A Metamodel for Generating Performance Models from UML Designs

Dorin B. Petriu, Murray Woodside

Dept. of Systems and Computer Engineering
Carleton University, Ottawa K1S 5B6, Canada
{dorin,cmw}@sce.carleton.ca

Abstract. Several different kinds of performance models can be generated from sets of scenarios that describe typical responses of a system, and their use of resources. The Core Scenario Model described here integrates the scenario and resource elements defined in a UML model with performance annotations, preparatory to generating performance models. It is based on, and aligned with the UML Profile for Schedulability, Performance and Time, and supports the generation of predictive performance models using queueing networks, layered queueing, or timed Petri nets. It is proposed to develop it as an intermediate language for all performance formalisms.

1. Performance Analysis of Software Specifications

Preliminary performance analysis can be effective in avoiding performance disasters in software projects [11]. However, it takes time and effort to derive the necessary performance models. The UML Profile for Schedulability, Performance and Time (SPT) [6] was developed to assist the capture of performance data, and automation of the model-building step. This should make the analysis more accessible to developers who are concerned about performance issues in their designs. Figure 1 illustrates the type of processing that is envisaged by the Profile.

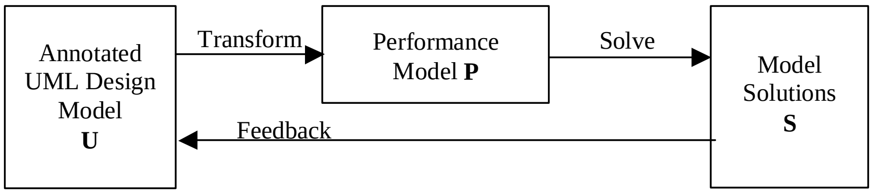

Fig. 1. Transformations and performance model solutions as envisaged in the UML SPT Profile

The range of applications covered by the SPT Profile is broad, ranging from embedded systems with schedulability concerns, to business systems. The present paper is directed to applications with probabilistic behaviour and statistical performance requirements, which are common in distributed information processing such as telecom, business systems and web services.

The relevant information in the UML design U is scattered in behaviour and deployment submodels, and possibly in other submodels. Some of it is expressed in the stereotypes and tag values of the SPT Profile, and some (e.g. the sequence of actions) is implicit in the UML. The Core Scenario Model (CSM) collects and organizes all this into a form that is convenient for generating P, and allows us to check for consistency and completeness of this information from the viewpoint of P. We thus propose a two-step processing sequence as shown in Fig. 2, with two transformations: U2C extracts the scenario model and C2P to derives a performance model, Different C2P transformations may support different performance formalisms for P.

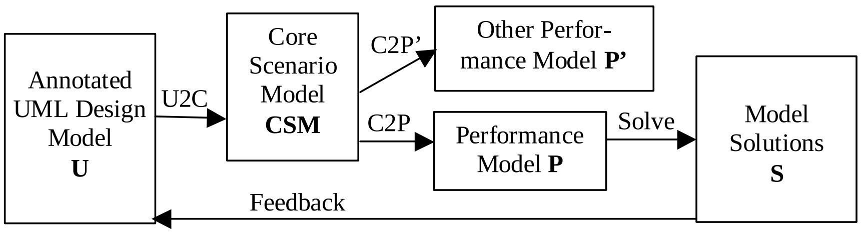

Fig. 2. Two-step transformation supporting consistency-checking and a variety of performance formalisms

The purpose of this work is to describe the CSM, demonstrate that it captures all the information defined by annotations in the profile, and discuss its feasibility for deriving performance models.

2. UML Profile for Schedulability, Performance, and Time (SPT)

The SPT Profile [6] extends the UML standard by defining stereotypes and tags which can be applied to object instances, and to instances of action executions in behaviour specifications. The UML specification together with the stereotypes determines structural properties of a performance model, and the tags provide parameter values. The profile is based on domain sub-models for resources and for performance, which are the basis of the CSM metamodel described below.

