ArgoSPE: Model-Based Software Performance Engineering*

Elena Gómez-Martínez and José Merseguer

Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza
C/María de Luna, 1 50018 Zaragoza, Spain
{megomez, jmerse}@unizar.es

Abstract. Stochastic Petri nets (SPNs) have been proved useful for the quantitative analysis of systems. This paper introduces ArgoSPE, a tool for the performance evaluation of software systems in the first stages of the life-cycle. ArgoSPE implements a performance evaluation process that builds on the principles of the software performance engineering (SPE). The theory behind the tool, i.e. the underlying SPE process, has been presented in previous papers and consists in translating some performance annotated UML diagrams into SPN models. Therefore, ArgoSPE prevents software engineers to model with SPN since they are obtained as a by-product of their UML models. The design of the tool follows the architecture proposed by OMG in the UML Profile for Schedulability, Performance and Time specification.

Keywords: GSPN, UML, UML-SPT, software performance evaluation.

1 Introduction

Performance evaluation focusses on the analysis of the dynamic behavior of systems and the prediction of indices or measures such as their throughput, utilization or response time. Among the different formalisms used in this field, stochastic Petri nets (SPN) [1] have been proved as a very powerful one.

Concerning the performance evaluation of software systems, Software Performance Engineering (SPE) [2] proposes methods to evaluate them in the early stages of the development process. Being the Unified Modelling Language (UML) [3] a standard *de facto* for software engineers, the SPE community has adopted it to specify performance parameters in software designs, then defining the *UML Profile for Schedulability, Performance and Time Specification* (UML-SPT) [4].

Many approaches, see [5] for a survey, have arisen to derive from UML-SPT specifications, performance models based on a given modelling formalism, such as SPN. A step forward for the application of these approaches, and to contrast its maturity, should be the development of tools that support their proposals.

* This work has been developed within the projects: TIC2003-05226 of the Spanish Ministry of Science and Technology; and IBE2005-TEC-10 of the University of Zaragoza.

S. Donatelli and P.S. Thiagarajan (Eds.): ICATPN 2006, LNCS 4024, pp. 401–410, 2006.

The availability of these tools is a necessary condition for the industry in order to apply these proposals and to discover the feasibility of the SPE research field. The UML-SPT, being concerned about the problem, has proposed the main steps and modules that any SPE tool should follow, see Figure 1.

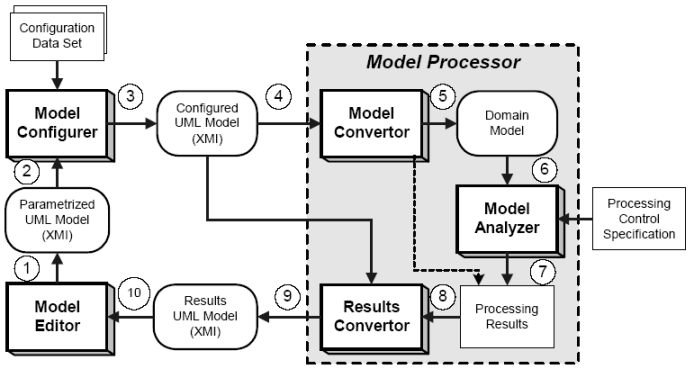

Fig. 1. Architecture proposed in the UML-SPT of OMG

Nevertheless, to the best of our knowledge, there not exist tools that fully implement the remarked SPE methodologies and follow the UML-SPT proposals. We guess that it is mainly motivated because the field is young. As a consequence, software performance prediction is still accomplished by means of *classical* performance evaluation tools such as [6, 7, 8], that introduce a gap between their methods and notations and those commonly used by software engineers. Good news are that prototypes and first attempts to implement parts of these approaches are arising in these last years.

