A Model Transformation Tool: PMIF+ to QNAP

Catalina M. Lladó, Pere Bonet
Universitat de les Illes Balears
Dep. de Ciències Matemàtiques i Informàtica
Palma de Mallorca, Spain
cllado@uib.es, pere.bonet@gmail.com

Connie U. Smith
Performance Engineering Services
PO Box 2640
Santa Fe, NM 87504-2640 USA
www.spe-ed.com

ABSTRACT

An extension to the Performance Model Interchange Format (PMIF+) to move models among tools has been recently introduced. The original PMIF was limited to models solvable with efficient, exact solution methods such as mean value analysis and product form solutions. The extensions allow models with features that require simulation or analytical approximation solutions, such as passive resources, workloads that fork into multiple concurrent workloads, etc. This paper presents a tool for the automated transformation from PMIF+ to Qnap, and explains some of the challenging transformations required by the new PMIF+ features.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; C.4 [**Modelling Techniques**]: Experimentation

General Terms

Performance, Experimentation

Keywords

Interchange Format, Performance Models, Model Driven Performance Engineering, SPE, Tool interoperability.

1. INTRODUCTION

The Performance Model Interchange Format (PMIF) is a common representation of input required by all tools that provide Queueing Network Models (QNM) solutions [7]. More generally, MIFs (Model Interchange Formats) provide the basis for easily transferring the model information among modeling tools with either one import/export interface per tool, or by an automated transformation from the MIF to a tool's input language. The QNM models may be created by translating design models into performance models [1, 6]. They may be created by a tool that provides a graphical user interface for specifying the model topology and parameters then creates model interchange files. They may also be exported by one modeling tool in order to compare results to other modeling tools.

VALUETOOLS 2014, December 09-11, Bratislava, Slovakia
Copyright © 2015 ICST 978-1-63190-057-0
DOI 10.4108/icst.valuetools.2014.258219

The original PMIF was limited to models that could be solved by efficient, exact algorithms (e.g., mean value and product form solutions) to focus on the feasibility of the model interoperability approach. Once feasibility was established, PMIF+ [2] extended the scope to allow analytical approximations or simulation solutions. Qnap [4, 5] is one of the few modeling tools that solves models with all of the features in PMIF+.

Some of the new features can be quite challenging to represent in QNM, such as workloads that fork into multiple concurrent workloads, and requests for passive resources that may require queueing for the resource but no active service from it. Demonstrating the viability of PMIF+ requires creating and solving models with these complexities. This paper presents a tool that transforms models from PMIF+ to Qnap's input format and illustrates how to transform models with these complex features. This information is also useful for anyone who would like to create an import/export or transformation for other modeling tools.

The PMIF+ meta-model is specified using the Eclipse Modelling Framework (EMF) which allows for Model-to-model (M2M) and Model-to-Text (M2T) transformations. The PMIF+2Qnap transformation is developed using Acceleo [3] which is a code generator based on templates that implements the OMG's M2T specification.

2. PMIF+

The PMIF+2Qnap transformation tool's input is a PMIF+ [2] model. PMIF+ extends the scope of PMIF supported models to include features, such as

1. Fork/Split/Join Workload - A workload may *Fork* or *Split* into one or more child workloads that execute concurrently. Forked workloads later *Join*; the parent workload waits until all child workloads *Join*, then the parent workload resumes execution. *Split* workloads do not join, they eventually complete and leave the system.

2. Call/Accept/Return Synchronization Point - A workload may *Call* another workload and wait for the called workload to signal that it has completed the request; the called workload *Accepts* the request and *Returns*

