Resource Contention Analysis of Cloud-based Systems through fUML-driven Model Execution ** Additional Material **

VARIOUS AUTHORS

July 31, 2013

1 Introduction to Additional Materials.

This documents contains some additional materials that better illustrate the content of the paper

Resource Contention Analysis of Cloud-based Systems through fUML-driven Model Execution

currently submitted for revision to international venues.

This is a working document subject to frequent changes/updates according with the progresses of the project.

The rest of the document is organized as follows. Section 2 introduces the *PetStore* UML Model. Section 3 illustrate the initial comparison between our Performance Analyzer tool and JSIMgraph (JSIM), a queueing network models simulator with graphical user interface included in the Java Modelling Tools (JMT) suite[1].

2 Petstore UML Model.

The *PetStore* UML Model has been modeled using Papyrus UML[2]. The papyrus project is available at `www.modelexecution.org`.
The following figures have been created from the same model imported in MagicDraw[3].
The *PetStore* software architecture is then integrated with a set of UML activities that provides an executable specification for all the operations provided by the software services. For example, Figures 2 and 3 show the activity associated with the *login()*

[1] `http://jmt.sourceforge.net/`
[2] `http://www.eclipse.org/papyrus/`
[3] `www.magicdraw.com`

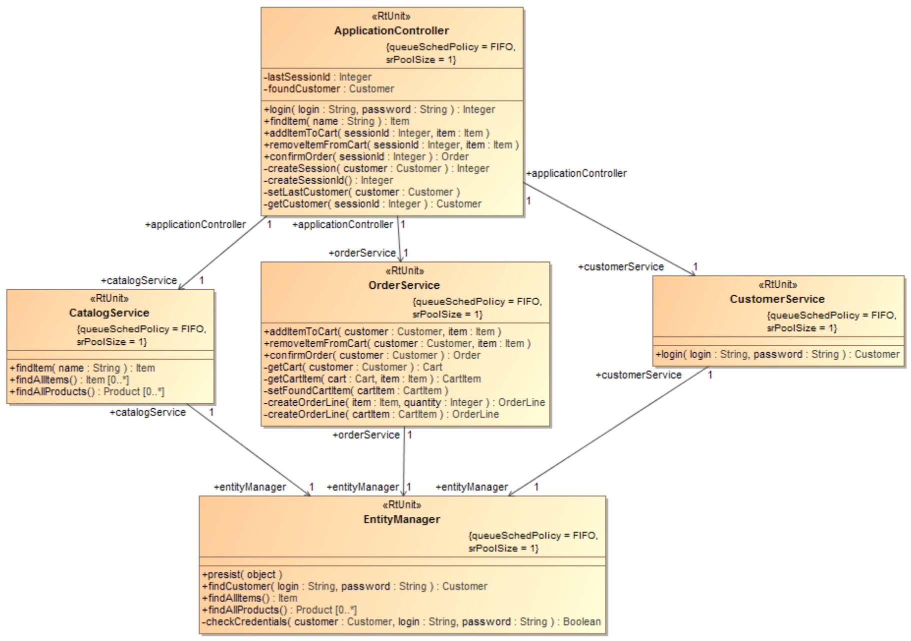

«RtUnit»
ApplicationController
{queueSchedPolicy = FIFO, srPoolSize = 1}
-lastSessionId : Integer
-foundCustomer : Customer
+login(login : String, password : String) : Integer
+findItem(name : String) : Item
+addItemToCart(sessionId : Integer, item : Item)
+removeItemFromCart(sessionId : Integer, item : Item)
+confirmOrder(sessionId : Integer) : Order
-createSession(customer : Customer) : Integer
-createSessionId() : Integer
-setLastCustomer(customer : Customer)
-getCustomer(sessionId : Integer) : Customer
+applicationController 1
+applicationController 1
+applicationController 1
+catalogService 1
+orderService 1
+customerService 1
«RtUnit»
CatalogService
{queueSchedPolicy = FIFO, srPoolSize = 1}
+findItem(name : String) : Item
+findAllItems() : Item [0..*]
+findAllProducts() : Product [0..*]
«RtUnit»
OrderService
{queueSchedPolicy = FIFO, srPoolSize = 1}
+addItemToCart(customer : Customer, item : Item)
+removeItemFromCart(customer : Customer, item : Item)
+confirmOrder(customer : Customer) : Order
