

# EQNM<sup>2</sup>L: Towards a Metamodel for Extended Queuing Networks

Abdelhabib Bourouis<sup>1</sup> and Brahim Belattar<sup>2</sup>

<sup>1</sup> Department of Mathematics and Computer Science, University of Oum El Bouaghi,  
Oum El Bouaghi, 4000, Algeria

<sup>2</sup> Department of Computer Science, University of Batna  
Batna, 5000, Algeria

## Abstract

Queuing theory is the mathematical study of waiting phenomena that allows several performance measures derivation and calculation. Furthermore, queueing systems could be simulated, especially in case of complex models including non-classical concepts. The theory has been applied successfully in diverse fields and the development of a Domain Specific Modeling Language (DSML) for extended queueing systems seems to be a necessity and of great interest. This paper focuses on the cornerstone of this process by providing a minimal but clear and extensible metamodel called EQNM<sup>2</sup>L. Our aim is to induce discussion on and contributions for elaborating a whole mature DSML. The proposed metamodel and the XML-based exchange format are the first step in enhancing interoperability between analytical solvers and simulation tools. Based on the Model Driven Engineering (MDE) approach principles and tools, modeling environments and simulation/analytical code are generated automatically and could be maintained easily.

**Keywords:** *Extended Queueing Systems, Metamodeling, Domain Specific Modeling Language, Discrete Event Simulation, Interoperability.*

## 1. Introduction

Several Domain Specific Modeling Languages (DSMLs) have been developed in the last decade. Their use gained a lot of success and a growing popularity. However, developing a DSML is still a challenging and time-consuming task. Generally, Domain specific modeling (DSM) includes also automating the code generation directly from models. Automatic construction and maintenance of source code increases significantly developer's productivity. The reliability of automatic generation compared to manual coding reduces notably the number of defects in the resulting code thus improving quality.

Queueing network models have been used extensively as a modeling paradigm for deriving analytical as well as simulation based performance measures. They are commonly used to model a wide range of discrete-event systems. The Kendall notation is a mean for describing queueing networks especially in case of simple systems. For complex ones, a graphical notation with textual annotations is used instead. To analyze a model either by simulation or by mathematical analytic tools, the model is commonly coded and saved directly in a proprietary tool file format. Although it is the same formalism, there are always some ambiguities and disagreements on a number of concepts as well as on the exchange format. Therefore, tools are not interoperable and models can't be reused nor interchanged easily. In addition to rebuilding models from scratch every time the tool is changed, the development of modeling environments and model transformations, including code generation, are time-consuming, very expensive and hard to validate and maintain. The best solution to these problems seems to be based on MDE concepts including metamodeling and models manipulation in a generative manner.

The next section addresses key motivations and objectives of our work. A clear development methodology is presented in section 3. Section 4 discusses the basic domain concepts used for building the metamodel presented itself in section 5. The concrete syntax and the exchange format are discussed in section 6 while implementation is addressed in section 7. Finally, conclusions and improvements are given in section 8.

## 2. Motivations and objectives

The development and use of a common metamodel and a file exchange format is motivated by:

1. Lack of interoperability: performance evaluation tools of discrete event systems based on queueing

networks use different file formats for describing their models. There is no standard to ensure interoperability and reuse of conceptual models. Importing and exporting models is still hard to achieve due to the multitude of file formats. The problem is beyond simple technical solutions. XML-based standard exchange format seems to be the best solution.

2. Lack of expressiveness: unfortunately the expressive power of modeling languages depends on the tool purposes and capabilities. Tools don't model what they can't handle. In our opinion, a modeling language must be tool independent and focuses particularly on offering all necessary constructs for describing the largest possible range of systems.

3. Difficulty of the design and development of modeling environments: building modeling environments from scratch is hard, complex and time consuming. It is also difficult to maintain as the modeling language evolves. MDE seems offering the solution by automating the development based on metamodels.

4. Difficulty of code generation: the translation of the conceptual model into a simulation model is complicated and needs careful verification in addition to programming skills. Besides the manual translation, the classical automatic translation alternative is barely coded in simulation environments using programming languages. So translation rules used in this case are not clear and not easy to modify as the used formalism evolve.

The first initiative for developing such a metamodel and an XML-based exchange format named QNML was undertaken by the Center for Computer integrated Manufacturing enterprises [1]. It focuses only on analytical resolution of queueing networks and simulation was not considered. In addition, the expressiveness power of the formalism is limited and notably identical to that of the analysis tool RAQS developed at the same laboratory [2], so only concepts supported by this tool are considered.

Our aims behind this work are to explore basic concepts of Queueing theory, extract common terminology and clear semantics for designing a metamodel and an XML-based exchange format. An XML-schema will serve as a mean to define the markup language for describing queueing networks. Designing of an UML metamodel based on considered concepts that will serve as a basis for applying MDE techniques, especially, creating an integrated modeling environment and generating low level code from conceptual models. Taking into account the language extensibility, since this work constitutes only the starting version, other new concepts may be included. Modeling environment and automatic simulation code generation would evolve easily also. The A large acceptance of this formalism facilitates greatly the task of experts and researchers in the domain of systems performance evaluation.

### 3. Development methodology

Several methodologies for developing Domain Specific Languages (DSLs) have been proposed in [3], [4], [5] and [6]. A study conducted in [7], identified clearly and in detail the development process of DSLs independently of the used tools and languages. We can summarize the different stages of development:

1. Decision: the adoption of an existing DSL is the best solution and the decision to create a new one must be justified. The developing of a new DSL will have an economic impact to be considered (tools used, time spent, effort to develop, learning ... etc.).
2. Analysis: the domain is clearly identified and basic concepts are collected. This requires the cooperation of domain experts and computer scientists. The domain analysis is often done informally, but clear methodologies have been proposed and can effectively serve as DARE [8] DSSA [9] and FODA [10].
3. Design: two important elements in this phase are considered, the relationship of the new DSL with an existing language and the degree of formality of the design description. The second element is the design specification. Once the image of the DSL is clear, it can be done informally using natural language for example, easy to produce but more difficult to handle, or formally, using known techniques such as regular expressions and formal grammars to specify syntax and rewrite systems, finite state machines and attribute grammars to specify the semantics.
4. Implementation: the implementation of the design is subject to various choices made about the nature of execution of the DSL. A non-exhaustive list of alternatives includes interpretation, compilation, pre-processor, embedding and extension (as a library of modifications made to the compiler/interpreter for the host language, etc...).
5. Deployment: the effort devoted to the deployment of the DSL and its acceptance is reduced by the success of the previous phases. In addition, the ease of use, adaptation to the domain, efficiency, expressiveness, access to the DSL, involvement of a large community for testing and the impact on productivity are key points to consider.