In this paper, we present ArgoSPE, a tool for the performance evaluation of software systems in the first stages of the development process. It is based and implements most of the features given in our previous works [9, 10, 11] and gathered in [12] as a software performance modelling process. So, the system is modeled as a set of UML diagrams, annotated according to the UML-SPT, which are translated into Generalized Stochastic Petri Nets (GSPN) [13]. The UML diagrams used to obtain a performance model by means of ArgoSPE are those considered in our process: statemachines, activity diagrams and interaction diagrams. The use case diagram is taken into account in the process, but it has not been considered in ArgoSPE yet. The class and the implementation diagrams (components and deployment) are used to collect some system parameters (system population or network speed).

ArgoSPE has been implemented as a set of Java modules, that are plugged into the open source ArgoUML CASE tool [14]. It follows the software architecture proposed in the UML-SPT, see Figure 1. ArgoSPE has been used to model and analyze software fault tolerant systems [15] and mobile agents software [16].

The rest of the article is organized as follows. Section 2 presents the most interesting features of ArgoSPE, while section 3 focusses on its software architecture.

Section 4 surveys those tools prototypes, developed to analyze performance of software systems, based on the UML-SPT. Finally, conclusions and further works are exhibited in section 5.

2 ArgoSPE Features

From the user viewpoint, ArgoSPE is driven by a set of "performance queries" that s/he can execute to get the quantitative analysis of the modeled system.

We understand that a performance query is a procedure whereby the UML model is analyzed to automatically obtain a predefined performance index. The steps carried out in this procedure are hidden to the user. Each performance query is related to a UML diagram where it is interpreted, but it is computed in a GSPN model automatically obtained by ArgoSPE.

Moreover, the performance analyst, that has expertise in Petri net modelling and analysis, can use the GreatSPN tool to compute domain specific metrics using the GSPN models, that ArgoSPE generates automatically.

2.1 Queries in the Statechart Diagram

A statechart models the behavior of a class. Figure 2 depicts an example of statechart that models a Consumer class and a very simplified version of its GSPN translation.

- **State population.** This query computes the percentage of objects in each state. For example, in Figure 2, the 40% of the objects could be `Consuming` and the others `WaitingForProducer`.

  The query can be useful to detect saturated software processes or to know how an agent shares out its execution among different tasks (states). The state population is obtained by dividing the number of objects in the state among the mean number of objects that populate the class.

  For instance, in the state `WaitingForProducer`, the *State population* is computed by dividing the mean marking of place `p9` among the initial marking of the net in place `p1`.

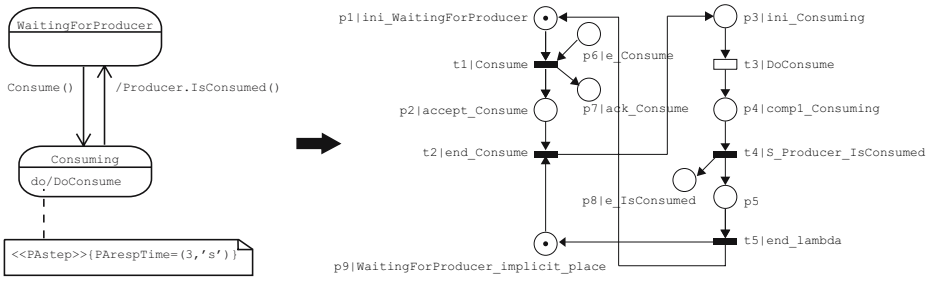

Fig. 2. Statechart and its corresponding GSPN

- **Stay time.** Represents the mean time that the objects of a class spend in each state. For each state, this value is computed by dividing the mean number of objects in it among its throughput, therefore, applying the Little's Law.

  In the example of Figure 2, the *Stay time* of the state `WaitingForProducer` represents the mean time that a `Consumer` spends waiting for consuming, and it is computed by dividing the mean marking of place `p9` among the throughput of the transition `t5`.
- **Message delay.** When the sender and the receiver of a message reside in different physical nodes, this query calculates the time spent by the message to reach the receiver's node. This value is straightforward calculated by dividing the size of the message among the network delay (see section 2.2).

