS model. This model is specified as a UML model consisting of a Component Diagram, a Deployment Diagram and a set of Activity Diagrams, one for each federate of the DS system;
3) performance model building: an automated step that takes as input the DS model and yields as output the performance model, specified according to the metamodel introduced in (Bocciarelli et al. 2012), specifically for what concerns the EQN topology;
4) performance model parameterization: an automated step that takes as input the DS model and yields as output the parameters of the EQN-based performance model obtained at step 3;
5) performance model implementation: an automated step that transforms the EQN-based performance model into the corresponding implementation, by use of EQN implementation technologies – e.g., jEQN (D'Ambrogio et al. 2006; Gianni et. al 2008) or OMNET++ (www.omnetpp.org/);
6) performance model evaluation: an automated step that executes the model implementation obtained at step 5 to yield the indices for the prediction of the DS system performance, in terms of execution time.

In the rest of the paper, we focus the attention on step 3 (performance model building) and step 4 (performance model parameterization). The remaining steps are briefly discussed in the example application (see Section 4) as they are either application specific or of more conventional MDA nature.

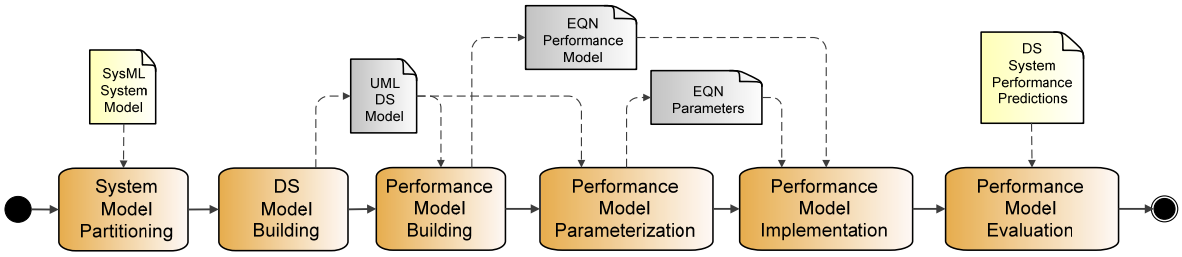

Figure 1: Overview of the method for performance prediction of DS systems.

The main objective of the following sections is to illustrate the knowledge that is embedded into the transformation patterns required to automatically obtain an EQN-based performance model from a system model specified in SysML and a DS model specified in UML. The specification of such transformation patterns into a standard MDA-based transformation language (e.g., QVT) and their serialization into XML is not covered here. The interested reader may refer to existing MDPE contributions to get a grip on how to translate the proposed transformation into a formal specification defined by use of the languages described in Section 2.2.

3.1 Performance Model Building

This step yields to the definition of the EQN-based performance model from a DS model consisting of: 1) a Component Diagram, a Deployment Diagram and a set of DS Activity Diagrams for the HLA-based time conservative federates.

The step is developed in two sub-steps: a) derivation of the EQN horizontal schema, which identifies the HW resources and their high-level interconnections, and b) derivation of the EQN performance model, which uses EQN primitives (service centers, queues, fork, join, etc.) to detail the interconnections depending on the interactions defined in the DS Activity Diagram of the generic federate.

3.1.1 Derivation of the EQN Horizontal Schema

The HW execution platform of the DS system consists of a set of hosts interconnected by a LAN, MAN or WAN infrastructure. In UML, such platform is specified by use of a Deployment Diagram annotated with the standard MARTE (OMG 2009) profile for representing the performance characteristics, such as CPU speed, network latency and bandwidth. Let's assume that the reference HW platform is described by the Deployment Diagram in Figure 2.

To complete the full specification of the HW execution platform, the diagram also indicates the software allocation of federate components (stereotyped as <<artifact>>) onto the above defined hosts, by use of dependency associations stereotyped as <<deploy>>. We limit the presentation of our method to a DS system consisting of a RTI Server, a Federation Manager and two federates (Federate 1 and Federate 2). The allocation on the individual hosts is arbitrary and is also presented to show the reasonable assumption that no host is running more than one federate.