The SPT domain model for performance is summarized in Fig. 3. It is centered on a Scenario class, representing behaviour for one kind of system response. A scenario is an ordered sequence of steps, each of which can be a sub-scenario. The ordering supports forks, joins and loops in the flow. Stereotypes and tagged value names are prefixed by P or PA for "performance" or "performance analysis".

Each scenario has a "workload" which describes the intensity of its execution. It may be an open workload, with arrivals from the environment (described by their rate), or a closed workload in which a fixed number of potential arrivals are either in the system, or are waiting to arrive again.

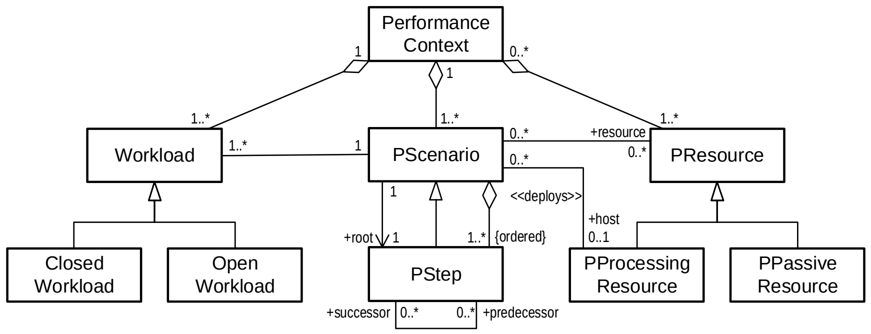

Fig. 3. The Performance domain model of the UML SPT Profile (from Fig. 8-1 of [6])

Resources may be attached to a scenario. The domain submodel for resources (in Chapter 4 of [6]) distinguishes between active resources (such as a user) which spontaneously generate events, and passive resources that respond to requests. Both types of resource may be protected, (in which case a client gets exclusive use of one or more units of the resource), or unprotected, in which case they can be shared without control. Chapter 8 distinguishes between processing resources (devices) and logical resources (created by the software, such as buffers, tasks, or semaphores). Every primitive Step has a host processing resource or CPU, which executes the step.

3. The Core Scenario Model (CSM) Metamodel

The CSM metamodel captures the essential entities in the domain model which are required for building performance models, and it makes explicit some facts which have to be inferred from the UML and the SPT Profile data. The class structure of CSM, consistent with the Meta-Object Facility (MOF, [7]) is shown in Fig. 4.

The CSM provides explicit representation of the scenario flow in a Path Connection type. The Profile depends on a simple successor association between Steps. Here, there is a PathConnection object between each pair of Steps, with subtypes which correspond to the sequential relationship types common in path models for real-time software: Sequence for simple sequence, one step after another; Branch for an OR-fork with Merge for an OR-join, to describe alternative paths, and Fork and Join for AND-fork and AND-join respectively (to describe parallel paths). Probabilities for a Branch are attributes of the target Steps. Explicit Path Connections (instead of just successor associations) simplify the later generation of a performance model, when the UML context is stripped away.

Each PathConnection subtype takes a different number m of source steps, and n of successor steps, and these are labeled with the subtypes in the diagram. For example, a Sequence has exactly one source and one target Step, while a Fork or Branch has one source Step and multiple target Steps.

Explicit subtypes of Step, for Resource Acquire and Release, and for Start and End of a Scenario, also support checking of the model, and performance model generation.

A Message class, which may be associated to any path connection, has been added for future use (it is not supported in [6]), to describe the size of network messages sent between system nodes.