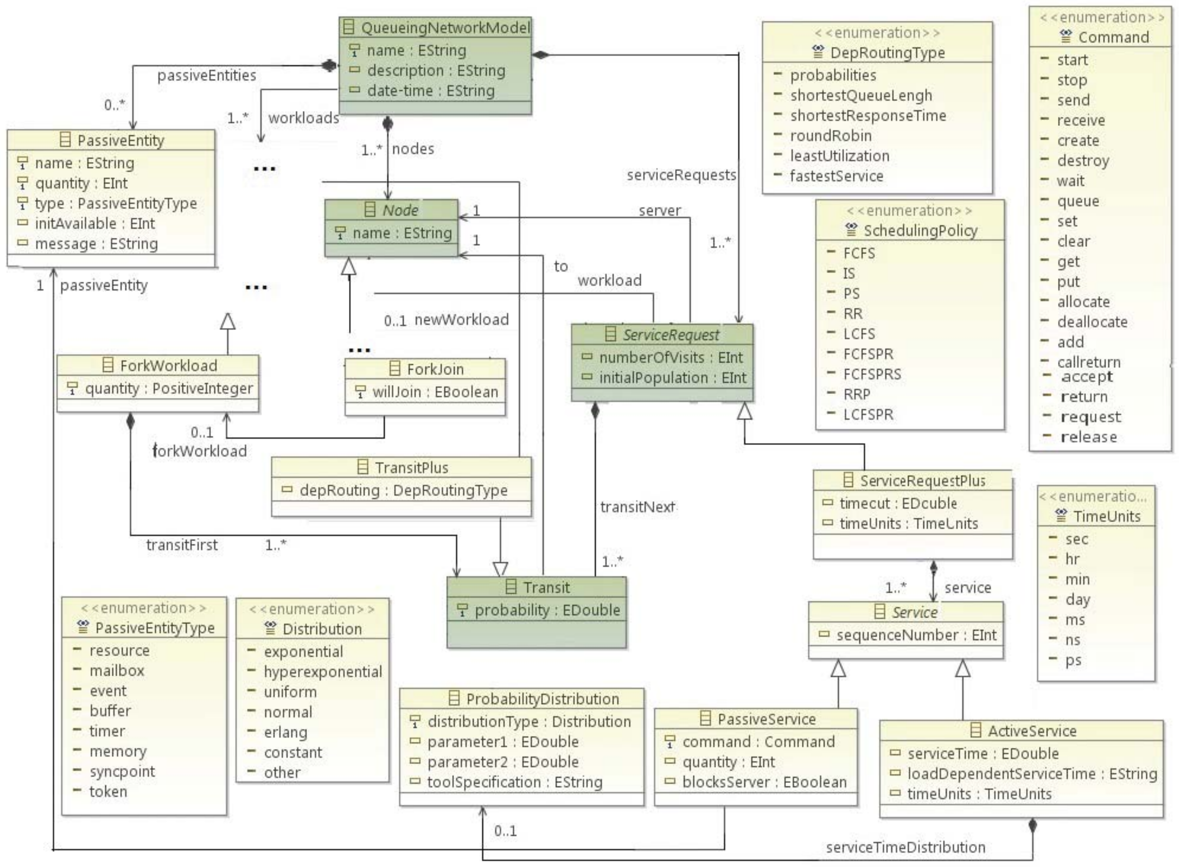


Figure 1: PMIF+ meta-model

to the waiting workload.

3. Allocate/Deallocate Resource - When access to a passive resource is restricted, a workload may request access and wait in a queue until the resource is *Allocated*. When access to the resource is no longer needed the workload *Deallocates* the resource. A scheduling policy determines the next workload to receive the Allocation.

4. Wait/Queue/Set Event - An Event may be *Set* or *Cleared*. Workloads may *Wait* or *Queue* for an event to be *Set*. When an event is *Set*, all waiting workloads and one queued workload may proceed.

Items 2-4 are examples of features that use *PassiveEntities*: resources that are required for execution but provide no active service themselves.

A broader set of arrival and service distributions as well as queue scheduling disciplines, including priorities, is included. The complete list is defined and represented in the PMIF+ meta-model in [2]. An excerpt presenting the new PMIF+ features is shown in Fig 1. The elements that are added for PMIF+ are yellow in the figure (or not shaded gray if viewed in black and white), some of the original PMIF elements are shown in green (or shaded grey) for the comprehension of the diagram, others are missing due to lack of space. We describe a few of the new features in detail in this section as background for the description of the transformation in the next section.

