An XML Based Simulation Method for Extended Queuing Networks

Andreas W. Liehr, Klaus J. Buchenrieder
Institut für Technische Informatik
Universität der Bundeswehr München
D-85577 Neubiberg, Germany
Email: {andreas.liehr|klaus.buchenrieder}@unibw.de

KEYWORDS

Extended Queuing Network Models, Extensible Markup Language, Event Based Simulation Method

ABSTRACT

Extended Queuing Networks (EQN) are popular models for performance simulation of computer systems and communication networks. The EQN simulator developed in this work provides a simple and efficient simulation environment, which can be seamlessly integrated by XSLT transformations into modern system-design environments. Novel in this approach, is the consistent exertion of XML for all programmed modules of the simulator and the specification of the EQNs itself. The simulation method developed profits from XML Schema and provides a framework for constructing and solving standard Queuing Networks (QNM) as well as EQNs. The application of the XML Path Language (XPath) warrants a simple, robust and error resistant simulation, that can be interfaced with numerous programming languages with an XML interface. The paper presents the simulation method, the underlying XML model, model validation and the analysis of results. In the contribution, we also provide implementation details and experimental results for the prototype simulator.

INTRODUCTION

Since the invention and introduction of Extended Queuing Network Models by Charles Sauer (Sauer et al., 1980; Sauer and MacNair, 1983) numerous advances for specification, simulation and use of EQNs have been brought forward. Sinclair and Mandala provide a capacious list of EQN simulation tools and ascents in the field (Sinclair and Madala, 1986). A more recent approach, relies on Java for model description and adopts HLA-based distributed simulation (D'Ambrogio et al., 2006; Gianni and D'Ambrogio, 2007).

Multipresent research (Musovic et al., 2005; Gu and Petriu, 2005; Woodside et al., 2005; Balsamo et al., 2006) on performance simulation and power estimation, profits from QNMs and EQNs, even though commercial support for native EQN simulation is unavailable. The gap between proprietary representations within simulation tools and simulation models in XML can be bridged by theoretical methods developed by Xu, D'Ambrogio and Liehr (Xu and Lehmann, 2004; D'Ambrogio and Iazeolla, 2005; Liehr and Buchenrieder, 2007). These academic foundations lead us to develop an event driven EQN simulator, that works on XML representations of a simulation model. Through the design of our EQN description in XML and a dexterously designed XML Schema definition, the syntactical and semantical verification reduces to the validation of the XML representation against the XML Schema. This enables us to suspend the simulation process in every state by writing the current state to an XML file and to an overarching resume file of the simulation process. This resume constitutes the baseline for the independently conducted analysis.

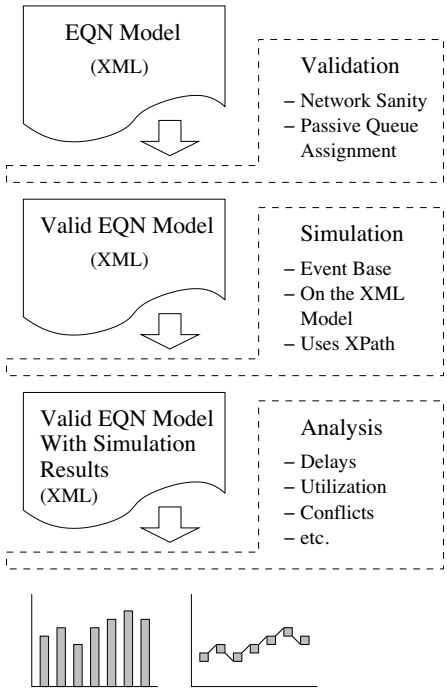

Figure 1: The XML Based EQN Simulator

The design of our event-driven simulation method splits the process into a validation phase, a simulation phase and an analysis phase as presented in Figure 1. While there is no need to define the desired simulation results prior to analysis, the structure of the gained simulation results foster a fast gathering of required informa-

Proceedings 22nd European Conference on Modelling and Simulation ©ECMS Loucas S. Louca, Yiorgos Chrysanthou, Zuzana Oplatková, Khalid Al-Begain (Editors)
ISBN: 978-0-9553018-5-8 / ISBN: 978-0-9553018-6-5 (CD)

tion, concerning arbitrary aspects (like average waiting time for jobs, overall throughput on distinct points of the EQN, etc.) of the simulation model.