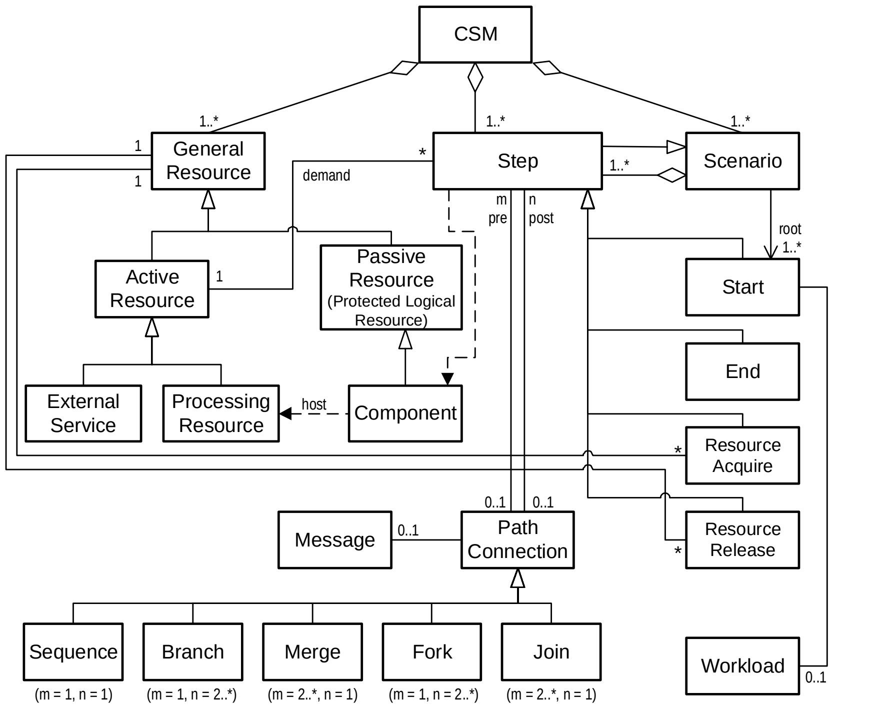

Fig. 4. Classes in the Core Scenario Model meta-model. Attributes are described in Table 1

Active and passive resources represent the resources defined in the Profile. Active resources include devices (Processing Resources) and subsystems (captured by a placeholder called External Service). The latter are service operations executed by some resource outside the design document. Passive resources include operating system processes (or threads) identified as Components, and hosted by Processing Resources. In this way a primitive Step (but not an abstract Step) has a host resource through its Component. Unprotected resources have been combined with protected resources, based on a multiplicity parameter which defines the number of units of the resource, such as a number of buffers, or of threads. An exclusively-held resource has multiplicity one, while an unprotected resource is indicated by an infinite multiplicity. This avoids the need for separate classes and is consistent with resource notation in queueing models. The attributes of the CSM correspond to tagged values in the

Profile and are described in Table 1. These include ID-ref attributes representing meta-associations which are not shown in Fig. 4 to avoid cluttering the diagram.

4. A Building Security System (BSS) Example

An example which has been used in previous work to describe the use of the SPT Profile [9][13], will be used here to describe the use of CSM. It is a Building Security System (BSS), which controls access and monitors activity in an institutional building. The BSS is deployed on a main processing node, a database node, and a set of other resources all communicating over a LAN, as shown in Fig 5.

Table 1. Attributes of the CSM Metaclasses (ID is a unique identifier generated automatically, opt stands for optional, and ref stands for an object reference)