RTI implementations, such as pRTI 1516 (www.pitch.se/pRTI1516), are commonly based on decentralized architectures, which offer improved performance with respect to the centralized ones. In decentralized architectures, the RTI Server participates actively only in the initialization phase when direct TCP/IP connections are established among the participating RTI Locals, which are incorporated into each federate. As a consequence, the RTI Server can be safely omitted for a performance analysis that aim to determine the steady-state performance of a DS system. Analogous observations can be made on Host 4 which runs the Federation Manager — a special type of federate that coordinates the federation execution (Kuhl et al. 1999).

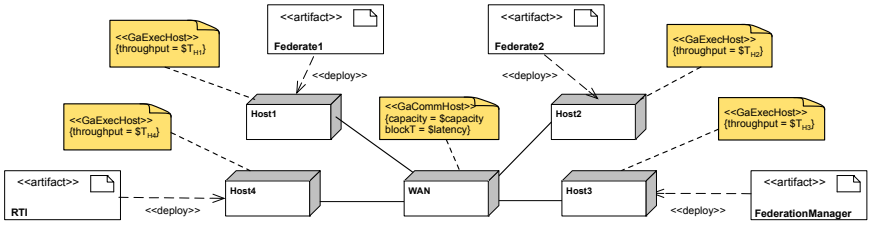

Figure 2: DS Deployment Diagram.

The Federation Manager heavily participates in the initial phase and in the final phase, which can be safely omitted from our analysis. During the federation execution, the Federation Manager also regulates the advancement of the global time in the DS. However, this operation has minimal impact on the DS system performance if the Federation Manager is run by a dedicated host that 1) is also physically located between Federate 1 and Federate 2, and 2) presents an identical delay time with respect to Host 1 and Host 2. Differently, Host 1 and Host 2 are central to the performance prediction and can be characterized at several levels of details. Each host is provided with its own resources: CPUs, main memory and data storage devices. For the sake of simplicity, the CPU can be assumed to be the only local resource used in each host. Main memory blocks can be reasonably assumed to be of infinite capacity, implying that the simulator would never incur in execution suspension while waiting for the release of memory locations. Moreover, the main memory can be assumed to introduce a delay that can be seamlessly incorporated in the CPU performance modeling. Similarly, data storage devices can also be omitted for simulation systems that do not store any intermediate data during the simulation execution. However, it is important to remark that these assumptions are only simplifications useful to ease the illustration of the method, which can be easily extended to incorporate additional resources. Basing on such considerations, the Deployment Diagram is transformed into a horizontal schema that defines the resources of interest and their high-level interconnections, that in this case is constituted by two computational nodes (e.g., Host1 and Host2) interconnected by a node that represents the network.

The interconnections can be further detailed depending on the characteristics of the interactions among the federates. As afore mentioned, we restrict our analysis to the IEEE HLA Standard and to the case of conservative execution using DS implementation technologies in which only safe events are processed, such as the SimArch technology (https://sites.google.com/site/simulationarchitecture/simarch) (Gianni et al. 2011).

3.1.2 Derivation of the EQN Performance Model

The horizontal schema can be further detailed by use of the Activity Diagrams to determine the complete EQN topology in which the detailed interconnections and the jobs flow are illustrated. Considering the decentralized architecture of the RTI, and the marginal role of the Federate Manager in the federation execution, we focus on the definition of the Activity Diagrams for Federate 1 and for Federate 2. Both federates share the same algorithmic structure for the conservative synchronization, and therefore we only illustrate Federate 1 Activity Diagram (see Figure 3).

Before detailing the diagram, it is important to remark that this Activity Diagram represents the software statements flow in the form of a generic template that can be further refined to incorporate application-specific simulation and multi-resolution modeling of the simulation software. As a consequence, both Federate 1 and Federate 2 are illustrated by Activity Diagrams that are structurally equivalent and that differ only in terms of parameters annotated by use of the UML MARTE profile. Moreover, the diagram assumes that the RTI Server is already running and that the Federation Manager has been already launched and is waiting for the federates to join (Kuhl et al. 1999).