A *QueueingNetworkModel* now has zero or more *PassiveEntities* in addition to *Nodes*, *Workloads* and *ServiceRequests*. There is a new *Node* called *ForkJoin* to handle Fork and Join operations. When $willJoin = T$, it is a Fork and the parent waits for all children (*ForkWorkloads*, see below) to complete and return to the same node for Join. When $willJoin = F$, it is a Split; the parent continues execution and the children workloads exit the system upon completion. The *Server* has an extended schedulingPolicy as shown in Fig. 1. There is also a new type of *Workload*, *ForkWorkload* that represents a child workload with a maximum population that is created by its parent at a *ForkJoin* node.

A new *ServiceRequest*, *ServiceRequestPlus*, specifies a combination of active and passive service requests that can be made at any node with an optional *sequenceNumber* which specifies their order of execution when ordering is required (sequence numbers are increasing but need not be consecutive).

ActiveService requests specify a service time similar to the *TimeServiceRequest*. They may use a special *Probability-Distribution* or a load dependent service time. The latter is specified as a string which will be interpreted by the tool. *PassiveService* requests specify the *command*, the quantity, and reference the *PassiveEntity*. Table 1 shows the types of *PassiveEntity* and the commands associated with each.

Note that *PassiveService* does not normally block the server, it only blocks the workload. It has an optional attribute, *blocksServer* which is set to *True* when the server needs also to be blocked.

Pas. Entity	Commands
timer	start/stop
event	wait/queue/set/clear
mailbox	send/receive
buffer	get/put/create/destroy
resource	allocate/deallocate
memory	allocate/deallocate/add
token	wait/queue/create/destroy
syncpoint	callreturn/accept/return

Table 1: *ServiceRequestPlus* attribute options

3. COMPLEX TRANSFORMATIONS

The PMIF+2Qnap tool (Fig. 2) uses the Eclipse Modeling Framework (EMF). The tool takes a model that conforms to the PMIF+ meta-model and transforms it into the Qnap input language. The transformations are implemented with Acceleo [3] which is a code generator based on templates that implement the OMG's M2T specification. Acceleo is also fully integrated in the EMF framework. Therefore the transformation specification is created as a file with extension *mtl*. Fig. 2 shows how to execute a transformation with Acceleo, with part of the *mtl* specification file for the transformation also on the screen.

The *ServiceRequestPlus* is the key element that extends the scope of supported models. The *ForkJoin* and *ForkWorkload* are key to synchronization features of models. Therefore, these two transformations are described next.

3.1 ServiceRequestPlus

A *ServiceRequestPlus* (SRP) can contain several *Service* elements which can be active or passive. The *PassiveService* elements can also be blocking or non-blocking (which is an attribute). Figure 3 summarizes the main steps of a ServiceRequestPlus transformation.

The way that Qnap can implement a *PassiveService* is using an extra station for each server where the customers will go while the *PassiveService* occurs. A *PassiveService* is the execution of a *Command* on a *PassiveEntity*. Table 1 shows the types of *PassiveEntity* and the commands associated with each of them. Table 2 shows how Qnap can implement the different passive entities and commands and so the way it is implemented by the transformation.

In order to develop a correct transformation, several approaches were studied. Fig. 4 shows the final approach. The transformation generates two Qnap stations for each *ServiceRequestPlus*, one for the active services and one for the

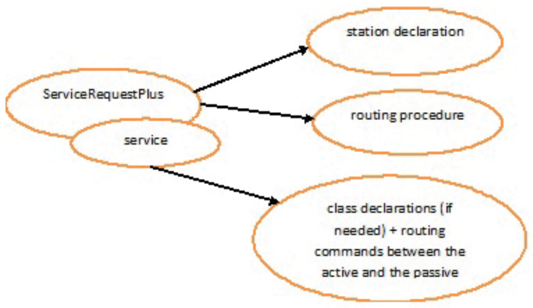

station declaration
ServiceRequestPlus
routing procedure
service
class declarations (if needed) + routing commands between the active and the passive