-getCart(customer : Customer) : Cart
-getCartItem(cart : Cart, item : Item) : CartItem
-setFoundCartItem(cartItem : CartItem)
-createOrderLine(item : Item, quantity : Integer) : OrderLine
-createOrderLine(cartItem : CartItem) : OrderLine
«RtUnit»
CustomerService
{queueSchedPolicy = FIFO, srPoolSize = 1}
+login(login : String, password : String) : Customer
+catalogService 1
+orderService 1
+customerService 1
+entityManager 1
+entityManager 1
+entityManager 1
«RtUnit»
EntityManager
{queueSchedPolicy = FIFO, srPoolSize = 1}
+presist(object)
+findCustomer(login : String, password : String) : Customer
+findAllItems() : Item
+findAllProducts() : Product [0..*]
-checkCredentials(customer : Customer, login : String, password : String) : Boolean

Figure 1: PetStore Software Architecture.

and *checkCredentials()* operations of the ApplicationController and EntityManager services, respectively, as they appear once modeled in Papyrus UML.

Hardware Platform The *PetStore* UML model also includes the modeling of the *PetStore* hardware platform. It is composed of a unique execution host, where all the *PetStore* services as shown in Figure 1 are deployed on.
The hardware platform is shown in Figure 4 where the software services depicted in Figure 1 are allocated on the *PetStore* execution host through *allocate relationships* pointing from the services to the execution host. The execution host is equipped with its an hardware processor (*CPU*) [4] which is capable of executing a certain millions of assembly instructions per second (MIPS)[5]. The reference *PetStore* hardware platform includes a unique CPU capable of executing $200,000$ MIPS.

Estimation of execution time of services' operations. We then use MIPS to obtain an early estimation of the execution time of the operations provided by the software services allocated on the *PetStore* execution host. Such an estimation is done according an *overhead matrix* as defined in [2].

[4]For sake of simplicity, we do not consider storage and communication resource. However, the interested reader can refer to [1] for a more detailed modeling of the hardware platform.

[5]see `http://en.wikipedia.org/wiki/Instructions_per_second` for MIPS calculated for several real hardware processors.

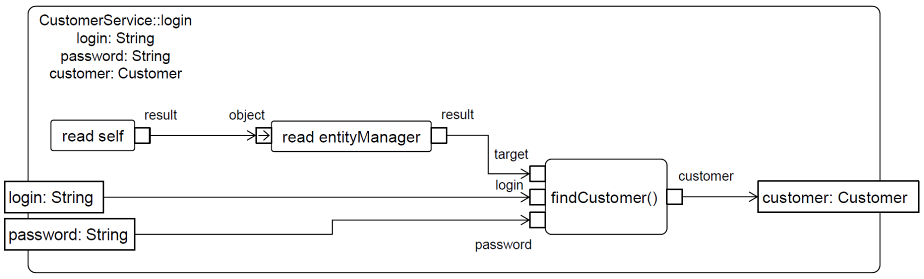


Figure 2: The UML Activity associated to the login() operation of the ApplicationController.

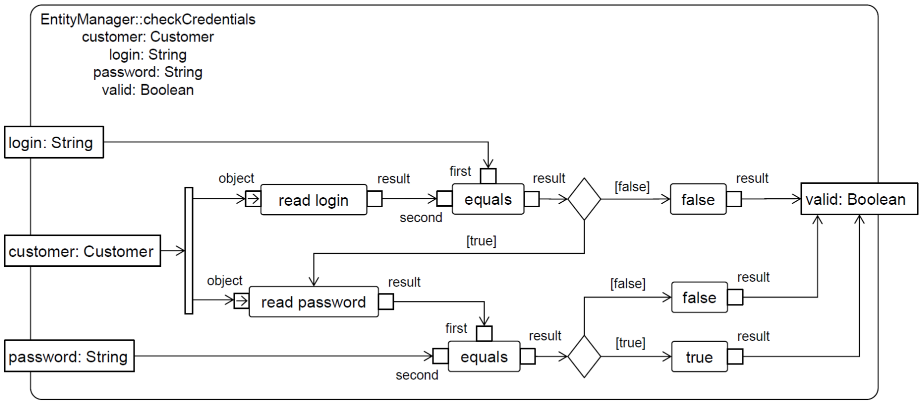


Figure 3: The UML Activity associated to the checkCredentials() operation of the EntityManager.