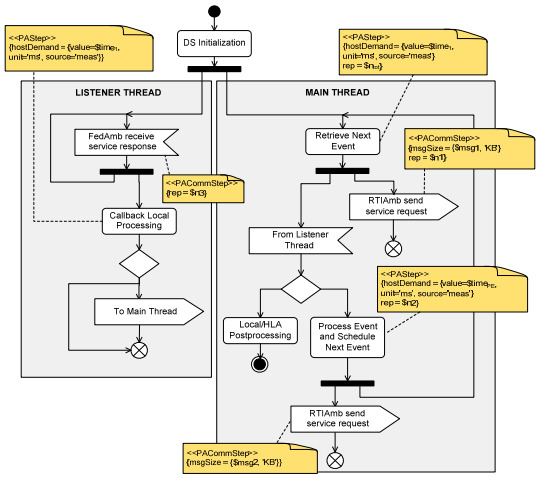

Figure 3: DS Activity Diagram.

The diagram begins with the "DS Initialization" action node, which represents the computations needed for the initialization of the local environment and for the HLA environment, respectively. Depending on the specific characteristics of the federate, such action node can incorporate one or more local and HLA statements. Specifically, adopting the federation lifecycle defined in (Kuhl et al. 1999), the "DS Initialization" action node embeds the invocation of RTI services to join the federation, to set the data publishing and subscription preferences, and to register for the synchronization points (populating and ready-to-run). However, such invocations are carried out at initialization time only, and therefore do not affect the steady-state performance analysis.

After completing the initialization activity, Federate 1 proceeds with the execution of the simulation logic and reaches a fork control node. This node creates two threads: the Main Thread (MT), representing Federate 1 initiated computations, and the Listener Thread (LT), representing Federate 2 initiated computations.

The MT block consists of a cycle of local computations, exit test decision node, and RTI services invocations. Using conservative time advancement, the MT proceeds with the "Retrieve Next Event" action node, which retrieves the next event time. Such a time is used to issue a service request to the RTI Ambassador, by use of the "RTIAmb send service request" send signal action node. The request enquires the RTI for delivering all the distributed events scheduled at any time lesser than the one retrieved or a TimeAdvanceGrant notification if no event is available before the retrieved time. While waiting for this notification, the MT suspends on the "From Listener Thread" accept event action node. The MT resumes when the LT issues the "To Main Thread" send signal action node. If the event received is the simulation end event, the MT terminates the cycle and proceeds to the execution of the "Local/HLA Postprocessing" action node, which leads the entire DS system to the original state before the activation of the DS execution, including the resignation and destruction of the federation. Differently, if the received event is not the simulation end event, the MT proceeds with the "Process Event and Schedule Next Event" action node. In the reference DS implementation technology, i.e., SimArch, distributed events are transparently scheduled into the local event list, and therefore this event is guaranteed to be the one with the least time stamp. The event processing requires internal computation, such as the event retrieval from the list as well as the execution of the associated logic. In turn, the simulation logic may require scheduling internal events, which can be obtained with statements performed internally to Host 1, or distributed events, which can be obtained invoking the "RTIAmb send service request" send signal action node.

The LT block consists of a cycle for the listening of incoming RTI Callbacks. The cycle starts with the activation of a listening thread that remains suspended on the "FedAmb receive service response" accept event action node. Such a response may be either a NextEventRequestAvailable signal or a TimeAd-

vanceGrant signal. The former is received upon delivery of an HLA interaction containing an event from Federate 2. Differently, the latter is received when no HLA events are available until the specified time, and therefore the Federate is granted the right to safely advance to the specified time. In both cases, the handling of the callback requires a local processing, which has been identified with the "Callback Local Processing" action node. Once the callback is handled, the LT proceeds towards a decision node that evaluates whether the "To Main Thread" send signal action node needs to be delivered to the MT.

The nodes of the Activity Diagram are annotated by use of the MARTE profile to specify the federate-specific parameters, such as the number of iterations (*rep* tag of the <<PAStep>> and <<PACommStep>> stereotypes), the service demand (*HostDemand* tag of the <<PAStep>>) and the sizes of messages exchanged (*msgSize* tag of the <<PACommStep>> stereotype). Such parameters are used to obtain the parameters of the EQN-based performance model, as illustrated in Section 3.2.