The remainder of this paper is organized as follows: The following section introduces the syntax of our XML representation for EQNs and the benefits of the XML Schema. Section 3 introduces our simulation algorithm and in the following section we present details of the analysis phase. The concluding sections bring forward our prototype implementation and provide experimental results.

THE XML MODEL

The EQN structure used in our work, resembles the definition originally proposed by Sauer, McNair and Salza (Sauer et al., 1980; Sauer and MacNair, 1983). The networks are composed of the following network elements: a set of $Nodes$, token pools, variables and connections between the elements of $Nodes$.

The set $Nodes_{PQ}$ contains all elements of the network which belong to passive queues except the token pool itself:

$$Nodes_{PQ} = \{ALLOCATION\_NODES, CREATE\_NODES, DESTROY\_NODES, RELEASE\_NODES\} \quad (1)$$

The set $Nodes$ contains all elements through which jobs can be routed:

$$Nodes = \{Nodes_{PQ}, ACTIVE\_QUEUES, ROUTING\_NODES, ROUTING\_POINTS, SET\_NODES, SOURCES, SINKS\} \quad (2)$$

In contrast to the original definition, we apply minor modifications concerning ingoing and outgoing connections, to allow for a validation step. The semantics of the modified and the original version, however remain identical.

Elements in $Nodes$ are restricted to exactly one outgoing connection per node except for sinks and routing nodes. Paths can be split only at routing nodes, containing a list with an arbitrary number of outgoing routes and a rule specifying the routing of the arriving jobs. The rule decides which route each job will take.

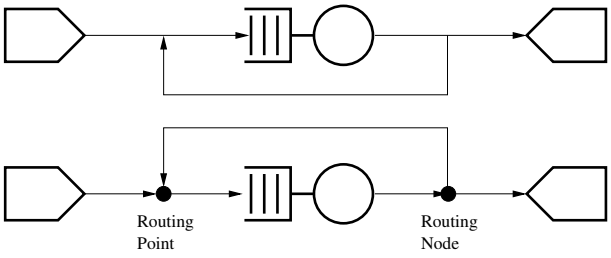

Figure 2: The Introduction of Routing Node and Routing Point

The set $Nodes_s$ denotes all elements with a successor, the jobs can be routed through. Because routing nodes have as many successors as routing decisions are possible at this point of network, the ROUTE elements of routing nodes, representing this possibilities, are included in this set:

$$Nodes_s = \{Nodes_{PQ}, ACTIVE\_QUEUES, ROUTING\_POINTS, SET\_NODES, SOURCES, ROUTING\_NODES/ROUTES\} \quad (3)$$

Furthermore, we introduce routing points to ensure that every element of $Nodes$ has exactly one incoming connection. Exceptions from this rule are made only for routing points merging a number of paths.
The set $Nodes_p$ contains all elements with a predecessor, through which jobs can be routed. Because routing points have as many predecessors as routes, the ROUTE elements of routing points are part of the set:

$$Nodes_p = \{Nodes_{PQ}, ACTIVE\_QUEUES, ROUTING\_NODES, SET\_NODES, SINKS, ROUTING\_POINTS/ROUTES\} \quad (4)$$

Figure 2 shows a traditional EQN and the semantically equivalent representation, using the modified notation, below. For simulation, it is stored in a single XML file with the top level element SIM_EQN. Listing 1 provides an example. It constitutes the initial state for the simulation of the EQN shown in Figure 3. The nodes of the EQN are stored as child elements of the element NETWORK.

Listing 1: The XML Notation for EQN Models

```
<SIM_EQN>
<NETWORK>
 <SOURCE Id='1' Rule='D:30' NId='2'>
  <JOB>
   <TOKENS/>
   <VARIABLES/>
  </JOB>
 </SOURCE>
 <ALLOCATION_NODE Id='2' NId='3' PQId='1' Rule
     ='D:1'/>
 <ACTIVE_QUEUE Id='3' NId="4' Rule='D:50'
     Scheduler='FCFS'/>
 <RELEASE_NODE Id='4' NId='5' PQId='1'  Rule='
     D:1'/>
 <SINK Id='5'/>
</NETWORK>
<PASSIVE_QUEUES>
 <PASSIVE_QUEUE Id='1' Size='1' Tokens='1'
     Scheduler='FCFS'/>
 </PASSIVE_QUEUES>
<VARIABLES/>
<GLOBAL_TIMER Time='0' Stop='10000'/>
<JOBS/>
<HISTORY/>
<NEXT_EVENTS>
  <EVENT Type="NEW_JOB" NId="1" Time="0"/>
</NEXT_EVENTS>
</SIM_EQN>
```