An overhead matrix contains the software-to-hardware unit conversion factors that are used to calculate the execution time required to execute a software behavioral unit (e.g., a CallOperation action) demanding for certain hardware resources installed on execution hosts.
In our case study, the execution host depicted in Figure 4 provides a CPU capable of executing a certain amount of MIPS. We then measured the complexity of the operations provided by the *PetStore* software services as a number of high level instructions executed for their invocation (e.g. number of java instructions). We then set two distinct unit conversion factors, i.e., (i) from assembly-level to bytecode-level (x20) and (ii) from bytecode level to high-level (x25) as show in Table 1. The accuracy of this kind of estimation depends on the accuracy of such factors.
It is worth noting that we apply the same multiplying factor for each type of instruction when moving (that is *compiling*) it from one level to another. This is an approximation that can be avoided by specifying a distinct factor for different types of instructions.

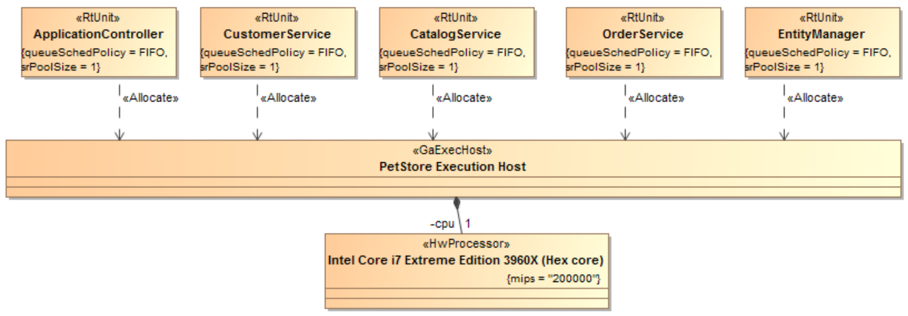

«RtUnit»
ApplicationController
{queueSchedPolicy = FIFO, srPoolSize = 1}
«RtUnit»
CustomerService
{queueSchedPolicy = FIFO, srPoolSize = 1}
«RtUnit»
CatalogService
{queueSchedPolicy = FIFO, srPoolSize = 1}
«RtUnit»
OrderService
{queueSchedPolicy = FIFO, srPoolSize = 1}
«RtUnit»
EntityManager
{queueSchedPolicy = FIFO, srPoolSize = 1}
«Allocate»
«Allocate»
«Allocate»
«Allocate»
«Allocate»
«GaExecHost»
PetStore Execution Host
-cpu
1
«HwProcessor»
Intel Core i7 Extreme Edition 3960X (Hex core)
{mips = "200000"}

Figure 4: PetStore Hardware Platform.

Table 1: Overhead Matrix for the *PetStore*.

->	MIPS L1	MIPS L2	MIPS L3
MIPS L1	1.0000	0.0500	0.0020
MIPS L2	20.0000	1.0000	0.0400
MIPS L3	500.0000	25.0000	1.0000

Legend:	MIPS L1	assembly level instruction
	MIPS L2	bytecode level instruction
	MIPS L3	high level instruction

According to these conversion factors we obtained an early estimation for the operation calls involved in the The *PetStore Single Buy Scenario* in Figure 5.
The estimated execution times are shown in Table 2

3 Comparison of Performance Analyzer with JMT.

In order to verify the accuracy of the analysis results obtained by our Performance Analyzer (PA), we plan to compare its results with ones from existing QN solvers. Among them, our first choice is JSIMgraph (JSIM), a QN models simulator with graphical user interface, a continuously up to date documentation and very responsive support from its developers. JSIMgraph is included in the Java Modelling Tools (JMT) suite [6].
JMT includes a set of advanced queuing network analyzers and simulators. Our aim is not the re-implementation of the whole set of algorithms and functionalities provided by JMT. We consider JMT as a reference implementation from which we select a very small set of basic functionalities that are indispensable to calculate a minimum set of performance figures from inputs to our framework, like:

- Utilization (U) and Throughput (T) of each single service center.