The first step towards the development of the Extended Queueing Networks Modeling and Markup Language (EQNM<sup>2</sup>L) has been presented in the previous section where motivations and objectives are discussed. The analysis of the domain knowledge is the critical phase that guides the design and implementation which are discussed in the next sections. For the deployment stage, an online and open source project has been created to

involve a large community and easing access to this DSML.

#### 4. Domain knowledge

Most of basic concepts discussed in this paper are based on [11], [12] and [13] where a queueing network is considered as a set of service stations that are visited by jobs. Terms as transaction, customer, client and job refer to the same concept which characterizes any entity that moves through the network, from one station to another to acquire services.

##### 4.1 Job classes and priorities

In queueing networks, jobs could belong to different classes. Job classes may differ in their service and arrival distributions, the number of resource units needed to accomplish services and in their routing schemes leading to different life cycles.

A class independent priority mechanism may be considered to organize the whole population of jobs. Hence, jobs may be scheduled according to their priorities. Within the same priority category, another scheduling discipline could be indicated to distinguish between multiple jobs with equal priorities. It is possible also to consider High Value First (HVF) or Low Value First (LVF). A priority may also be static or dynamic. While a dynamic priority changes its value over time, a static one assigned to a job remains unchanged. A dynamic priority may be time-dependent in the form of  $Pr(t)$  or be a function of some system parameters. It is also possible that a job changes its class or priority when it moves inside the system.

##### 4.2 Service stations

Service centers, nodes, stations refer to the same concept. Each station is distinguished by a unique identifier and characterized by a number of parallel servers, usually identical, a waiting queue with limited or unlimited capacity, class independent scheduling disciplines, and a service probability distribution for each job class. For a particular class of jobs the service is identified by a probability distribution. Rates are generally static, but it is also possible to apply dynamic rates varying over time, especially in terms of some system indicators.

Several service disciplines are used to serve jobs including FCFS (First Come First Served), LCFS (Last Come First Served), Service in Random Order (SIRO), Processor Sharing (PS), Shortest Job First (SJF), Round Robin, and Shortest Remaining Time First (SRTF). If Round Robin is used, a time slice is indicated. A variant of this discipline called Weighted Round Robin; consider job

classes with weighted slices. Other specific disciplines may be considered.

The study of a system is influenced by its initial state which is described by the number of jobs in each station. In contrast to the case where the system is initially empty, realistic cases take into account the exact number of jobs in the system. This is important especially for closed systems where jobs are permanent components.

##### 4.3 Asymmetric stations

If servers are identical, the station is qualified to be simple, ordinary or symmetric. Otherwise, it is said to be asymmetric where parallel servers present different characteristics. We consider here those having different service distributions or various vacation/failure schemes. In real world, the heterogeneity models, for example, machines with different ages or manufacturers. This category of stations requires more information about job affectation in case of multiple idle servers. For example:

1. Random: the task is assigned to a free server at random.
2. Fastest Service: the task is assigned to the fastest free server (the greatest service rate or the smallest average time of service calculated from the log).
3. Longest idle time: the task is assigned to the server which remained idle for the longest period of time from the free servers set.

It is possible to define other allocation strategies. In addition, sometimes the allocation process can't distinguish a single server and the use of another criterion is required.

##### 4.4 Passive stations

If we consider a computer system, then a job (a process or a program) may use two or more resources at the same time, memory space and CPU for example. We consider here passive resources which are simple stations with a number of tokens (identical units) and allocate queues. A passive station has no service to offer except allocating its tokens, so the service duration is generally assumed to be nil. These passive nodes are of two types. The first is devoted to allocation of tokens and the other for release.

When arriving at an allocate station, a job requests a number of tokens. If it gets them, it can continue visiting other nodes of the network; otherwise it must wait at this node. When arriving at the corresponding release node, it releases all its tokens. These tokens are then available for other jobs [11].

##### 4.5 Decision stations

A task, during its life cycle, may have to make decisions based on the state of the system. Since a discrete event

system is described by a set of variables known as state variables, a choice structure may be assimilated to a service station in which the task takes a decision. The decision is based on a Boolean expression containing one or more state variables of the system among those predefined or user-defined.

The evaluation of the Boolean expression can have only two possible values. Hence, a decision structure has only two directions or issues. The task is then directed upon the result of the evaluation. This process is not time consuming.

#### 4.6 Service types

Service could be performed in several ways including:

1. Batch processing: the simplest way is to serve only one job at time, but it is possible to do so with a batch. Class dependent batch processing is possible by indicating the minimum and the maximum batch sizes. In this case, the whole batch is treated as one job. It is clear that jobs of a batch are homogeneous and belong to the same class.
2. Pre-emption: Some scheduling disciplines may cause preemption of the processed job when a new job arrives. A preempted job returns to the queue and may resume later starting at the point where it was stopped (preemptive resume) or restarting its service from the beginning (preemption without resume or preemptive repeat). For example, with Priority or LIFO scheduling, preemption could be enabled or disabled. It is also possible to use preemption in case of server failure. Generally, preempted job returns to the front of the queue, but other strategies may be considered.
3. Blocking: it may arise in a station when its queue has a limited capacity. Several blocking models could be distinguished such as:
  - a) Rejection: the blocked job is forced to leave the system.
  - b) Blocking after service (BAS): the blocked job will be forced to wait in its origin station until the next station is able to accept it. Hence, the origin station will be blocked (still busy) and stops servicing other jobs.
  - c) Repetitive service (RS): the blocked job is forced to repeat service by joining an Orbit until the next station is able to accept it. The blocked job may also choose a different destination according to the routing probability.
  - d) Waiting queue (WQ): The job that has just been served from a station is routed to a temporary waiting queue until next station can accept it. This method doesn't block the origin station.
4. Load dependent service: In some situations, service duration may depend on the station load, expressed by the number of jobs inside the station. In this case, for

each range of the station load, a service distribution is indicated.