Each element of $Nodes_s$ has a unique identification attribute Id. A predecessor element from $Nodes_p$ points to its successors Id attribute with its attribute NId. Depending on the type, each node may contain additional

attributes or child elements such as: building rules, job templates, size information, scheduler type, etc. These attributes allow us to store the information of these nodes in our XML representation. As an example, consider the source in Listing 1: It contains a building rule and a template for jobs as a child element. Each node is assigned to a passive queue and contains the Id of the matching passive queue as attribute PQId.

A passive queue is stored as a child element named PASSIVE_QUEUE of the element PASSIVE_QUEUES. It is attributed with an Id, a maximal size, a scheduler type, the current number of tokens, and the token pool of the queue.

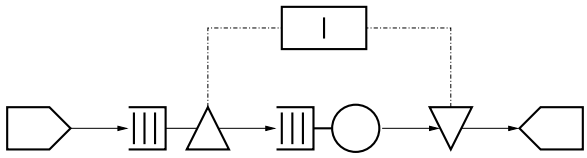

Figure 3: An EQN with one Active Queue and one Passive Queue

The element VARIABLES contains global variables as child elements. These may be required by routing nodes to perform complex routing decisions for the jobs.
The element GLOBAL_TIMER of the simulation model holds the current time stamp, followed by the time stop for the end of the simulation. Jobs generated by the source elements are stored inside the element JOBS. The HISTORY element provides a movement trace of jobs and the flow of tokens. The element NEXT_EVENTS contains information about pending creations and the finishing time for the servicing of jobs.

Properties of the Simulation Model

The model validation against the XML Schema guarantees three properties: (i) all elements of $Nodes_s$ from definition (3) have a predecessor and can be reached by jobs; (ii) every element of $Nodes_p$ from Definition (4) has exactly one successor node; (iii) every element of $Nodes_{PQ}$ is assigned to exactly one passive queue.
While (i) and (ii) are ensured by the network validation, (iii) is enforced by passive queue assignment validation.

Network Validation: The validation process ensures, that the connections between nodes in the network are well defined according to the definition of EQNs. This holds when the function, with $Nodes_p$ as source and $Nodes_s$ as target, is bijective. The mechanism involves two pairs of keys and key references in the XML Schema definition.
The key of the first pair, addresses the successor-definition within the network. It is constructed from the Id attributes of all elements in $Nodes_s$ (3) and thereby implies, that two elements of $Node_s$ can not have the same Id:

$$\forall m \in Nodes_s \nexists n \in Nodes_s : m \neq n \wedge m.Id = n.Id \quad (5)$$

The associated key reference is build from the NId attributes of all nodes having a successor, as defined in (4). This key reference ensures, that a successor for every element of $Nodes_p$ exists in $Nodes_s$:

$$\forall m \in Nodes_p \exists n \in Nodes_s : m.NId = n.Id \quad (6)$$

Listing 2 shows an XML Schema definition for the first key/keyref pair.

Listing 2: An XML Schema Key Definition for EQN Nodes

```
<xsd:key name="networkKey">
 <xsd:selector xpath="./EQN:SINK | ./
      EQN:ACTIVE_QUEUE | ./EQN:ALLOCATION_NODE
      | ./EQN:RELEASE_NODE | ./EQN:DESTROY_NODE
       | ./EQN:CREATE_NODE | ./EQN:SET_NODE |
      ./EQN:ROUTING_NODE | ./EQN:ROUTING_POINT/
      EQN:ROUTES/EQN:ROUTE" />
 <xsd:field xpath="@Id" />
</xsd:key>
<xsd:keyref name="networkKeyRef" refer="
     EQN:networkKey">
 <xsd:selector xpath="./EQN:SOURCE | ./
      EQN:ACTIVE_QUEUE | ./EQN:ALLOCATION_NODE
      | ./EQN:RELEASE_NODE | ./EQN:DESTROY_NODE
       | ./EQN:CREATE_NODE | ./EQN:SET_NODE |
      ./EQN:ROUTING_NODE/EQN:ROUTES/EQN:ROUTE |
       ./EQN:ROUTING_POINT" />
 <xsd:field xpath="@NId" />
</xsd:keyref>
```