[6]`http://jmt.sourceforge.net/`

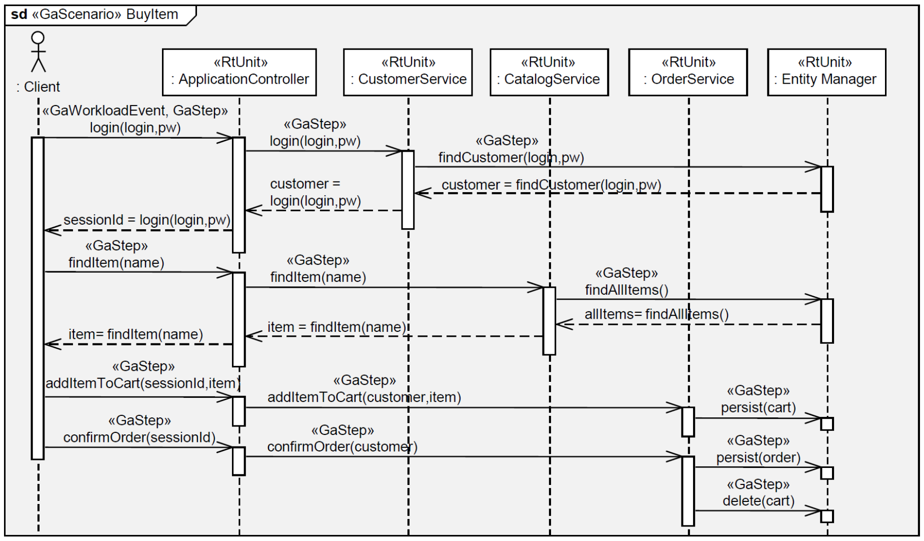

sd «GaScenario» BuyItem
: Client
«RtUnit» : ApplicationController
«RtUnit» : CustomerService
«RtUnit» : CatalogService
«RtUnit» : OrderService
«RtUnit» : Entity Manager
«GaWorkloadEvent, GaStep» login(login,pw)
«GaStep» login(login,pw)
«GaStep» findCustomer(login,pw)
customer = findCustomer(login,pw)
customer = login(login,pw)
sessionId = login(login,pw)
«GaStep» findItem(name)
«GaStep» findItem(name)
«GaStep» findAllItems()
allItems= findAllItems()
item = findItem(name)
item= findItem(name)
«GaStep» addItemToCart(sessionId,item)
«GaStep» addItemToCart(customer,item)
«GaStep» persist(cart)
«GaStep» confirmOrder(sessionId)
«GaStep» confirmOrder(customer)
«GaStep» persist(order)
«GaStep» delete(cart)

Figure 5: The *PetStore Single Buy Scenario*.