2.2 Queries in the Deployment and Collaboration Diagrams

The deployment diagram specifies the execution architecture of a system. Hardware resources are represented as nodes where software components can be deployed. Moreover, the physical network connections are modeled as relationships between nodes.

- **Network delay.** Calculates the network delay (bit rate) between two non adjacent hardware resources (nodes). Given a system configuration, the network delay is useful to find the node where a new software component could be deployed, i.e. the node that minimizes the delay of the component's messages.

The collaboration diagram is an interaction diagram that focusses on how objects exchange messages. It describes the behavior of the system in a specific context (scenario).

- **Response time.** For a given collaboration diagram (system scenario), this query computes its mean response time, i.e., the mean duration of a certain system execution.

2.3 Performance Annotations

ArgoSPE uses as input a UML-SPT annotated model, i.e. UML models have to explicitly include performance characteristics. These performance annotations, defined in the UML-SPT, are made by means of the UML extension mechanisms: *stereotypes* and *tagged values*.

The *stereotypes* specify the main performance characteristics of the UML model elements, while the *tagged values* specify the attributes of the stereotypes. As an example, see the performance annotation in Figure 2, where the stereotype `PAstep` means that `DoConsume` is a computation step, while the tagged value `PArespTime` models its response time.

The UML-SPT defines a *Tag Value Language* (TVL), a subset of the Perl language, that allows to specify complex and parameterized expressions in the tagged values.

The annotations supported by ArgoSPE are those necessary to compute the proposed performance queries, see Table 1.

Table 1. Performance annotations in ArgoSPE

Annotation	**Stereotype**	**Tagged value**	**Unit**	**Model elements and Diagrams**
Activity duration	PAstep	PArespTime	ms, s, m, h	Activities in the SC and AD
Probability	PAstep	PAprob	-	Transitions in SC & AD Messages in the Coll.
Size	PAstep	PAsize	b, B, kb, kB, Mb, MB	Messages in the SC and Coll.
Network speed	PAcommu-nication	PAspeed	bps, Bps, kbps, kBps, mbps, MBps	Deployment
Population	PAclosedLoad	PApopulation	-	Class in Class diagram
Initial state	PAinitial-Condition	PAinitialState	$true or $false	State in the SC and AD
Resident classes	GRMcode	-	-	Deployment

3 Software Architecture

ArgoSPE follows the architecture proposed in the UML-SPT [4], see Figure 1. The ArgoUML [14] CASE tool works as the Model Editor, while the ArgoSPE modules implement and coordinate the Model Configurer and the Model Processor (Model Convertor, Model Analyzer and Results Convertor) functions. Figure 3 depicts the ArgoSPE menu inside the menu bar of ArgoUML.

3.1 Model Editor

The Model Editor is used to create and modify performance-annotated UML diagrams.

ArgoUML allows to model and to annotate the UML diagrams involved in the translation process [9, 10, 11]. ArgoUML, as most CASE tools, exports UML models into XMI [17] files, allowing the standard exchange of information with another tools.

From the performance-annotated UML diagrams, ArgoSPE creates a parameterized XMI file, that will be an input for the Model Configurer. This XMI file contains the modeling and performance information of: the statecharts, describing the behavior of the classes in the system; the activity diagrams, specifying the activities of the statecharts; the deployment diagram, that gathers information about physical nodes location and network transmission speed; the class diagram with information about the system workload and the collaboration diagram.