5. Server failure and vacation: In practical queueing systems, service stations are not always reliable. They may become unavailable for a period of time for a variety of reasons (vacation, maintenance, failure ...). Servers failures are assumed to be independent identically distributed random variables specified by a distribution function, but some models requires different types of failure and follow different distributions. Failure may be synchronous (all servers fail at the same time) or asynchronous (servers fail independently). Two probability distributions are useful for modeling the mean time between failures (MTBF) and mean time to repair or recover (MTTR). Vacation is slightly different from failure. While the latter is accidental, the former is scheduled and conscientiously undertaken [14]. It is a process governed by a policy that explicitly specifies:
  - a) Staring rules: exhaustive when beginning vacation only if queue (system) is empty, otherwise it is non-exhaustive and can begin at any time.
  - b) Termination rule: determines when resuming. Multiple vacation policy if keeping vacation until having a job to serve. Single vacation policy if resuming immediately and begin servicing if customers are waiting or stay idle otherwise.
  - c) Vacation duration: symmetric (independent and identically distributed) or asymmetric (different distributions).For multi-server systems, starting and termination policies are more complicated (synchronous or asynchronous). Other policies are also possible like guarantee of minimum service availability.

#### 4.7 Sources

A service station may or not receive jobs from outside of the system (exogenous arrivals), but must have at least an output job flow towards another station or outside the system.

An input from outside denotes a source of jobs and has a unique identifier. It is associated to an exogenous job flow characterized by a probability distribution, a class and a priority to describe the corresponding arrival process. It is also possible to specify batch arrivals by indicating the distribution of the batch size. Arrival rate for a specific job class is generally static, but we may consider dynamic rates. Some sources have a limited population which is important to consider.

#### 4.8 Job behavior

In a queuing model, the behavior of jobs is an important feature and three major situations may arise:

1. **Balking:** Some customers decide not to join the queue because of its length or insufficient waiting space. This behavior results in the discouragement of customers for not joining an improper or inconvenient queue. It could be modeled using decision stations.
2. **Reneging:** Once in a queue, a job may choose to leave it if it has waited too long (“timeout reneging”). It requests a resource and wants to leave before acquiring it, for example because it could get quicker service by another resource with a shorter queue (“conditional reneging”). Reneging pertains to impatient customers. After being in queue for some time, few customers become impatient and may leave the queue.
3. **Jockeying:** Jockeying is a phenomenon, in which customers move from one queue to another queue with a hope that they will receive quicker service in the new position.

Job’s behavior is station dependent and is more suited to be indicated for each station.

#### 4.9 Orbits

In many situations, jobs leaving a service station join a region called “Orbit”. After a delay they retry their queries. So, an orbit is a source of jobs fed by the system itself and allows its jobs to retry queries after a random amount of time. In queuing theory, this is a key feature for retrial queues [15].

#### 4.10 Outputs

Each station may have one or more outputs. A routing strategy must be defined for each job class. An output is identified by a reference to the destination station and routing information according to the routing strategy.

It is possible that a job changes its class after a service. This situation is common in manufacturing systems where a given machine transforms received products to new ones. An output normally needs to indicate the associated resulting job class for every received one.

#### 4.11 Routing strategy

After service completion, a job is routed according to a routing strategy which may be for example:

1. **Probabilistic:** job moves to the next station with a given probability.
2. **Round robin:** all possible destinations for a job class are chosen in a cyclic manner.

3. **Shortest queue length:** job moves to the next station having the shortest queue.
4. **Smallest response time:** job moves to the next station having the smallest response time calculated from the log.
5. **Smallest utilization:** job moves to the next station having the smallest average utilization calculated from the log.
6. **Fastest service:** job moves to the next station having the smallest average service time calculated from the log.

#### 4.12 Regions

The concept of regions is very powerful and denotes a set of stations in the system with controlled capacity. It allows expressing global constraints on a group of stations as finite capacity for specific job classes. It is also useful in global statistics computation. Hence, a *Finite Capacity Region* (FCR) is a region of the model or a set of stations, where the number of jobs is under control [16]. It is possible to define an FCR capacity by setting upper bounds for the number of customers in the region for specific classes or globally regardless of the job classes.

### 5. Metamodel

Models are created using a modeling language called a metamodel. The modeling language is itself is described in another language called meta-metamodel. The philosophy of DSM favors the creation of a new language for a specific task, and hence there are naturally new languages designed as meta-metamodels. DSM environments can considerably reduce the cost of obtaining tool support for a DSML, since a well-designed DSM environment will automate the creation of application modules costly to build from scratch, such as domain-specific editors, compilers and models transformers. The domain experts only need to specify the domain specific constructs and rules, and the DSM environment provides a modeling tool tailored for the target domain.

It has been established that a language is characterized by its syntax and semantics. The syntax describes the different language constructs and their arrangement rules while semantics refers to the relationship between a signifier (program or model) and a signified (mathematical object) to give meaning to each of the constructs of the language. So the relationship between semantics and syntax is the same as between the content and the form.

The metamodel represents the abstract syntax and is the heart of the language, since it captures the whole domain concepts and their relationships. It acts as a pivot between the concrete syntax description and the semantics

description. Concrete syntax contains information on how the concepts in the metamodel are to be represented to the language user. This is sometimes amended by a mapping in which each abstract concept is related to a textual notation or a graphical symbol.