Figure 3: *ServiceRequestPlus* transformation steps

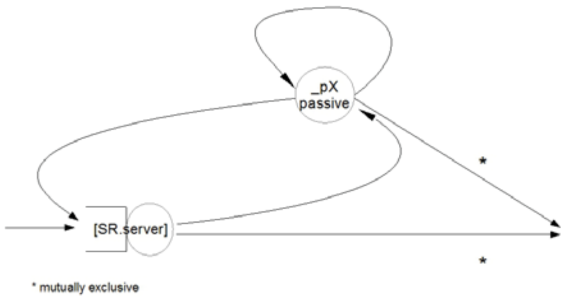


Figure 4: SRP transformation approach

passive services. The active station always receives the incoming workload. If the first service is passive, the workload receives no service and it is routed to the passive station as shown in Fig. 4. However, there is still a problem: how to treat sequences of services in which after an active service multiple passive services block the server, and this may occur multiple times. The solution adopted groups the consecutive services that are either active or passive with the attribute *blocksServer* true and they are performed in the active station whereas the passive services with *blocksServer* false are performed one by one in the passive station. In other words, in the resulting structure of the transformation the transitions (transits in Qnap) between the active station and the passive station are determined by the presence of non-blocking passive services as shown in Fig 5. Therefore sequences of services (either active services or passive services that are blocking) on the same server are grouped for them to happen sequentially.

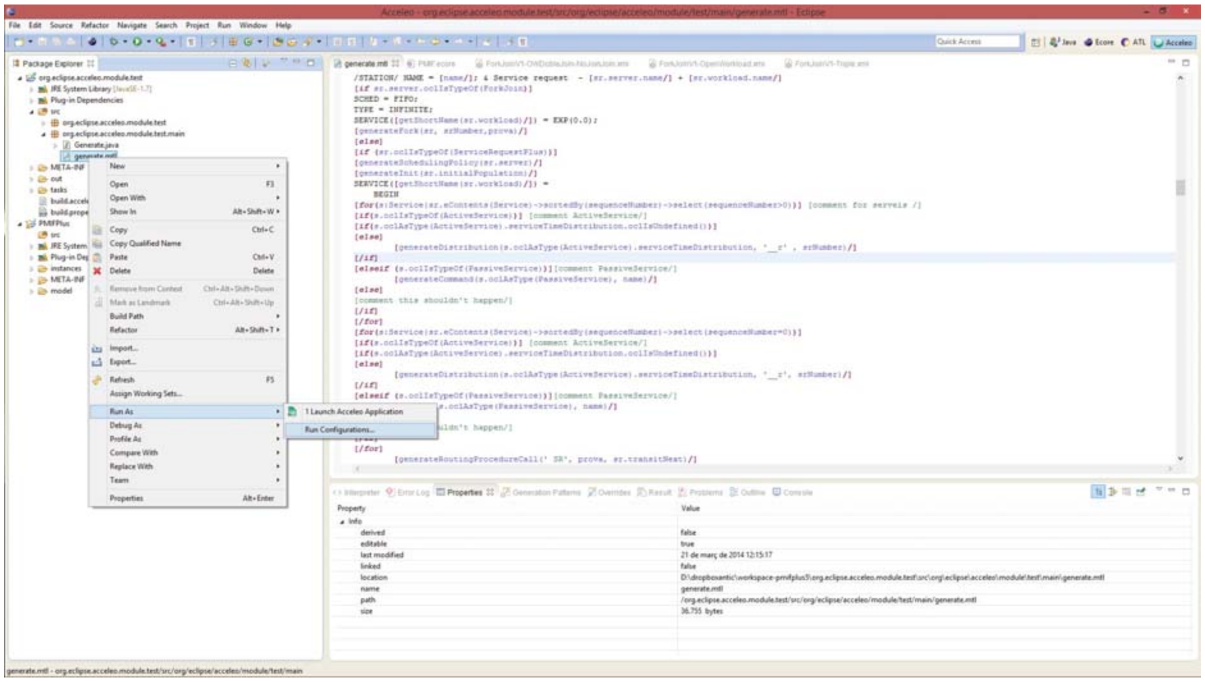
3.2 ForkJoin

For a *ForkJoin* node, the transformation generates (as shown in Fig. 6):