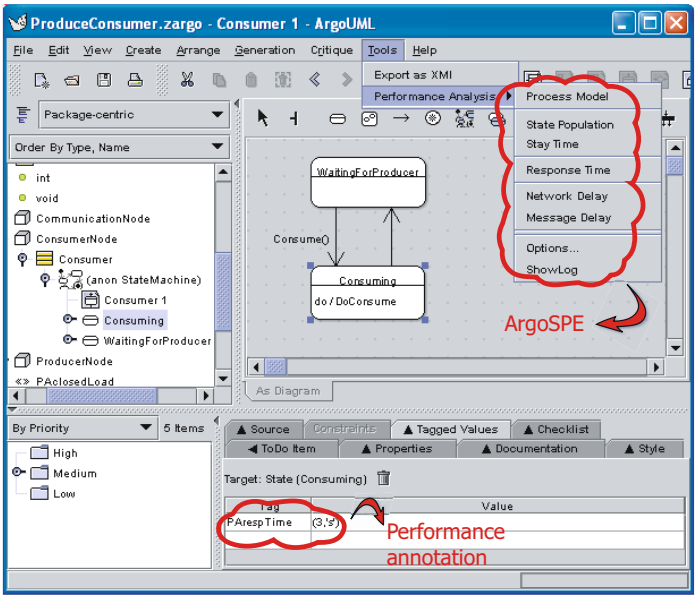

Fig. 3. ArgoSPE menu inside ArgoUML

3.2 Model Configurer

The Model Configurer functionality, see Figure 1, consists in converting a parameterized UML model in XMI format, into a configured UML model using a configuration data set. The main target is to substitute in the XMI file, the tagged values written in TVL with parameterized expressions that represent the performance annotations, for the equivalent evaluated expressions.

The first task is to parse the XMI file, obtaining a tree structure, called Document Object Model (DOM) [18]. This one is visited recursively from its root node into their children nodes to search for performance annotations. So, a list of XMI identifiers with known stereotypes is extracted. For each element in that list, a TVL expression is obtained, some of them with variables, that will be evaluated, then modifying the tree. At the same time, a symbol table is created containing the performance annotations. Finally, the tree is serialized to an XMI file.

Since the TVL expressions can contain variables, ArgoSPE prompts the user to choose a configuration file containing a configuration data set. An example of this kind of file is depicted in Figure 4.

```
#A very simple configuration file written in Perl
$value=5;
$value2=10;
$value3=($value<40)?100-$value2:100;
```

Fig. 4. A configuration file

ArgoSPE evaluates this file by invoking a Perl interpreter to get the actual value for variables and expressions. Then, for a performance annotation like `<<PAstep>>{PArespTime=($value,'s')}` the variable `$value` will be evaluated and replaced according to the configuration file (e.g. the one in Figure 4), so the evaluated expression will be `<<PAstep>>{PArespTime=(5,'s')}`. Multi-valued expressions are supported by ArgoSPE.

3.3 Model Processor

The Model Processor turns the configured model, obtained from the Model Configurer, into an analyzable model (GSPN model), analyzes it and returns the results. These tasks are respectively addressed by the Model Convertor, the Model Analyzer and the Results Convertor.

Model Convertor. The Model Convertor module encapsulates the translation process from the configured model into the target performance formalism. Therefore, in ArgoSPE it is a heavy process that implements the translation theory proposed in [9, 10, 11, 12]. The GSPN models are obtained in the GreatSPN file format [7].

The high-level algorithm implemented in the tool for the Model Convertor is illustrated in Algorithm 1. Note that it needs as inputs not only the configured XMI, as proposed in the UML-SPT, also the symbol table, that allows to speed up the translation process, but it increases the module coupling.

The lines 2 to 14 correspond to the translation process of the statecharts and its associated activity diagrams. Then, a GSPN, called `SysGSPN`, that models the whole system behavior is obtained by merging the Petri nets of the classes (line 26). The translation of the collaboration diagrams is described from lines 17 to 24. The result is a set of GSPNs, $\texttt{Scenario}_i\texttt{GSPN}$, each one modeling the scenario specified by the collaboration diagram i.