The structural properties of the Activity Diagram are instead used to generate the detailed EQN topology according to the UML-to-EQN model transformation illustrated in (Bocciarelli and D'Ambrogio 2012). The original transformation has been here revised and extended to introduce a novel mapping rule, which has been specified to properly manage the asynchronous communications among federates. More specifically, in its execution, each federate interacts with other federates according to a communication paradigm based on signal (e.g., events) exchange. In this respect, a federate may stop its execution until it receives from the RTI the notification of events occurred at other federates side. Similarly, a federate may need to notify, via RTI, an event to other federates that wait for its occurrence.

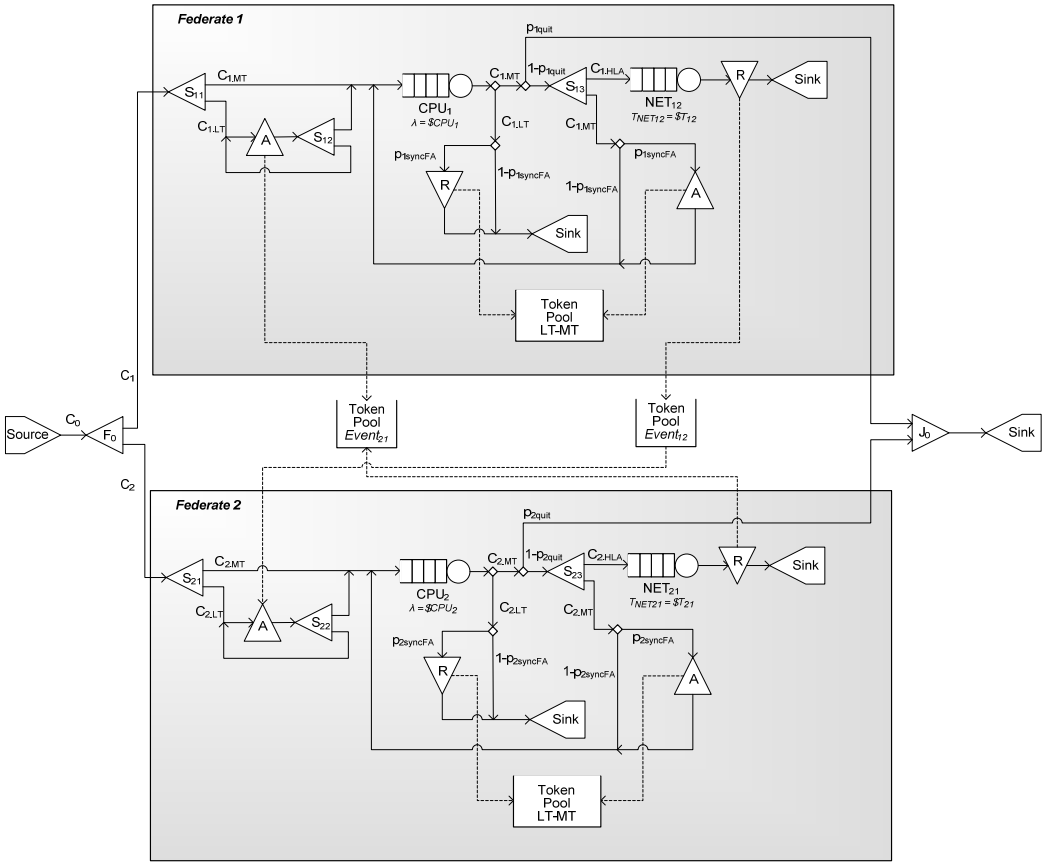

Figure 4: EQN Topology.

The rationale of the novel mapping rule can be summarized as follows: in the source UML Activity Diagram, *send signal* and *accept event* action nodes are used to represent the notification of an event and

the execution suspension while waiting for an event occurrence, respectively. For each pair of *send signal* and *accept event* action nodes related to the same event, the novel UML-to-EQN mapping rule introduces in the corresponding target EQN model a pair of allocate-release nodes and a token pool.