1. a *splitter* station, where the Qnap *SPLIT* command is performed,

2. a *router* station for each type of customer created with the *splitter* station, and

3. a *fork* station to model the exit of the *ForkJoin*. When the *WillJoin* attribute is true, a Qnap *MATCH* operation is performed at the *fork* station. If *WillJoin* is



```
/STATION/ NAME = [name/]; & Service request  - [sr.server.name/] + [sr.workload.name/]
[if sr.server.oclIsTypeOf(ForkJoin)]
SCHED = FIFO;
TYPE = INFINITE;
SERVICE([getShortName(sr.workload)/]) = EXP(0.0);
[generateFork(sr, srNumber,prova)/]
[else]
[if (sr.oclIsTypeOf(ServiceRequestPlus))]
[generateSchedulingPolicy(sr.server)/]
[generateInit(sr.initialPopulation)/]
SERVICE([getShortName(sr.workload)/]) =
    BEGIN
[for(s:Service|sr.eContents(Service)->sortedBy(sequenceNumber)->select(sequenceNumber>0))] [comment for serveis /]
[if(s.oclIsTypeOf(ActiveService))] [comment ActiveService/]
[if(s.oclAsType(ActiveService).serviceTimeDistribution.oclIsUndefined())]
[else]
        [generateDistribution(s.oclAsType(ActiveService).serviceTimeDistribution, '__r' , srNumber)/]
[/if]
[elseif (s.oclIsTypeOf(PassiveService))][comment PassiveService/]
        [generateCommand(s.oclAsType(PassiveService), name)/]
[else]
[comment this shouldn't happen/]
[/if]
[/for]
[for(s:Service|sr.eContents(Service)->sortedBy(sequenceNumber)->select(sequenceNumber=0))]
[if(s.oclIsTypeOf(ActiveService))] [comment ActiveService/]
[if(s.oclAsType(ActiveService).serviceTimeDistribution.oclIsUndefined())]
[else]
        [generateDistribution(s.oclAsType(ActiveService).serviceTimeDistribution, '__r', srNumber)/]
[/if]
[elseif (s.oclIsTypeOf(PassiveService))][comment PassiveService/]
                s.oclAsType(PassiveService), name)/]

                uldn't happen/]

[/for]
        [generateRoutingProcedureCall(' SR', prova, sr.transitNext)/]
```

Property	Value
Info	
derived	false
editable	true
last modified	21 de març de 2014 12:15:17
linked	false
location	D:\dropboxantic\workspace-pmifplus3\org.eclipse.acceleo.module.test\src\org\eclipse\acceleo\module\test\main\generate.mtl
name	generate.mtl
path	/org.eclipse.acceleo.module.test/src/org/eclipse/acceleo/module/test/main/generate.mtl
size	36.755 bytes

Acceleo - org.eclipse.acceleo.module.test/src/org/eclipse/acceleo/module/test/main/generate.mtl - Eclipse
Package Explorer
generate.mtl
Run As
1 Launch Acceleo Application
Run Configurations...
Properties

Figure 2: Execution of a transformation with Acceleo

Passive Entity	Qnap entity	Commands in Qnap
timer	timer	SETTIMER:ABSOLUTE/SETTIMER:CANCEL
event	flag	WAIT/WAIT/FREE/RESET
mailbox	Resource (Multiple) + Flag	V/P
buffer	Semaphore (Multiple)	PMULT/VMULT/VMULT/PMULT
resource	Semaphore (Multiple)	P/V
memory	Semaphore (Multiple)	PMULT/VMULT
token	Semaphore (Multiple)	WAIT/WAIT/VMULT/PMULT
syncpoint	Semaphore (Multiple)	P+ FREE + WAIT/V + WAIT/FREE