The second key is defined by the NId attributes of $Nodes_p$. It implies that two elements of $Nodes_p$ can not have the same successor attribute NId:

$$\forall m \in Nodes_p \nexists n \in Nodes_p : m \neq n \wedge m.NId = n.NId \quad (7)$$

The second reference key is defined by the same set of Ids as the first key. This warrants a predecessor in $Nodes_p$ for every element of $Nodes_s$:

$$\forall m \in Nodes_s \exists n \in Nodes_p : m.Id = n.NId \quad (8)$$

From (6) and (7) follows, that the connections between nodes are a function f as defined in (9). Thereby, NId values of elements in $Nodes_p$ are the source X in function f. The Id values of elements in $Nodes_s$ are the target Y:

$$\begin{aligned} X &= \{x | \exists m \in Nodes_p : x = m.NId\} \\ Y &= \{y | \exists n \in Nodes_s : y = n.Id\} \\ f &: X \to Y, x \mapsto x := y \end{aligned} \quad (9)$$

Definition (9), together with (5) imply:

$$\forall m \in Nodes_p \exists! n \in Nodes_s : m.NId = n.Id \quad (10)$$

Consequently, f is surjective and the function f is injective as (8) and (9) imply. Hence, the combination of (8), (9) and (10) suggests f bijective.
The XML Schema allows only well formed EQNs according to the basic definition of EQNs. Nodes with an illegal number of predecessors, as shown in the left upper

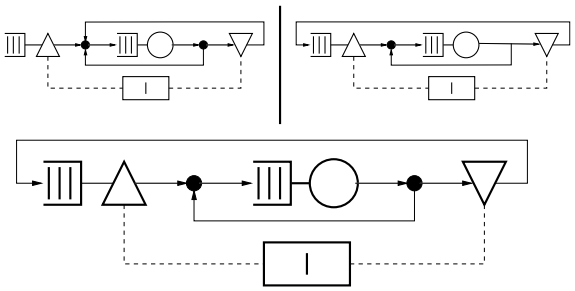

Figure 4: A Valid EQN and Two Invalid Modifications

part of Figure 4, will not be validated. The same holds for nodes with an illegal number of successors, as depicted in the upper right part of Figure 4.

Passive Queue Assignment Validation: A third key/keyref pair ensures, that every element of $Nodes_{PQ}$ is assigned to exactly one passive queue. The key is defined with the Id attributes of all passive queues.:

$$IDs = \{x | \exists\, m \in PASSIVE\_QUEUES : m.Id = x\} \quad (11)$$

Every passive queue has a unique Id:

$$\forall m \in PASSIVE\_QUEUES \nexists\, n \in PASSIVE\_QUEUES : m \neq n \wedge m.Id = n.Id \quad (12)$$

The key reference is defined by the PQId attribute of every element from $Nodes_{PQ}$.:

$$PQIDs = \{x | \exists\, m \in Nodes_{PQ} : m.PQId = x\} \quad (13)$$

From (12) and the definition of the key reference follows:

$$\forall m \in PQIDs\, \exists!\, n \in IDs : m.PQId = n.Id \quad (14)$$

This proves that every element of $Nodes_{PQ}$ is assigned to exactly one passive queue. Hence, the EQN constructions shown in Figure 5 are impossible.

While we only needed to write an appropriate XML Schema, the whole validation process is done by off the shelf XML validation tools. Because every part of the simulation result will be created by the XML manipulation methods, the result of the simulation is valid by construction, as long as the XML Schema is correct.