- Average Queue Length at each service center

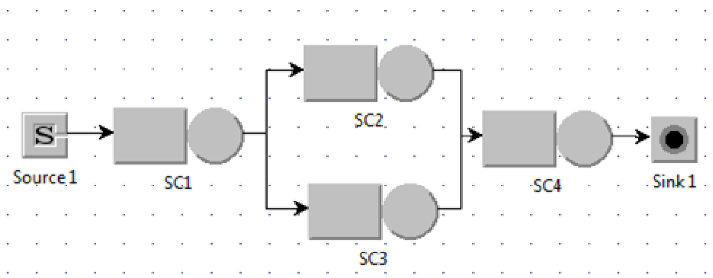
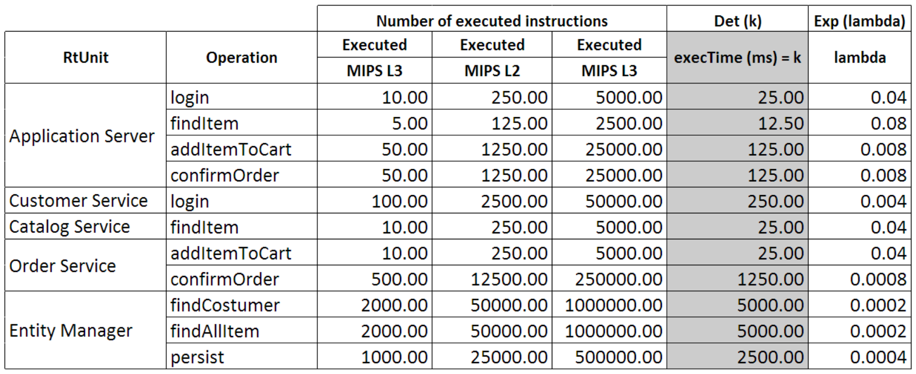


Figure 6: A QN specified in JSIM.

In order to set up the comparison, we first created, parametrized and execute with JSIMgraph a simple JSIMgraph model as shown in Figure 6. The QN is composed by a *source* and *sink* nodes and four service centers (*SCs*). Such a QN defines two different classes of customers (*class01* and *class02*) according with two distinct open workloads. The interarrival times among jobs, expressed in milliseconds. are generated by two distinct exponential probability distributions $exp(\lambda)$. Each job from both customer classes generates three service requests before leaving the QN from the same *sink* node. In particular, all the jobs generate a service request on SC1 and SC4, while SC2 and SC4 serves only jobs belonging to the *class01* and the *class02* classes, respectively. At each SC, the service time (in milliseconds) required to complete the processing of a

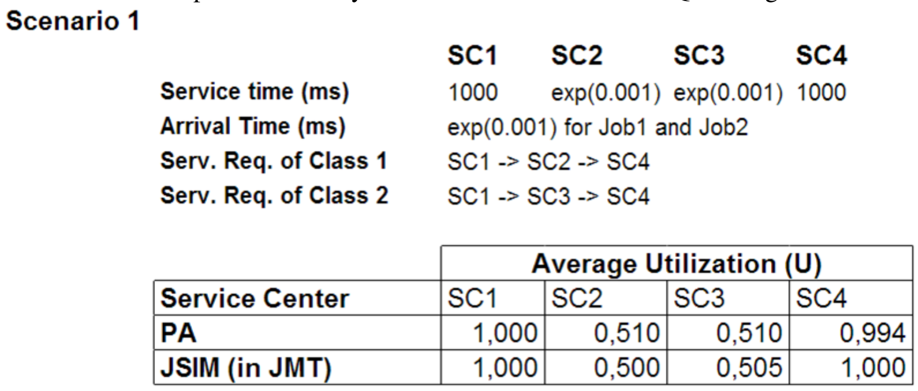
Table 2: Execution times for operations in the *PetStore Single Buy Scenario*.



		Number of executed instructions			Det (k)	Exp (lambda)
RtUnit	Operation	Executed MIPS L3	Executed MIPS L2	Executed MIPS L3	execTime (ms) = k	lambda
Application Server	login	10.00	250.00	5000.00	25.00	0.04
	findItem	5.00	125.00	2500.00	12.50	0.08
	addItemToCart	50.00	1250.00	25000.00	125.00	0.008
	confirmOrder	50.00	1250.00	25000.00	125.00	0.008
Customer Service	login	100.00	2500.00	50000.00	250.00	0.004
Catalog Service	findItem	10.00	250.00	5000.00	25.00	0.04
Order Service	addItemToCart	10.00	250.00	5000.00	25.00	0.04
	confirmOrder	500.00	12500.00	250000.00	1250.00	0.0008
Entity Manager	findCostumer	2000.00	50000.00	1000000.00	5000.00	0.0002
	findAllItem	2000.00	50000.00	1000000.00	5000.00	0.0002
	persist	1000.00	25000.00	500000.00	2500.00	0.0004

single job, is obtained by two probability distribution functions: $Deterministic(k)$ or $Exponential(\lambda)$. The former returns always k (that is, the service time is deterministic and always equal to k milliseconds) while the latter generates a positive random natural number (integer if $0 < \lambda < 1$), according with the exponential distribution.
We then run the simulation model in JSIMgraph and we obtain the results shown in Tables 3 and 4.