Table 2: Passive entities implementation in Qnap

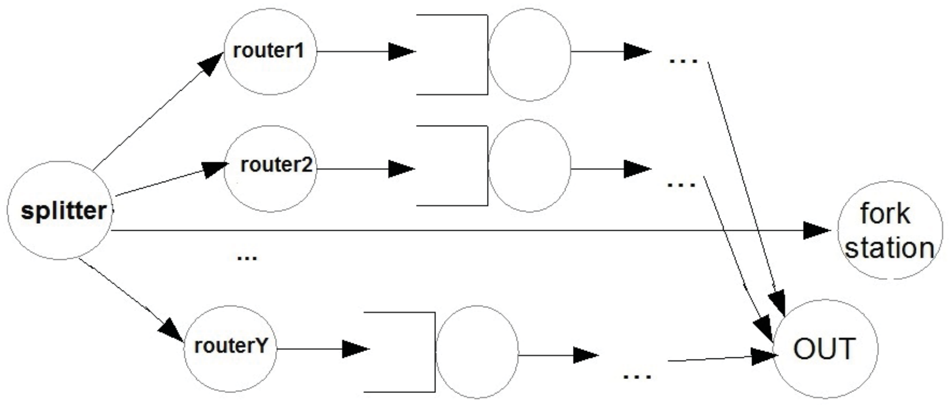


Figure 6: *ForkJoin* transformation (*WillJoin* = F)

false, each son exits the system after its itinerary has ended and the original client is sent directly from the *splitter* station to the *fork* station.

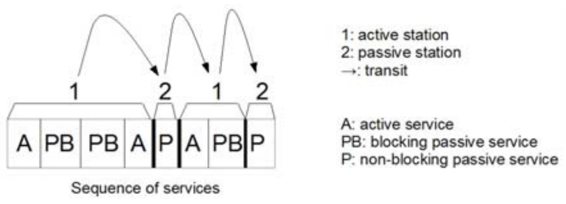


Figure 5: Service grouping for *ServiceRequestPlus*

4. CONCLUSIONS

This paper presents the PMIF+2Qnap transformation tool which has been developed using the Acceleo Model to Text (M2T) transformation tool with Eclipse EMF. It demonstrates how to convert the complex extensions for passive resources and fork-join of workloads for those who would like to interface their tool(s) with PMIF+. Models with the PMIF+ extensions have been transformed and solved with Qnap thus establishing the viability of the extensions. The complete specification of the PMIF+ meta-model as well as the tool source code and some examples can be found at `www.mifs.cat/pmif+/`. A short tutorial on how to use Acceleo with Eclipse EMF can also be found at the same url.

Acknowledgments

Smith's participation was sponsored by US Air Force Contract FA8750-14-C-0152, approved for public release with clearance number 88ABW-2014-4880.

5. REFERENCES

[1] S. Balsamo and M. Marzolla. Performance evaluation of UML software architectures with multiclass queueing network models. In *Proc. of the Fifth International Workshop of Software and Performance (WOSP)*, July 2005.

[2] C. M. Lladó and C. U. Smith. Pmif+: Extensions to broaden the scope of supported models. In *Computer Performance Engineering LNCS 8168. Proc. of the 10th European Workshop, EPEW 2013*, 2013.

[3] OBEO. Acceleo. `www.eclipse.org/acceleo/`.

[4] D. Potier and M. Veran. Qnap2: A portable environment for queueing systems modelling. In D. Potier, editor, *First International Conference on Modeling Techniques and Tools for Performance Analysis*, pages 25–63. North Holland, May 1985.

[5] Simulog. *QNAP2 9.3: Reference Manual*, 1996.

[6] C. U. Smith, V. Cortellessa, A. Di Marco, C. M. Lladó, and L. G. Williams. From uml models to software performance results: An SPE process based on XML interchange formats. In *Proc. of the Fifth International Workshop on Software and Performance (WOSP)*, pages 87–98, July 2005.

[7] C. U. Smith, C. M. Lladó, and R. Puigjaner. Performance Model Interchange Format (PMIF 2): A comprehensive approach to queueing network model interoperability. *Performance Evaluation*, 67(7):548 – 568, 2010.