A federate that waits for an external event is thus mapped to an allocate node that forces a job to wait until the related token is available in the token pool. Similarly, a federate that notifies an event is mapped to a job that passes through a release node, making the corresponding token available in the token pool.

The so extended UML-to-EQN model transformation has been executed to obtain the topology of the EQN-based performance model illustrated in Figure 5.

The model is split into two sub-models, one for each federate, and includes three job classes for each federate, one for the MT (C*i*.MT), one for the LT (C*i*.LT) and one for RTI service requests (C*i*.HLA). The routing of jobs of a given class is directly mapped from the sequence of nodes that are visited in the corresponding Activity Diagram.

The next section illustrates how to derive the numerical parameters that are to be associated to the resources (i.e., service times) and routers (i.e., routing probabilities) of the EQN in Figure 5. The derivation consists in a standard procedure illustrated in (Schmidt 2006).

3.2 Performance Model Parameterization

The performance model parameters are obtained by use of the MARTE annotations specified on both the Deployment Diagram and the Activity Diagram of each federate. In particular, the following parameters are to be determined: t_{CPU1}, t_{CPU2}, t_{Net12}, t_{Net21}, p_{1QUIT}, p_{2QUIT}, $p_{1SYNC\text{-}FA}$, and $p_{2SYNC\text{-}FA}$.

The t_{CPU1} and t_{CPU2} times can be evaluated using statistical inference methods on existing prototypes of the DS software. Their direct estimation from the statement description and the CPU performance may require advanced statistical approaches that are outside the scope of this paper. Basing on previous experience and considering the large number of CPU operations that concern insertion and retrieval of elements in event list, it can be safely assumed that the t_{CPU1} and t_{CPU2} are normally distributed, with $\mu_1=T_{CPU1}$ and $\mu_2=T_{CPU2}$, respectively, and $\sigma_1= \sigma_2=1$. In our approach, T_{CPU1} and T_{CPU2} are to be expressed as multiple of the reference processing unit T_{CPU}, which identifies the computation needed for executing the base processing unit (e.g., inserting or extracting an event in the events list). With this simplification, the statistical representation of the CPU service time is obtained as a stochastic function of the T_{CPU} parameter, according to what annotated by use of the *HostDemand* tag of the <<PAStep>> in the Activity Diagram. The annotated values are given as input to a standard graph reduction algorithm that obtains the composite service demand value for each identified job class.

The t_{Net12} and the t_{Net21} parameters can be assumed to be distributed with a k-Pareto distribution of probability, where k=4 (Wei and Jingsha 2007). For these distributions, the T_{Net12} and the T_{Net21} parameters can be determined by use of the following formulas:

T_{Net12}= $delayTime_{12}$ + $averagePacketLength_{12}/bandwidth_{12}$

T_{Net21}= $delayTime_{21}$ + $averagePacketLength_{21}/bandwidth_{21}$

where the delayTime is obtained from the *blockT* tag of the <<GaCommHost>> stereotype in the Deployment Diagram, the averagePacketLength from the *msgSize* tags of the <<PACommStep>> stereotype in the Activity Diagram and the bandwidth from the *capacity* tag of the <<GaCommHost>> stereotype in the Deployment Diagram.

The probabilities p_{iQUIT} (i=1..2) and $p_{iSYNC\text{-}FA}$ (i=1..2) are derived from the Activity Diagrams of Federate 1 and Federate 2. The probability $p_{iSYNC\text{-}FA}$ can be computed using the expression: $p_{iSYNC\text{-}FA} = \$n_2/n_{HLAR}$, where $\$n_1$ is the number of repetitions annotated onto the initial "RTIAmb send service request" send signal action node and n_{HLAR} is the number of all HLA service request invocations, i.e., $n_{HLAR}=\$n_1+\$n_2$. This definition of $p_{iSYNC\text{-}FA}$ inherently assumes that the lookahead value is chosen optimally from the system model specification, and therefore the number of suspensions of main thread coincides with the number of time advance requests. Variations of the lookahead value will directly affect the value of $p_{iSYNC\text{-}FA}$. However, it is important to note that the steady-state conditions, in which the DS system is assumed to operate, contribute to reduce the dependency of the aforementioned parameters from

the DS performance variability deriving from the lookahead value, due to the uniform characteristics of the workload on the individual hosts.