The EQNM<sup>2</sup>L metamodel illustrated in Figure 1 is expressed in UML class diagram. It represents the abstract syntax and a part of the static semantics. The rest of the static semantics could be expressed using OCL.

An extended queueing network (EQNET) is an aggregate of nodes, decision stations, sources, orbits, outputs, outsides, sink and finite capacity regions. It may

contain a description which consists of a set of meta-information about the model as name, used tool, version and author. A node is an abstract concept that models a variety of specific stations as simple, passive and asymmetric stations. In contrast of an open network, a closed network does not need a sink which models the system's environment.

It is important to note that some domain concepts discussed in previous sections are not considered in this minimal metamodel. The metamodel in its actual state lacks of many concepts but is extensible to include new concepts in order to gain more maturity.

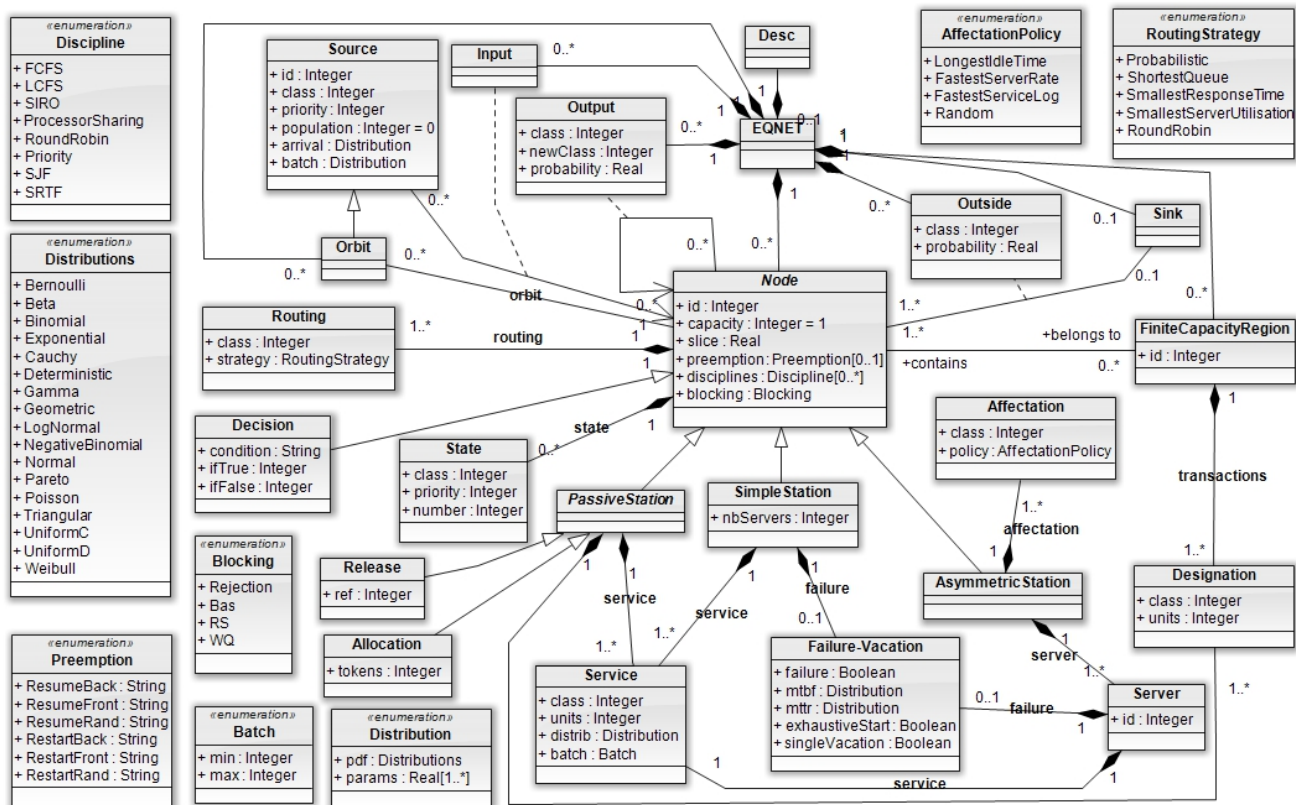


Fig.1 EQNM<sup>2</sup>L Metamodel.

The static semantics is part of the metamodel. It consists of a set of rules to ensure that it is well formed (well-formedness rules). Rules allow expressing constraints and thus reducing the overall valid models. We present these constraints in natural language while trying to be concise and clear. Subsequently, these rules are expressed in OCL during the implementation phase in respect to the chosen implementation tool. In addition to the cardinality constraints already present in the class diagram, the other rules are:

1. All identifiers are strictly positive integers which include nodes, sources, orbits and finite capacity regions ( $id > 0$ ).
2. The identifier of a node is unique in the model.
3. A source of jobs directs its job flow to only one node.
4. A job flow of the same class and priority can't be directed more than once from one node to another, to the outside or to an Orbit. It means that identical job flows from the same node could not be directed to the same target (Node, Orbit or Outside).

5. For a job class, the network of queues can be either open or closed.
6. Sources and orbits of the same station must have distinct identifiers.
7. A probability is a real number comprised between 0 and 1 ( $0 \leq pr \leq 1$ ).
8. If the routing strategy in a given station for a specific job class is probabilistic, then the sum of probabilities is equal to one ( $\sum pr = 1$ ).
9. Job classes are positive integers (for Routing, Designation, Output and Outside), but for a Source or an Orbit, it is strictly positive (the class number 0 is reserved to represent any class not explicitly mentioned).
10. Only one Sink is authorized in a model (for open networks) for which at least one source must exist (no sink without sources).
11. An Orbit manipulates only one job class, so it must receive only this class from the corresponding node to which, as a Source, it directs jobs.
12. Finite Capacity Regions have distinct identifiers (identifiers are unique).
13. A node may belong to different finite capacity regions and a finite capacity region may contain different nodes.
14. The population attribute of a source of jobs is a positive integer (population  $\geq 0$ ). Zero means unlimited.