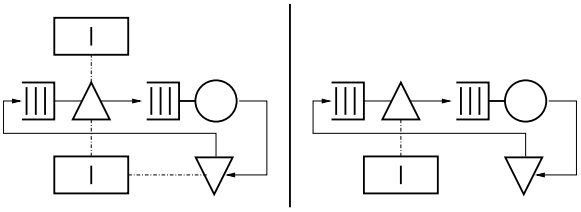

Figure 5: Two EQN with illegal assigned Passive Queues

THE SIMULATION PHASE

In the simulation phase, validated EQN models are executed on an event base. The simulation is synchronized with a global timer and extends from a current-time to the end-time prespecified in the XML representation. Since the current-time variable is advanced whenever a simulation event completes, the simulation process can be stopped and relaunched at any time. Hence, one can analyze partial simulations and continue thereafter.

The main_algorithm: For the explanation of the algorithm and improved readability we use Nassi-Schneidermann diagrams and XPath (Clark and DeRose, 1999) expressions, as illustrated in Figure 6.

First, the current time at the start of the simulation and the finishing time for the simulation are extracted from the XML representation. Afterwards a loop is initiated, that repeats until the time inside the simulation exceeds the predefined finishing time.

Inside this loop, the lists for triggered servers and passive queues are initialized. Then all events from NEW_EVENTS, that have reached their end-time, are extracted. Those events can be either from the type NEW_JOB or SERVED. When the type is NEW_JOB, a source releases the next job to the network. The method create_job (), explained below, is executed for this source as shown in the diagram.

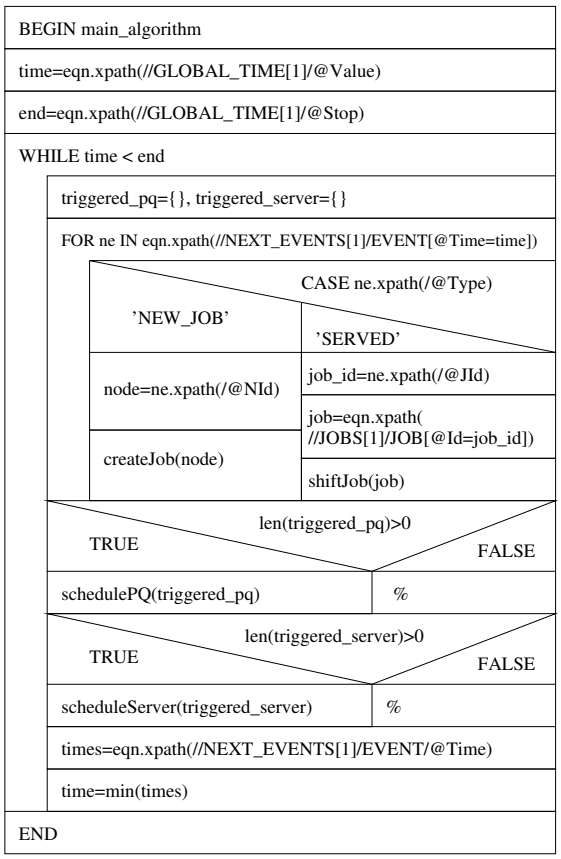

Figure 6: The Method main_algorithm

If an event of the type SERVED occurs, a server has finished the processing of the current job and shift_job () executes.

During the processing of jobs in the network, tokens are created or released into a token pool. In this case, the Id of the passive queue is appended to the variable schedule_pq and causes the activation of the appropriate scheduler. A scheduler of a queue, belonging to a server, is activated if the server finishes a job or jobs queue in front of an idle server.

The last XPath expression gathers the timestamps for all events that occur at a defined time in the network. Such events can be the generation of new jobs by sources and the end of the service time of a job at a server. The last line determines the minimum of these timestamps and sets the global time to this value before the loop is started again.

The method create_job: This method injects new jobs into the network when called by the main algorithm. It builds a copy of the job template provided by the source the job is created from. The copy is assigned to an XML element named PRESENT that represents the current state and position of this job. The PRESENT, representing the current time as start and stop time and the Id of the source as current position.

The time for job creation of this source is determined and an EVENT-element, holding this information, added to the list of next events. Last, the job is handed over to the shift_job () method.