p_{iQUIT} is similarly computed from the Activity Diagram as $p_{iQUIT} = 1/\$n_{tot}$, where $\$n_{tot}$ is the number of repetitions annotated onto the "Retrieve Next Event" action node, i.e., the total number of iterations in the MT. p_{iQUIT} represents the stochastic condition for the simulation termination and should not be confused with the steady-state assumption for the DS system.

4 EXAMPLE APPLICATION

This section illustrates the application of the above defined method to the derivation of a performance model for a DS system of a manufacturing system, including the parameterization. The considered example is simple but effective enough to illustrate the application of the proposed method.

4.1 System Model

The manufacturing system receives raw materials and produces processed goods. The system consists of three Service Stations: Station 1 is the main service machine implementing the production chain; Station 2 and Station 3 are support machines for the maintenance of the production tools in Station 1. The system internal structure that can be specified by use of a SysML Internal Block Diagram, has been omitted for the sake of brevity.

In our scenario, we assume a continuous and constant incoming flow of raw materials which are processed by a set of tools within the Service station. These tools need occasional maintenance, which can be of two types: engine or belt maintenance. In the case of engine maintenance, the tool is routed towards Station 2. Differently, in the case of belt maintenance, the tool is routed towards Station 3.

The system specification includes the details of all the procedures needed for the maintenance of a tool, in both cases of engine and belt maintenance. In the real system, Stations 2 and 3 operate according to a FIFO enqueuing policy. For each incoming tool, a number of maintenance operations may be required depending on unpredictable conditions. The operations can be identified by performing a set of test procedures on the tool. In the system model, these procedures are represented with a detailed SysML Activity Diagrams describing all the activities that are to be performed on the tool, appropriately annotated to specify the required times. However, in our case study, the procedures specification can be safely omitted and the related operations can be represented in terms of the required maintenance time, which can be synthesized by random distributions.

4.2 DS Model

The DS platform is specified by use of a Deployment Diagram illustrating the properties of the distributed platform and the allocation of the system components onto the individual nodes. In line with the method description, the Deployment Diagram of Figure 2 has been annotated with the properties of the actual hosts and network, as described in Table 1.

Table 1: Deployment Parameters.

Parameter	MARTE parameter	Value
Bandwidth	$capacity	94 KB/s
Delay Time	$latency	20 ms
T_{CPU1}	$1/\$T_{H1}$	10 ms per processing unit
T_{CPU2}	$1/\$T_{H2}$	10 ms per processing unit

In addition, we have partitioned the system model into two federates: Federate 1, which simulates Station 1, and Federate 2, which simulates Stations 2 and 3, with Federate 1 allocated onto Host 1 and Federate 2 allocated onto Host 2. In line with the aforementioned consideration on the lookahead value, we have derived the maximum lookahead by identifying the critical path in the event chains across the simulation

components on Federate 1 and Federate 2, in particular considering the truncated exponential and normal distributions that simulate the delays introduced by the service stations.

4.3 Example Performance Model

The performance model is derived by use of what we defined in Section 3.1 and Section 3.2. T_{CPU1} and T_{CPU2} can be assumed to be equivalent to the processing time for an individual processing unit, which from previous experiments has been set to T_{CPU1}=T_{CPU2}=10 ms (Gianni et al. 2010).

In SimArch and pRTI 1516 simulations, measurements have yielded to an average packet size of ca 90 bytes, which combined with the deployment parameters in Table 1, leads to T_{Net12}= T_{Net12}=21.5 s.

Finally, the routing probabilities are determined by identifying the values of the $n1, $n2 and $n3 parameters for each federate. The values of such variables can be statically determined using established practices for the determination of average arrival rates at each center. More empirical techniques can use SysML model animations which can either incorporate counters or generate simulation traces for offline analysis. For an operational period of 60 months, estimations yielded to the values in Table 2.

Table 2: Values of the n-variables.