## 6. Concrete syntax and exchange format

Some DSL developers consider that in modern visual (or graphical) language environments there is no need to be very specific about concrete syntax. At the low level, information can be exchanged between different tools in a textual concrete syntax using XML, and at the high level, humans can input linguistic utterances into a tool using graphical components. In consequence, the only thing that remains is the need for renderings of models which are meaningful to humans. However, because a language is a means of communication that must preserve interoperability, all tools developers need to agree on the XML schema for interchange as well as on the symbols to be used in rendering. In addition, both the XML schema and the set of symbols for rendering are considered concrete syntaxes of the language. If no agreement on concrete syntaxes is done, lack of interoperability and confusion would arise. It is clear that a mapping from abstract syntax to a concrete syntax is as important as the inverse mapping.

It is possible to define several concrete syntaxes for the same abstract syntax. We adopted the derivation illustrated in Figure 2, matching each element of the abstract syntax with a visual (graphic) appearance. It is

important to provide visual elements as distinct (or similar) as their abstractions. The visual syntax must be as close as possible to the domain graphical notation. This will facilitate the task of domain experts, ensure large deployment and encourage acceptance of the DSML.



















Abstract syntax	Visual (concrete) syntax
Simple Station	
Asymmetric Station	
Allocation Station	
Release Station	
Source	
Sink	
Finite Capacity Region	
Decision	
Orbit	
Routing	
State	
Server	
Input/Output/Outside	
Distribution	
Service	
Affectation	
Designation	
Failure-Vacation	

Fig. 2 Mapping between abstract and visual concrete syntax.

For simple languages and in most of the current DSM tools, concrete syntax representation is directly mapped to the abstract syntax. Logical entities are always visualized as nodes, and logical relationships as edges. Nevertheless the experience has shown that using more complex metamodelling, especially those conceived for automated model transformations, not only results in visual models being too complicated to overview, but it can also drain system resources heavily. Consequently, modern approaches make use of a separate visualization metamodelling, which describe the structural appearance of diagrams. The last technique allows hiding superfluous details; however it is still limited in the sense that classes can only be mapped to nodes and references to edges.

The problem with EQNM<sup>2</sup>L formalism is the exchange and reuse of developed models between various simulation and analysis tools. It is necessary to refer to an exchange format which must be open and promotes maximum interoperability.

Various textual exchange formats exist and most of them are based on XML. For example, QNAT software [17] uses the Mathematica format and RAQS software [2] uses a specific ASCII format. For JMT [16], it uses XML as a mean to describe models. The syntax is specified by the *JMTmodel* schema which serves as the unique format

used by the simulation and the analytical engines. The main problem here is not only the multiple used formats, but even if XML is used, the solution suffers from the lack of expressiveness and agreement on a standard format. The exchange format for EQNM<sup>2</sup>L [18] is a proposed draft which may serve as a basis for developing more adequate, stable and mature templates. It represents a metamodel described as an XML-schema. Each element of the metamodel is projected as an XML element. For the inheritance concept, XML provides restrictions and extensions that are similar and useful. In addition, some constraints could be expressed easily, especially those of uniqueness.

### 7. Implementation

One of the advantages of using the MDE approach is the gain in productivity where the task of developing tooling for domain-specific languages in a cost-effective manner is possible.

The modeling environment must offer at least visual syntax-aware editor to assist the user in model construction. Actually, several tools support domain metamodeling and domain application models, such as MetaEdit+ from the MetaCase Company [19], GME from ISIS laboratory of Vanderbilt University [20], Microsoft DSL Tools [5], AToM3 [21] as well as Eclipse EMF/Ecore [22] and Eclipse GMF [23].

Eclipse GMF and GME are appropriate tools to implement the EQNM<sup>2</sup>L modeling environment. Main advantages are that both are open source, freeware, mature, well documented and offer a sufficient set of constructs to define most aspects of the DSML including model validation and transformation. Actually, our project is based mainly on Eclipse platform, together with its EMF and the GEF plugins. The latter is a basic diagram drawing engine. In addition, the static metamodel mapping-driven GMF platform is the solution for linking the two previous plugins and complete tool building environment.

The domain model illustrated in Figure 3 covers only a subset of EQNM<sup>2</sup>L. OCL constraints are inserted in the EMF/Ecore model as shown in Figure 5 to allow model validation.

The Eclipse DSL Toolkit is based mainly on EMFs capabilities, including diagram definitions, transformation definitions and code-generation templates, in addition to model serialization and persistence. Many of these capabilities are developed using EMF models. For domain-specific modeling surfaces generation, Graphical Modeling Framework (GMF) is based on a collection of EMF models that are considered as DSLs themselves. GMF allows providing a graphical concrete syntax where

the proposed graphical notation is mapped to the abstract syntax.

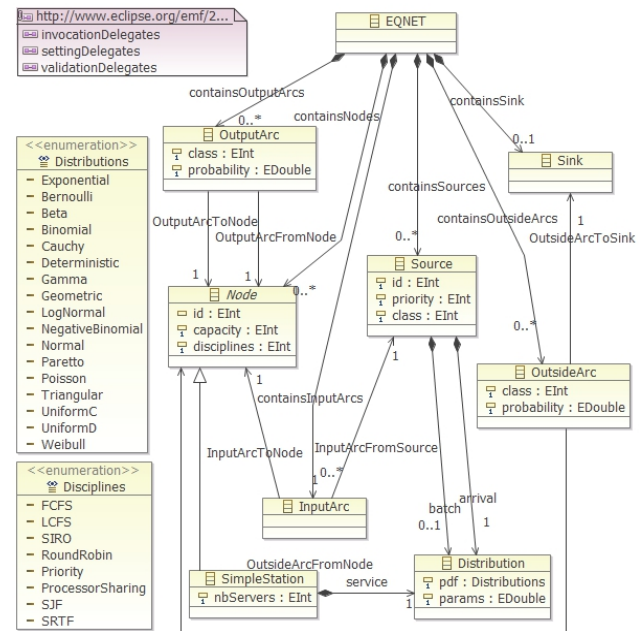


Fig. 3 Domain model as an Ecore diagram.