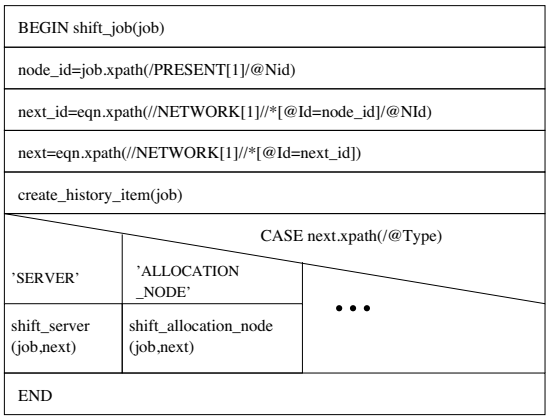

Figure 7: The Method shift_job

The method shift_job: The method shift_job () implements the movement of jobs through the network, Figure 7. First, the successive node for the job and the type of this node is extracted by XPath. Second, a history item is created and appended to the simulation HISTORY. It contains the job Id, the node Id, the arrival-time and the leaving-time of the job on the current node.

Depending on the type of the successor node, a method containing a node specific algorithm is called. This modular structure enables the extension of the algorithm for new types of nodes and the selective modification of the behavior of single node types without influencing the remainder of the simulation system.

THE ANALYSIS PHASE

During the simulation phase, the XML based EQN is autonomously simulated. In the analysis phase, the desired simulation results, such as waiting times, service times, resource utilization, throughput, etc. are specified by the user. Then, average, worst case, best case or complex interpretations of the demanded parameters are calculated.

In the following example, we derive the average waiting time for the jobs of the network shown in Figure 3.

Listing 3: Average Waiting Time for Jobs to be Served

```
items=eqn.xpath('/SIM_EQN/HISTORY/HI[@NId=3]')
sum=0
for item in items:
 sum+=item.xpath('@Stop')-item.xpath('@Start')
result=sum/len(items)
```

While the first XPath expression collects all entries in the history related to the allocation node, the for-loop sums up the total waiting time of its queue. The sum is then divided by the total number of jobs that were served by this server:

Listing 4: Number of Token Allocations from Passive Queue 1

```
items=eqn.xpath('/SIM_EQN/HISTORY/HI[@NId=2
      and @Tokens>0]')
result=len(items)
```

The code in Listing 4 produces the total number of allocation processes, connected to the passive queue with Id 1.

It is also possible to return to the simulation phase from the analysis phase. In this case only the attribute Stop of the element GLOBAL_TIMER must be altered. Partial simulations are carried out by specific time-intervals.

IMPLEMENTATION

The current version of the simulator is implemented in Python on a standard X86 architecture under Mac OSX. All XML manipulations are performed with the Python LXML library. For performance reasons, we split the XML model into several parts after import completes. The first part contains the network comprised of the static elements NETWORK and PASSIVE_QUEUES. The second part contains all active Jobs. The third part consists of the elements GLOBAL_TIMER, VARIABLES and NEXT_EVENTS. In each simulation step all elements belonging to HISTORY and jobs arriving at a sink are output. For analysis, module outputs are merged as soon as the simulation terminates or when interrupted by user interaction. This allows us to separate EQN creation, validation, simulation and analysis.

The current implementation enables the simulation of complex EQNs, as shown in Figure 8.

EXPERIMENTS

The EQN-based specification and simulation method was successfully tested with representative examples, such as the multiqueue network shown in Figure 8. It is composed of a CPU with a single-level cache, memory and a bus.

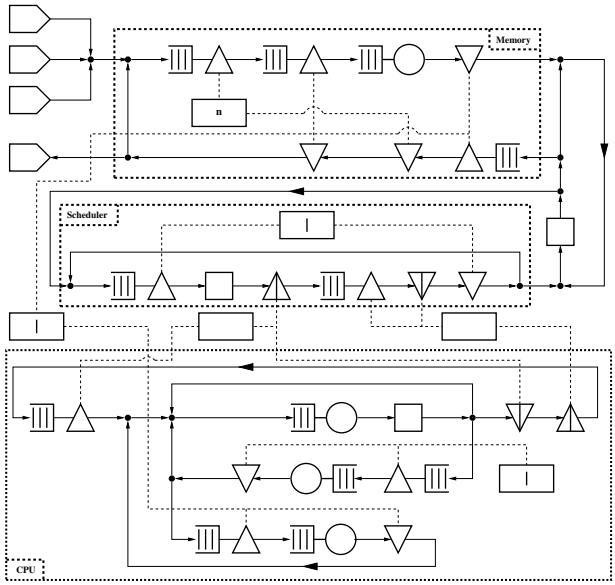

Figure 8: EQN Model for Experiments

When execution starts, three sources release jobs of different types into the system. Jobs from type one are compute intensive with little IO. Source one releases a type one job every 50 seconds into the system. IO intensive jobs of type two are generated and released at random time-intervals reaching from 1 to 10 seconds. Jobs from type three are very short but time-critical. Their generation interval spans from 5 to 50 seconds.