MARTE Parameter	Federate 1	Federate 2
$\$n_1$	163	208
$\$n_2$	741	580
$\$n_3$	208	163
$\$n_{tot}$	949	743

From the values in Table 2, the routing probabilities are calculated and their values are shown in Table 3, for each federate.

Table 3: Routing probabilities.

Variable	Federate 1	Federate 2
p_{QUIT}	0,001	0,001
$1\text{-}p_{QUIT}$	0,999	0,999
$p_{SYNC\text{-}FA}$	0,82	0,74
$1\text{-}p_{SYNC\text{-}FA}$	0,18	0,26

4.4 Preliminary Validation

The validation process has been carried out in four steps: 1) the implementation of the EQN-based performance model to get the numerical solution; 2) the development of a DS for the system model; 3) the identification of the metrics for the numerical comparison; and 4) the comparison among predictive analysis results and measures taken on the DS system.

Step 1 has been implemented by use of a model-to-text transformation that directly maps the EQN elements with the language primitives of jEQN, a domain-specific language for the specification, implementation and execution of EQN models (Gianni and D'Ambrogio 2008). In Step 2, we have used the above mentioned SimArch technology for the rapid prototyping of the DS model (Gianni et al. 2011). At Step 3 we have identified the total execution time as the metric for the numerical comparison between the results provided by the actual DS system and the predictions obtained from the execution of the performance model. Finally, Step 4 consisted of two substeps: 4.1) DS execution and measurements collection in the deployment settings defined in Table 1 and Figure 2; 4.2) evaluation of the performance model experiment, including the sampling of the predicted total execution time. To minimize the impact of correlation among the samples, we have reiterated Step 4.1 and step 4.2 for ten times, obtaining an average actu-

al execution time of 880 sec (Step 4.1) against an average predicted time of 850 sec (Step 4.2). These results are also within reasonable margins with our previous experiments (Gianni et al. 2010), and therefore confirms the preliminary validation of the proposed model-driven method for the DS performance prediction.

CONCLUSIONS

Designing distributed simulation (DS) systems that meet the specified requirements is a complex activity. Specialized know-how on software performance engineering is required and an extra effort is often needed to develop and validate ad-hoc performance models. In this paper, we have mitigated the effects of the above issues with the introduction of a method for the model-driven performance prediction of DS systems. The method defines a mechanical derivation of performance models of DS systems from model-based systems specification, thus reducing the specialized know-how, effort and time required to manually apply software performance engineering techniques. Moreover, the method fully integrates with system specifications in SysML, thus minimizing the need of developing ad-hoc artifacts for performance engineering purposes. The proposed method produces predictive performance models based on the EQN formalism. In the paper, we have shown an example application of the methodology to the derivation of a predictive performance model for a HLA-based distributed simulation of a manufacturing system. Work is in progress to fully automate the proposed method by specifying additional model transformations to produce the UML-based DS model from the SysML-based system model. Future work will also include a generalization of the method for a higher number of federates. Besides new EQN elements in the performance model, a number of additional functions may be required for a model parameterization that takes into account the implementation of the HLA synchronization and communication.

REFERENCES

Balsamo S., Di Marco A., Inverardi P, Simeoni M. 2004. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, vol. 30, n. 5, pp. 295-310.

Bocciarelli P., D'Ambrogio A. 2008. Model-Driven Performability Analysis of Composite Web Services. In Proceedings of SPEC International Performance Evaluation Workshop (SIPEW 2008). June 27-28 2008, Darmastadt, Germany.

Bocciarelli P., D'Ambrogio A., Fabiani G. 2012. " A Model-driven Approach to Build HLA-Based Distributed Simulations from SysML Models". In *Proc. of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2012)*. July 28-30 20012, Rome, Italy.

Bocciarelli P. and D'Ambrogio. A. 2012. "Automated Performance Analysis of Business Processes". In *Proceedings of the Symposium On Theory of Modeling and Simulation*, DEVS-TMS '12.