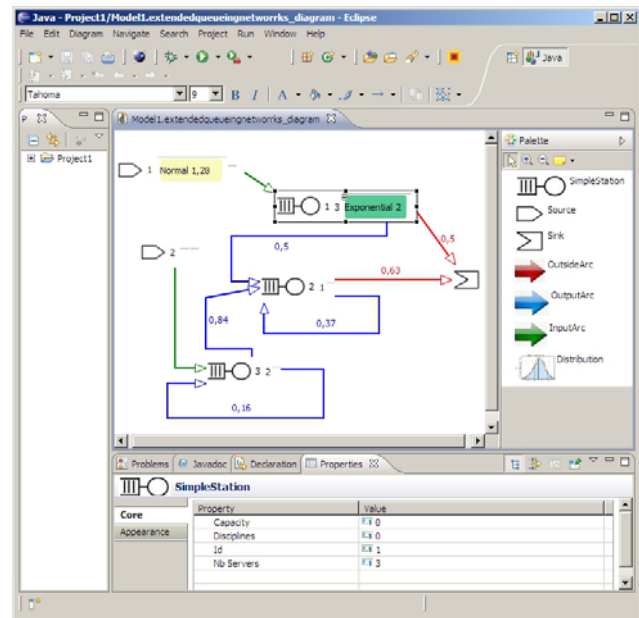


Fig. 4 EQNM<sup>2</sup>L diagram editor in Eclipse.

The EQNM<sup>2</sup>L diagram editor illustrated in Figure 4, offers a complete environment for graphical editing and model validation. The project is an Eclipse plugin and could be integrated easily.

Instead of coding model transformation rules, for code generation, directly in the programming language of



the application, as most software does, it is more practical to separate them and to lean on hard background of a formal theory.

```

36 class SimpleStation extends Node
37 {
38     invariant positiveNbServers: self.nbServers >= 0;
39     attribute nbServers : score_0:EInt[1] = '3';
40     property service : Distribution[1] { composes };
41 }
42 class Source
43 {
44     invariant positivePriorities: self.priority > 0;
45     invariant UniqueSourceIDs:
46     Source.allInstances()->forall(s1, s2 | s1 <> s2 implies s1._id <> s2._id);
47     invariant positiveClasses: self._class > 0;
48     invariant positiveIDs: self._id > 0;
49     attribute _id : score_0:EInt[1] = '1';
50     attribute priority : score_0:EInt[1] = '1';
51     attribute _class : score_0:EInt[1] = '1';
52     property batch : Distribution[?] { composes };
53     property arrival : Distribution[1] { composes };
54 }
55 class OutsideArc
56 {
57     invariant positiveClasses: self._class > 0;
58     invariant positiveProbabilities: self.probability >= 0 and probability <= 1;
59     attribute _class : score_0:EInt[1] = '1';
60     attribute probability : score_0:EDouble[1] = '0.25';
61     property OutsideArcToSink : Sink[1];
62     property OutsideArcFromNode : Node[1];
63 }
    
```

Fig. 5 EQNM<sup>2</sup>L OCL constraints.

Dynamic semantics of a DSML may be specified by the various model transformations. Operational semantics is well suited for directly executable or simulated models.

For EQNM<sup>2</sup>L, a model is either translated into a simulation/programming language to be executed or into an analysis tool specific format in order to be resolved. In the first case, the semantics domain is changed and transformation rules specify a denotational semantics.

The code generation process is a model transformation operation. More accurately, it is considered as model-to-text (M2T) transformation. The Xpand template language is an increasingly popular template engine, used extensively by the GMF project. It is originated with the OpenArchitectureWare component within GMT but has graduated to the M2T project.

Transformation rules are established according to the source and the target metamodels. EQNM<sup>2</sup>L models could be transformed into discrete event simulation code, expressed in a general purpose programming language (GPL) or a simulation language.

Several discrete event simulation libraries have been developed to offer a mean to assist programmers in writing customized and efficient simulation code. In the actual state of the work, a Java simulation code is generated automatically from EQNM<sup>2</sup>L conceptual models using Xpand templates illustrated in Figure 6. The Japrosim simulation library [24] has been chosen for code generation.

```

1 <<IMPORT ExtendedQueueingNetworks>>
2
3 <<EXTENSION template::GeneratorExtensions>>
4
5 <<DEFINE main FOR EQNET>>
6 <<FILE "QueueNetwork.java">>
7 import uoeb.japrosim.kernel.SimProcess;
8
9
10 public class QueueNetwork {
11     public static void main(String[] args){
12         SimProcess.sched.start();
13         <<FOREACH containsSources AS s>>
14         <<IF i.InputArcFromSource.id == s.id>>
15         <<LET i.InputArcToNode.id AS iid>>
16             new Transaction<s.id>(<iid>()).beginAfter(0.0);
17         <<ENDLET>><<ENDIF>><<ENDFOREACH>>
18         <<ENDFOREACH>>
19     }
20 }
21 <<ENDFILE>>
    
```

Fig. 6 Xpand-template For QueueNetwork class.