Numerous simulations for all active source combinations were carried out. In addition, different scheduling strategies for the CPU have been computed to demonstrate the simulator. In the analysis phase, we determined the average and the maximal time a job remained in the network. We also measured the average utilization of the system components during the simulated period.

Table 1: Execution Times and System Utilization for FCFS Scheduler

Sources	⌀ **1** (s)	⊤ **1** (s)	⌀ **2** (s)	⊤ **2** (s)	⌀ **3** (s)	⊤ **3** (s)	**Util.** (%)
1	5.76	5.81	-	-	-	-	11.49
2	-	-	4.06	6.94	-	-	65.74
3	-	-	-	-	0.04	0.05	0.07
1;2	6.84	9.00	4.80	11.53	-	-	74.24
1;3	5.80	5.99	-	-	0.37	5.79	11.64
2;3	-	-	4.12	10.03	1.94	7.00	63.81
1;2;3	7.38	10.77	5.41	13.81	3.14	12.75	76.17

FCFS Scheduling: With FCFS as scheduling algorithm, jobs are served continuously in the order of arrival. As expected, the service time increases when multiple jobs await processing. Table 1 depicts the simulation results for the system with FCFS. The left column indicates the active sources. Column two contains the average execution time for type 1 jobs, column three holds the maximum execution time figured. The last column gives the average utilization of the system during the simulation interval. Clearly, the execution time for small jobs of type three increases as multiple sources are activated.

Round Robin Scheduling: As shown in Table 2, the average and maximal execution time for jobs from type three decreases noticeable for multiple active sources and Round Robin scheduling. As a result, a shorter time slot lowers the worst case execution time for jobs of this type even more.

Table 2: Execution Times and System Utilization for Round Robin Scheduler

Sources	⌀ **1** (s)	⊤ **1** (s)	⌀ **2** (s)	⊤ **2** (s)	⌀ **3** (s)	⊤ **3** (s)	**Util.** (%)
Time Slot 20 ms							
1;2	14.89	31.80	4.62	9.03	-	-	74.83
1;3	5.73	5.85	-	-	0.04	0.67	11.49
2;3	-	-	4.23	11.51	0.13	0.49	61.42
1;2;3	13.19	26.24	4.38	10.33	0.12	0.53	68.0
Time Slot 100 ms							
1;2	15.20	29.06	4.39	10.34	-	-	72.08
1;3	5.77	5.93	-	-	0.05	0.16	11.57
2;3	-	-	4.63	10.16	0.51	1.82	65.34
1;2;3	18.30	50.01	5.56	13.74	0.57	2.37	77.76

Implementation Details: The EQN-based specification method and the simulator are implemented and executed with Python, version 2.5.2, on an Intel 2.2GHz Core 2 Duo CPU using a Mac OSX 10.5.2 operating system. While the first implementation with the PyXML package proved memory intensive and slow, we achieved a 13x speedup with an implementation based on the LXML Pythonic XML processing library in version 2.0apha6 and libxml2, version 2.6.21.

The maximal memory allocation during the simulation of the EQN of Figure 8 for $1.3x10^6$ job movements amounts less than 5 megabyte. The XML file, holding the simulation results, had a size of 56 megabytes. The simulation for this movement count required little over 27 minutes.

CONCLUSION AND OUTLOOK

In this paper, we propose a novel method to specify and simulate Extended Queuing Network Models entirely provided in XML. The method consists of a model validation, a simulation and an analysis phase. For all steps we use the power of XPath expressions to provide a simple and fault robust simulation. Model validation warrants that the presented EQN description fulfills the

properties, network sanity and singular passive queue assignment. In the analysis phase, we obtain system information from the simulation data. For excessive amounts of simulation data, due to model complexity , our method fosters distributed computation.