Bolch G., Stefan Greiner, Hermann de Meer, Kishor S. Trivedi: Queueing networks and Markov chains - modeling and performance evaluation with computer science applications; 2nd Edition. Wiley 2006

L. Chu-Cheow, L. Yoke-Hean, G. Boon-Ping, J. Sanjay, C. Wentong, H. Wen Jing, and H. Shell Ying 1999. "Performance prediction tools for parallel discrete-event simulation," In *Proceedings of the thirteenth workshop on Parallel and distributed simulation* Atlanta, Georgia, United States. IEEE.

D'Ambrogio A., Iazeolla G. 2003. Steps towards the automatic production of performance models of web applications, Computer Networks Journal, vol. 41(1):29–39.

D'Ambrogio A., D. Gianni and G. Iazeolla. 2006. "jEQN: a Java-based Language for the Distributed Simulation of Queueing Networks", LNCS vol. 4263/2006, In *Proceedings of the 21st International Symposium on Computer and Information Sciences (ISCIS'06)*, Istanbul, Turkey, Nov.

D'Ambrogio A., Bocciarelli P. 2007. "A Model-driven Approach to Describe and Predict the Performance of Composite Services". In *Proceedings of the Sixth International Workshop on Software and Performance (WOSP 2007)*, Buenos Aires, Argentina, February 5-8.

Ewald R., Himmelspach J., Uhrmacher A., Chen D., Theodoropoulos G. 2006. "A Simulation Approach to Facilitate Parallel and Distributed Discrete-Event Simulator Development," in *Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications*: IEEE Computer Society.

Gianni D. and D'Ambrogio, A. 2008. "A Domain Specific Language for the Definition of Extended Queueing Networks Models", *Proceedings of the 2008 IASTED Software Engineering Conference (SE08)*, Innsbruck, Austria, February.

Gianni D., D'Ambrogio A., Iazeolla G. 2010. "A Methodology to Predict the Performance of Distributed Simulations". *In Proceedings of the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2010)*. Atlanta, GA (USA), May 17-19.

Gianni D., D'Ambrogio A., Iazeolla G. 2011. "A Software Architecture to Ease the Development of Distributed Simulation Systems". *Simulation,* June.

IEEE. 2000. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - frameworks and rules. IEEE 1516.

Lavenberg S. 1983. Computer Performance Modeling Handbook, Academic Press, New York, 1983.

OMG. 2003. MDA Guide, version 1.0.1. 2003.

OMG. 2004. Meta Object Facility (MOF) Specification, version 2.0.

OMG. 2007. XML Metadata Interchange (XMI) Specification, version 2.1.1.

OMG. 2008. Meta Object Facility (MOF) 2.0 Query/View/Transformation, version 1.0.

OMG. 2009 UML profile for Modeling and Analysis of Real Time Embedded Systems, 1.0.

Perumalla K., Fujimoto R., Thakare P., S. Pande, Karimabadi H., Omelchenko Y., Driscoll J. 2005. "Performance prediction of large-scale parallel discrete event models of physical systems," in *Proceedings of the 37th conference on Winter simulation* Orlando, Florida.

Schmidt, Douglas C. 2006. *Model-driven engineering.* IEEE Computer, 39(2), February 2006.

Smith C. 1992. Performance Engineering of Software Systems, Addison Wesley, Reading, MA, 1992.

Wei Z. and Jingsha H. 2007. "Modeling End-toEnd Delay Using Pareto Distribution", in *Proceedings of the Second International Conference on Internet Monitoring and Protection*, IEEE Computer Society.

AUTHORS BIOGRAPHIES

DANIELE GIANNI is an internal research fellow at the European Space Agency. He received a PhD in computer and control engineering from the University of Rome Tor Vergata (Italy) and held research appointments at Imperial College and the University of Oxford (UK). His research interests are in the area of modeling and simulation.

PAOLO BOCCIARELLI is a postdoc researcher at the University of Rome Tor Vergata. His research interests include software and systems engineering and business process management, specifically with regards to the areas of modeling and simulation and model-driven development.

ANDREA D'AMBROGIO is associate professor at the Enterprise Engineering Department of the University of Roma TorVergata (Italy). His research interests are in the fields of model-driven software engineering, performance engineering, distributed and web-based simulation.