Model Analyzer. The Model Analyzer implements the performance queries described in sections 2.1 and 2.2. Concretely, the queries for the statecharts are computed using the `SysGSPN` net. While the *Response time* query for a collaboration diagram i is computed in the $\texttt{Scenario}_i\texttt{GSPN}$ net.

The Model Analyzer invokes the GreatSPN programs to get the answers to the queries. ArgoSPE currently uses the GreatSPN programs that implement analytical/numerical techniques. The use of GreatSPN simulation techniques will be considered to complement the results.

Results Convertor. The main function of the Results Convertor is to convert the results of the analysis back to the UML Model Editor, in a way that a software engineer can interpret them easily.

In the current version of ArgoSPE, the returned results are directly displayed by the Model Editor in a simple message window, then not directly in the UML models.

Algorithm 1. Model Convertor

Require: A configured XMI file and a symbol table
Ensure: A set of GreatSPN models (SysGSPN, Scenario$_i$GSPN)

```
 1: UML diagrams ← XMI file
 2: for all class c ∈ Class diagram do
 3:    if c has Statechart then
 4:       node ← Locate node in Deployment diagram(c)
 5:       A ← Activity diagrams associated with c
 6:       for all Activity ac ∈ A do
 7:          pa ← Annotations associated with ac ∈ symbol table
 8:          acGSPN[j] ← TranslateToGSPN(ac,pa)
 9:       end for
10:       sc ← Statechart associated with c
11:       pa ← Annotations associated with sc ∈ symbol table
12:       scGSPN ← TranslateToGSPN(sc,pa,node)
13:       clGSPN[k] ← Merge(scGSPN, acGSPN[])
14:    end if
15: end for
16: SysGSPN ← Merge(clGSPN[])
17: for all class c ∈ Class diagram do
18:    C ← Collaboration diagrams associated with c
19:    for all Collaboration co ∈ C do
20:       pa ← Annotations associated with co ∈ symbol table
21:       coGSPN[i] ← TranslateToGSPN(co,pa)
22:       Scenario_i GSPN ← Merge(SysGSPN,coGSPN[i])
23:    end for
24: end for
```

4 Related Work

A number of performance evaluation tools based on Petri nets have been developed in the last decade, such as Möbius [6], GreatSPN [7] or TimeNET [8]. But in this work, we only revise and compare those tools that focus in the SPE field.

DSPNexpress-NG [19], proposed by Lindemann et al., constitutes a framework that can evaluate both discrete-event systems specified as Petri nets and UML system models. It uses UML statecharts which are not annotated according to the UML-SPT, and transforms them into deterministic and stochastic Petri nets (DSPNs) to obtain numerical solutions.

Distefano et al. [20] developed a performance plug-in for ArgoUML. Following the UML-SPT, they focuss on use cases, deployment and activity diagrams and introduce an intermediate model, which is used to gather performance information. This intermediate model is transformed into SPN and analyzed with a web-based non-markovian Petri net tool.

Using formalisms different from Petri nets, Petriu and Shen [21] propose an algorithm to transform activity diagrams into LQN models. They obtain the XML files from existing UML tools, and change them by hand in order to add performance annotations to the different model elements. The tool of Gilmore

and Kloul [22] uses ArgoUML to compile statecharts and collaboration diagrams through a process algebra language. D'Ambrogio [23] introduces a framework to automatically translate LQN models from annotated activity and deployment diagrams. Cortellessa et al. [24] propose a tool that in two phases gets a parameterized QN from use cases, sequence diagrams and deployment diagrams. Marzolla and Balsamo [25] transform annotated use case, deployment and activity diagrams into a discrete-event simulation model.

5 Conclusion and Further Work

Petri nets are recognized as a useful modeling paradigm for the performance evaluation of a wide range of systems. Nevertheless, most software engineers do not feel comfortable far from their pragmatic (non formal) modeling languages, such as UML. Moreover, engineers find easier and more productive to use only one modeling paradigm for all the project stages. Since ArgoSPE obtains GSPNs as a by-product of the software life-cycle, software engineers can use their UML models to assess system performance properties.