Table 3: Comparison of analysis results obtained from the QN in Figure 6.



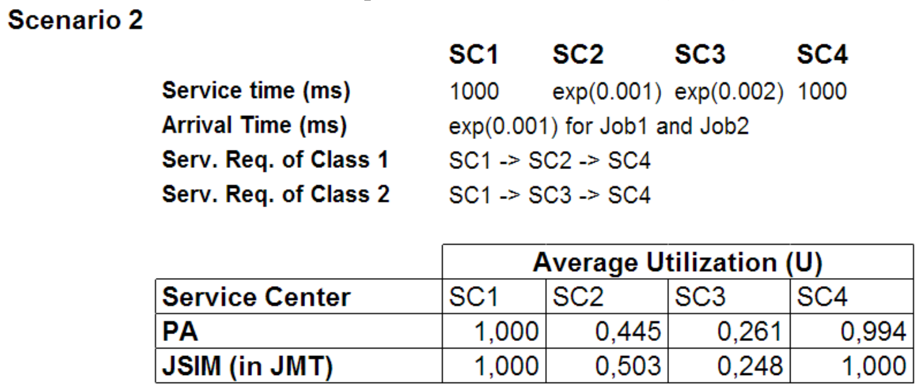
Scenario 1

	SC1	SC2	SC3	SC4
Service time (ms)	1000	exp(0.001)	exp(0.001)	1000
Arrival Time (ms)	exp(0.001) for Job1 and Job2			
Serv. Req. of Class 1	SC1 -> SC2 -> SC4			
Serv. Req. of Class 2	SC1 -> SC3 -> SC4			

	Average Utilization (U)			
Service Center	SC1	SC2	SC3	SC4
PA	1,000	0,510	0,510	0,994
JSIM (in JMT)	1,000	0,500	0,505	1,000

We then created the same model (i.e., the same SCs, the same connections and parameters) programmatically using the API provided by Performance Analyzer. Then we generated the random timed values (i.e., the interarrival times, service times etc) as they can be extracted from Timed Execution Traces of a corresponding fUML model. This step is repeated for several times and, in particular, for the number of samples required by JSIMgraph to compute its results in order to reach a comparable *confidence interval*. The results obtained from our PA are shown on the second row of Table 6(d). The comparison among the results from JSIM and PA, we performed a first successful

Table 4: Execution times for operations in the *PetStore Single Buy Scenario*.



Scenario 2

	SC1	SC2	SC3	SC4
Service time (ms)	1000	exp(0.001)	exp(0.002)	1000
Arrival Time (ms)	exp(0.001) for Job1 and Job2			
Serv. Req. of Class 1	SC1 -> SC2 -> SC4			
Serv. Req. of Class 2	SC1 -> SC3 -> SC4			