In order to facilitate understanding these rules, it is necessary to have an idea of how Japrosim works. For each model, a class named "QueueNetwork" is needed for system initialization by creating first events and then launching the simulation. The corresponding Xpand template is illustrated in Figure 6. Another class named "Transaction" which must extend the predefined "Entity" class is required. It is the place for jobs (or transactions) to declare needed shared resources as arrivals, services, routing schemes, stations, queues and shared behavior. Template in Figure 8 allows generating the previous Java class. In addition, each source of jobs requires its own Java class, named "Transaction<sub>i</sub>" that implements their specific behavior in the system. These Java classes are generated using the template shown in Figure 7.

```

69 <<DEFINE javaClass FOR Source>>
70 <<FOREACH ((EQNET) this.eContainer).containsInputArcs.typeSelect(InputArc) AS i>>
71 <<IF i.InputArcFromSource.id == this.id>>
72 <<LET ((SimpleStation) i.InputArcToNode).id AS iid>>
73
74 <<FILE "Transaction"+id+"_"+iid+".java">>
75 public class Transaction<id>_<iid> extends Transaction {
76     public Transaction<id>_<iid>() {
77         super();
78     }
79
80     public void body() {
81         new Transaction<id>_<iid>().beginAfter(arrival<id>.sample());
82         intoStation(
83         <<FOREACH ((EQNET) this.eContainer).containsInputArcs.typeSelect(InputArc) AS i>>
84         <<IF i.InputArcFromSource.id == this.id>>
85         <<((SimpleStation) i.InputArcToNode).id>>); <<ENDIF>><<ENDFOREACH>>
86     }
87 }
88 <<ENDFILE>><<ENDLET>><<ENDIF>><<ENDFOREACH>>
89 <<ENDDDEFINE>>
    
```

Fig. 7 Xpand-template For Transaction<sub>i</sub> classes.

The same work has been done in [25] based on the exchange format. The source model is expressed in XML which allows using XSLT for expressing transformation rules and helps maintaining and reusing the translation software for future versions or other simulation languages.

XSLT is a powerful XML-dialect for manipulating data in XML documents. It provides a set of operations and manipulators, while XPath provides precision in

locating elements and attributes. So in order to translate the conceptual model into an executable simulation model, an XSLT stylesheet and an XSLT processor are used.

The XML model description is transformed into the corresponding set of Java classes according to the

Japrosim library as discussed previously. The code generation module is extensible for other simulation languages or analysis tool format easily by associating the corresponding XSLT stylesheets. The user will be free to choose his target tool.

```
23 <<EXPAND javaClass FOREACH sources()>>
24 <<FILE "Transaction.java">>
25 import uceb.japrosim.kernel.*;
26 import uceb.japrosim.random.distributions.*;
27 public class Transaction extends Entity {
28 <<FOREACH containsSources AS s>>
29     static <<s.arrival.pdf>> arrival<<s.id>> = new <<s.arrival.pdf>>(<<s.arrival.params.first()>>
30     <<FOREACH s.arrival.params.withoutFirst().toList() AS p>>,<<p>><<ENDFOREACH>>);<<ENDFOREACH>>
31 <<FOREACH containsNodes.typeSelect(SimpleStation) AS ss>>
32     static <<ss.service.pdf>> serv<<ss.id>> = new <<ss.service.pdf>>(<<ss.service.params.first()>>
33     <<FOREACH ss.service.params.withoutFirst().toList() AS r>>,<<r>><<ENDFOREACH>>);
34     static Queue queue<<ss.id>> = new Queue("Queue 0<<ss.id>>");
35     static Resource server<<ss.id>> = new Resource("Station<<ss.id>>", <<ss.nbServers>>);
36     static int destination<<ss.id>>;
37     static Discrete outputs<<ss.id>> = new Discrete(new double[] {
38     <<FOREACH ((EQNET) (ss.eContainer)).containsOutputArcs.typeSelect(OutputArc) AS o>>
39     <<IF ss.id == o.OutputArcFromNode.id<<o.OutputArcToNode.id>>, <<ENDIF>><<ENDFOREACH>>0}, new double[] {
40     <<FOREACH ((EQNET) (ss.eContainer)).containsOutputArcs.typeSelect(OutputArc) AS o>>
41     <<IF ss.id == o.OutputArcFromNode.id<<o.probability>>, <<ENDIF>><<ENDFOREACH>>
42     <<FOREACH ((EQNET) (ss.eContainer)).containsOutsideArcs.typeSelect(OutsideArc) AS c>>
43     <<IF ss.id == c.OutsideArcFromNode.id<<c.probability>><<ENDIF>><<ENDFOREACH>>});<<ENDFOREACH>>
44     public Transaction() {
45         super();
46     }
47     public void intoStation(int i){
48 <<FOREACH containsNodes.typeSelect(SimpleStation) AS ss>>
49         if(i == <<ss.id>>){
50             queue<<ss.id>>.insert(this);
51             while (server<<ss.id>>.getAvailability() < 1) {
52                 passivate();
53             }
54             seize(server<<ss.id>>, 1);
55             queue<<ss.id>>.remove(this);
56             hold(server<<ss.id>>.sample());
57             release(server<<ss.id>>, 1);
58             int c = (int)outputs<<ss.id>>.sample();
59             if(c!=0){
60                 intoStation(c);
61             }
62         }<<ENDFOREACH>>
63     }
64 }
65 <<ENDFILE>>
```

Fig. 8 Xpand-template For Transaction class.

## 8. Conclusions

We presented EQNM<sup>2</sup>L, an extended queueing networks modeling and markup language where models are simply queueing networks with an extension for new concepts that help modeling more accurately a wider range of discrete event systems. Obtained models may be suited to analytical solutions, approximations, or simulation techniques.

As a DSML, it is tool-independent, extensible and well-established if we consider that it constitutes only a starting version. Its main advantages include interoperability enhancement at the conceptual modeling level through the proposed metamodel and the XML-based exchange format. A modeling environment as an Eclipse plugin is created guided by the proposed metamodel and the adopted visual syntax. Based on the MDE concepts, automation allows reducing the development time and effort and makes validation easier.

It is possible to define a common visual (concrete) syntax, but even if several rendering schemes are adopted by different tools, the metamodel and the exchange format remain unchanged.