The methods and prototypes presented in this paper are highly appreciated by our industrial and scientific partners. The long-term goal of this research is to develop a system-specification environment in which the software, hardware and the system-architecture is captured in UML as presented in Liehr and Buchenrieder (2008). Based on this specification, a set of XML-described queuing networks is generated to allow for an efficient architectural exploration of the behavioral model. In this context, the EQN-simulator is an integral component for system-evaluation and performance-estimation.

Future work will include the enhancement of our model with the node types SPLIT_NODE, FISSION_NODE and FUSION_NODE to meet the full specification of EQNs. Furthermore, we intend to embed the simulator component into a web service that accepts XML based models and returns the simulation results. This will increase the usability of the approach in frameworks outside of the focus-domain of this research.
Another advancement persecuted is the implementation of a distributed multi-CPU method for the analysis of performance results.

REFERENCES

Balsamo, S., Marzolla, M., and Mirandola, R. (2006). Efficient performance models in component-based software engineering. In *SEAA '06 Proceedings of the 32nd Euromicro Conference on Software Engineering and Advanced Applications*, Cavtat/Dubrovnik, Croatia.

Clark, J. and DeRose, S. (1999). Xml path language (xpath) version 1.0. Technical report, World Wide Web Consortium.

D'Ambrogio, A., Gianni, D., and Iazeolla, G. (2006). jeqn a java-based language for the distributed simulation of queueing networks. In *Computer and Information Sciences - ISCIS 2006*, volume 4263/2006, pages 854–865, Istanbul, Turkey. Springer Berlin / Heidelberg.

D'Ambrogio, A. and Iazeolla, G. (2005). Design of xmi-based tools for building eqn models of software systems. In *23rd IASTED International Multiconference, Software Engeneering*, Innsbruck. ACTA Press.

Gianni, D. and D'Ambrogio, A. (2007). A language to enable distributed simulation of extended queueing networks. In *Journal of Computers*, volume 2.

Gu, G. P. and Petriu, D. C. (2005). From uml to lqn by xml algebra-based model transformations. In *WOSP '05: Proceedings of the 5th international workshop on software and performance*, pages 99–110, New York, NY, USA. ACM Press.

Liehr, A. W. and Buchenrieder, K. J. (2007). Generation of related performance simulation models at an early stage in the design cycle. In *14th IEEE International Conference on the Engeneering of Computer-Based Systems (ECBS)*, pages 7–14, Tucson, AZ, USA. IEEE Computer Society.

Liehr, A. W. and Buchenrieder, K. J. (2008). Performance evaluation of hw/sw-system alternatives. In *Design, Automation and Test in Europe (DATE) University Booth - Demonstration and Poster Exhibition*, Munich, Germany.

Musovic, A., Fischer, R., Nageldinger, U., Buchenrieder, K., and Lehmann, A. (2005). An eqn* based approach for high-level performance and power estimation. In *Conceptual Modeling and Simulation Conference, 2005. CMS 2005. 2nd 13M International Mediterranean Modeling Multiconference*, pages 193–198.

Sauer, C. H. and MacNair, E. A. (1983). *Simulation of Computer Communication Systems*. Prentice Hall.

Sauer, C. H., MacNair, E. A., and Salza, S. (1980). A language for extended queuing network models. *IBM Journal of Research and Development*, 24(6):747–755.

Sinclair, J. B. and Madala, S. (1986). A graphical interface for specification of extended queueing network models. In *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, pages 709–718, Los Alamitos, CA, USA. IEEE Computer Society Press.

Woodside, M., Petriu, D. C., Petriu, D. B., Shen, H., Israr, T., and Merseguer, J. (2005). Performance by unified model analysis (puma). In *WOSP '05: Proceedings of the 5th International Workshop on Software and Performance*, pages 1–12, New York, NY, USA. ACM Press.

Xu, Z. and Lehmann, A. (2004). Generating queuing network models from uml-based models for software performance prediction. In *SPECTS' 04: Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, San Jose, California, USA.

AUTHOR BIOGRAPHIES

ANDREAS W. LIEHR is a research assistant at the Universität der Bundeswehr München. He is currently working towards his doctoral degree in the field of performance prediction and power estimation in the development cycle of embedded systems.

PROF. KLAUS J. BUCHENRIEDER, PH. D. is professor at the Universität der Bundeswehr München, where he is head of the "Embedded Systems / Computers in Technical Systems" group.