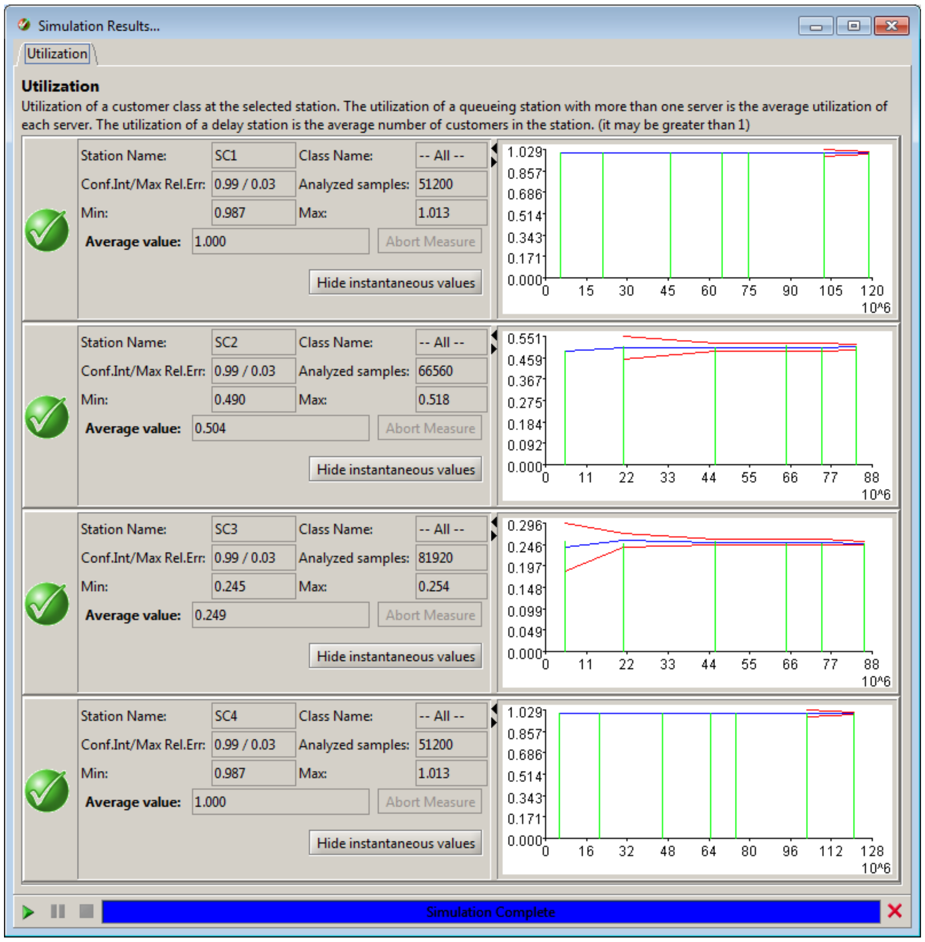
	Average Utilization (U)			
Service Center	SC1	SC2	SC3	SC4
PA	1,000	0,445	0,261	0,994
JSIM (in JMT)	1,000	0,503	0,248	1,000

test about the correctness of the first prototypical implementation of PA itself.
The variations among the results from PA and JSIM should be due to the different number of jobs created i.e. due to the number of samples to compute the average utilization (U) for each service center. As a future work, PA should be able to compute such average values with respect to a user-defined confidence interval (as available in JSIM).

References

[1] L. Berardinelli, P. Langer, and T. Mayerhofer. Combining fUML and Profiles for Non-Functional Analysis Based on Model Execution Traces. In *QoSA 2013*. ACM.

[2] C.U. Smith and L.G. Williams. *Performance solutions: A practical guide to creating responsive, scalable software*, volume 1. Addison-Wesley Boston, 2002.



Utilization

Utilization of a customer class at the selected station. The utilization of a queueing station with more than one server is the average utilization of each server. The utilization of a delay station is the average number of customers in the station. (it may be greater than 1)

Station Name:	SC1	Class Name:	-- All --
Conf.Int/Max Rel.Err:	0.99 / 0.03	Analyzed samples:	51200
Min:	0.987	Max:	1.013
Average value:	1.000		

Station Name:	SC2	Class Name:	-- All --
Conf.Int/Max Rel.Err:	0.99 / 0.03	Analyzed samples:	66560
Min:	0.490	Max:	0.518
Average value:	0.504		

Station Name:	SC3	Class Name:	-- All --
Conf.Int/Max Rel.Err:	0.99 / 0.03	Analyzed samples:	81920
Min:	0.245	Max:	0.254
Average value:	0.249		

Station Name:	SC4	Class Name:	-- All --
Conf.Int/Max Rel.Err:	0.99 / 0.03	Analyzed samples:	51200
Min:	0.987	Max:	1.013
Average value:	1.000		

Figure 7: A screen shot with the performance results from JSIMgraph.