The metamodel presented in this paper is a minimal version and several concepts are not yet considered. It could be extended easily, whenever a new concept has a clear semantics. In our opinion, it is much easier to conclude an agreement on a DSML than a specific tool. Therefore, EQNM<sup>2</sup>L modeling environment could be developed using any other tool that support domain metamodeling.

Domain experts must be involved in order to improve different aspects of this DSML. This includes more complete metamodel, more concise semantics and more adequate visual or concrete syntax. Furthermore the proposed exchange format must also reflect those efforts.

To ensure acceptability and large deployment, in addition to the adhering to the open source initiative to imply a large community, several code and model generators for the most used tools are to be considered.

Future work will focus also on the expressiveness improvement by including new concepts.

## References

- [1] U. M. Chalavadi, "Automatic configuration of queueing network models from business process descriptions". M. S. thesis. Oklahoma State University, USA, 2004.
- [2] M. Kamath, "Recent developments in modeling and performance analysis tools for manufacturing systems", in *Computer Control of Flexible Manufacturing Systems*, 1994, pp. 231-263.
- [3] P. Hudak, "Modular domain specific languages and tools", in the *Fifth International Conference on Software Reuse (JCSR98)*, 1998, pp. 134-142.
- [4] D. Spinellis, "Notable design patterns for domain-specific languages", *The Journal of Systems and Software*, Vol. 56, 2001, pp. 91-99.
- [5] S. Kelly, and J. P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley and Sons, Hoboken, New Jersey, 2008.
- [6] S. Cook, G. Jones, S. Kent, and A. C. Wills, *Domain-Specific Development with Visual Studio DSLTools*. Addison-Wesley, 2007.
- [7] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages". Report *Software Engineering SEN-E0309*, Stichting Centrumvoor Wiskunde en Informatica (CWI), Amsterdam, the Netherlands 2003.
- [8] W. Frakes, R. Prieto-Diaz, and C. Fox, "DARE: Domain analysis and reuse environment". *Annals of Software Engineering*, 1998, Vol. 5, pp. 125-141.
- [9] R. N. Taylor, W. Tracz, and L. Coglianese, "Software development using domain-specific software architectures", in *ACM SIGSOFT Software Engineering Notes*, 1995, Vol. 20, N° 5, pp. 27-37.
- [10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study". Technical Report *CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University 1990.
- [11] G. Bolsh, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, Second Edition. John Wiley & Sons, 2006.
- [12] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Model*. Prentice-Hall Inc, Englewood Cliffs: New Jersey, 1984.
- [13] R. B. Cooper, *Introduction to Queueing Theory*. Elsevier North Holland, Inc. 1981.
- [14] N. Tian, and Z. G. Zhang, *Vacation Queueing Models: Theory and Applications*. International Series in Operations Research & Management Science, Frederick S. Hillier, Series Editor, Stanford University Springer Science + Business Media, LLC. 2006.
- [15] R. Artalejo Jess, and A. Gmez-Corral, *Retrial Queueing Systems: A Computational Approach*. Springer-Verlag: Berlin Heidelberg. 2008.
- [16] M. Bertoli, G. Casale, and G. Serazzi, "JMT: performance engineering tools for system modeling", *ACM SIGMETRICS Performance Evaluation Review*, 2009, Vol. 36, Issue 4, New York, USA, March. ACM press, pp. 10-15.
- [17] D. Manjunath, D. M. Bhaskar, H. Tahilramani, S. K. Bose, and M. N. Umesh, "QNAT: A Graphical Tool for the Analysis of Queueing Networks", in *TENCON '98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, 1998, Vol. 2, pp. 320-323.
- [18] EQNM<sup>2</sup>L: <http://sourceforge.net/projects/eqnm2l/>. Accessed on November 15<sup>th</sup> 2011.
- [19] M. Rossi, and S. Kelly, "Construction of a CASE tool: the case for MetaEdit+", in *Proceedings of The First International Symposium on Constructing Software Engineering Tools (CoSET'99)*, 17-18 May 1999, Los Angeles.
- [20] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. T. IV, G. Nordstrom, J. Sprinkle, and P. Volgyesi. "The Generic Modeling Environment", In *Workshop on Intelligent Signal Processing*, Budapest, Hungary, May 17, 2001.
- [21] J. de Lara, H. Vangheluwe, "AToM3 as a Meta-CASE environment (DFD to SC)", in *4th International Conference On Enterprise Information Systems. Universidad de Castilla-La Mancha-Ciudad Real -Spain-* 3-6 April, 2002.
- [22] F. Budinsky, D. Steinberg, Ed.Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework: A Developer's Guide*. AddisonWesley, 2003.
- [23] Gronback. Richard C. *Eclipse Modeling Project: A Domain-Specific Language Toolkit*. Addison-Wesley Professional, 1st ed, 2009.
- [24] A. Bourouis, and B. Belattar. "JAPROSIM: A Java Framework for Discrete Event Simulation". *Journal of Object Technology*, Vol. 7, No. 1, January-February 2008, pp. 103-119.
- [25] A. Bourouis, and B. Belattar, "Using XML in Simulation Modelling: automatic code generation for XML-based models", in *Proceedings of the CARI 08, Tanger, Morocco*, October 23-25, 2008, pp 101-108.

**Abdelhabib BOUROUIS** received his BS degree in Computer science from the University of Constantine (Algeria) in 1999, his MS degree and Ph.D. degree respectively in 2003 and 2011 both from the University of Batna (Algeria). He is a Senior Lecturer at the University of Oum El Bouaghi (Algeria) since 2003. His research interests include Artificial intelligence, Model Driven Engineering, performance evaluation, parallel and distributed simulation.

**Brahim BELATTAR** received his BS degree in Computer science from the University of Constantine (Algeria) in 1981 and his MS and PhD degrees from the University Claude Bernard of Lyon (French) respectively in 1986 and 1991. He is an Associate professor at the University of Batna since 1992. He has also taught at the University of Constantine from 1982 to 1985. His research interests include simulation, databases, semantic web, Model Driven Engineering and Artificial intelligence.