A number of new features can improve the tool: First, the more system properties assessed the more useful the tool become. New performance queries have to be implemented. Second, a standard format, PNML [26], could be the target file format, then gaining the possibility to use other Petri net analyzers.

Acknowledgments. The authors would like to thank Aitor Acedo, Borja Fernández, Luis Carlos Gallego, Álvaro Iradier, Juan Pablo López-Grao and Isaac Trigo for their work in the development of this tool. Finally, thanks to Simona Bernardi for her useful paper corrections and for her work testing the tool.

Availability. ArgoSPE is GNU soft. Download at: `http://argospe.tigris.org`.

References

1. Molloy, M.K.: Performance analysis using stochastic Petri nets. IEEE Transactions on Computers **31**(9) (1982) 913–917
2. Smith, C.U.: Perf. Engineering of Software Systems. Addison–Wesley (1990)
3. Unified Modeling Language Specification. (`http://www.uml.org`) Version 1.4.
4. UML Profile for Schedulabibity, Performance and Time Specification. (`http://www.uml.org`) Version 1.1.
5. Balsamo, S., Marco, A.D., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. IEEE Trans. Software Eng. **30**(5) (2004) 295–310
6. The Möbius tool (`http://www.mobius.uiuc.edu/`)
7. The GreatSPN tool (`http://www.di.unito.it/~greatspn`)
8. The TimeNET tool (`http://pdv.cs.tu-berlin.de/~timenet/`)
9. Merseguer, J., Bernardi, S., Campos, J., Donatelli, S.: A compositional semantics for UML state machines aimed at performance evaluation. (In: IEEE WODES'02) 295–302

10. Bernardi, S., Donatelli, S., Merseguer, J.: From UML sequence diagrams and statecharts to analysable Petri Net models. (In: ACM WOSP'02.) 35–45
11. López-Grao, J.P., Merseguer, J., Campos, J.: From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. (In: ACM WOSP'04) 25–36
12. Merseguer, J.: Software Performance Engineering based on UML and Petri nets. PhD thesis, University of Zaragoza, Spain (2003)
13. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley Series (1995)
14. The ArgoUML project (`http://argouml.tigris.org`)
15. Bernardi, S., Merseguer, J.: QoS assesment of fault tolerant applications via stochastics analysis. IEEE Internet Computing (2006) Accepted for publication.
16. Merseguer, J., Campos, J., Mena, E.: Analysing internet software retrieval systems: Modeling and performance comparison. Wireless Networks **9**(3) (2003) 223–238
17. XML Metadata Interchange (XMI) (`http://www.omg.org`)
18. Java Tecnology (`http://java.sun.com`)
19. DSPNexpressNG (`http://www.dspnexpress.de`)
20. Distefano, S., Paci, D., Puliafito, A., Scarpa, M.: UML Design and Software Performance Modeling. (In: ISCIS'04, vol. 3280 of LNCS.) 564–573
21. Petriu, D., Shen, H.: Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. (In: TOOLS'02, vol. 2324 of LNCS.) 159–177
22. Gilmore, S., Kloul, L.: A unified tool for performance modelling and predicition. (In: SAFECOMP'03, vol. 2788 of LNCS.) 179–192
23. D'Ambrogio, A.: A model transformation framework for the automated building of performance models from UML models. (In: ACM WOSP'05.) 75–86
24. Cortellessa, V., et al.: XPRIT: An XML-Based Tool to Translate UML Diagrams into Execution Graphs and Queueing Networks. (In: IEEE QEST'04.) 342–343
25. Marzolla, M., Balsamo, S.: UML-PSI: The UML Performance Simulator. (In: IEEE QEST'04.) 340–341
26. PNML (`http://www.informatik.hu-berlin.de/top/pnml/about.html`)