CSM Class	**Attributes**
Component	ID; name; host ProcessingResource ID ref; 'is active' flag; description (opt); multiplicity (opt); containing component ID (opt)
ActiveResource	ID; name; time per operation; scheduling policy; description (opt)
Scenario	ID; name, collection of Steps
Step	ID; name; Component ref, host ProcessingResource demand, optional collection of pairs of ExternalService ID refs and demands; probability (opt); repetition count (opt); subscenario ref to nested Scenario (opt); description (opt); selection policy (opt)
Start	*Step attributes* + Workload ID ref
End	*Step attributes*
ResourceAcquire	*Step attributes* + Resource ID ref; resource units (opt); priority (opt)
ResourceRelease	*Step attributes* + Resource ID ref; resource units (opt)
Workload	ID; arrival stream type (open or closed); arrival process; distribution type; closed system population size (opt); mean inter-arrival delay (opt); lower bound on the inter-arrival delay (opt); upper bound on the inter-arrival delay (opt); inter-arrival process description (opt)
PathConnection	ID; Message ID ref (opt); condition (opt); label (opt)
Sequence	*Path Connection attributes* + source Step ref; target Step ref
Branch	*Path Connection attributes* + source Step ref; target Step refs
Merge	*Path Connection attributes* + source Step refs; target Step ref
Fork	*Path Connection attributes* + source Step ref; target Step refs
Join	*Path Connection attributes* + source Step refs (2 or more); target Step ref
Message	type (none, asynchronous, synchronous, reply); size; multiplicity (opt)

In Fig. 5, the nodes are either host ProcessingResources (<<`PAhost`>>) which execute the steps of the various components, or other physical ProcessingResources stereotyped as <<`PAresource`>>. The software components are concurrent processes (the stereotype <<`PAresource`>> has not been shown, but is shown in Fig. 6) and a buffer pool called Buffer (also <<`PAresource`>>), associated with the Buffer Manager. The size of the buffer pool is given by the tag PAcapacity, which takes the default value of 1 for the other resources.

A performance-sensitive scenario for video surveillance is presented as a UML Sequence Diagram in Fig. 6. Video frames are captured periodically from a number of web cameras located around the building, and stored in the database as needed. The example is explored in greater detail, including Activity Diagrams and an additional scenario for managing building access, in [13].

In Fig. 6 (and also Fig. 5) the annotations use the stereotypes and tagged values defined in the SPT Profile. A *performance context* (labeled <<`PAcontext`>>) defines the overall scenario made up of *steps* (<<`PAstep`>>). The first step is driven by a *workload* (<<`PAclosedLoad`>>). The step use resources, with a host resource (<<`PAhost`>>) for its processor. Each step is a focus of control for some concurrent component (<<`PAresource`>> in Fig 6). The stereotype can be applied to the focus of control or to the message that initiates it, and can be defined directly or in a note.

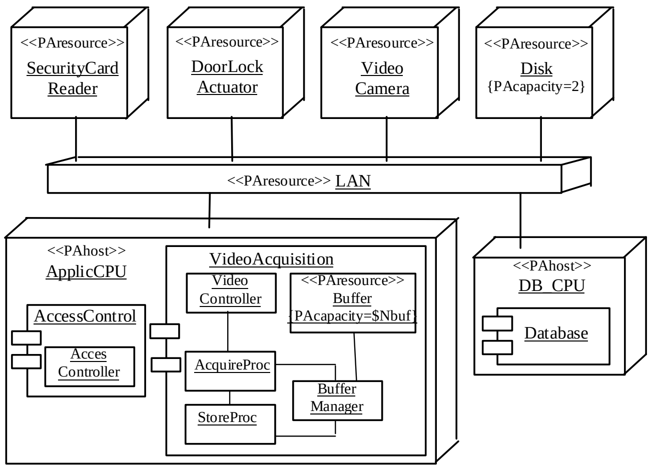

Fig. 5. Deployment and software components in the Building Security System (BSS)

The steps are tagged with a CPU demand value for processing time (tag `PAdemand`). The PAworkload stereotype is also tagged with a response time requirement, indicated by the tag PAInterval with parameter ‘req’, that the interval between successive frames be less than 1 second 95% of the time, and a placeholder with name $Cycle is defined for the model prediction for the 95th percentile.

The getImage step requires a network operation which is not included in the Sequence Diagram. It is described as a demand for an ExternalService (tagged as <<PAextOp>> on the getImage step), shown by the tag PAextOp = (Network, $P), indicating a number of network operations (and latencies) defined by the variable $P. The time for a network latency is not defined in the UML specification or the Profile, and is meant to be supplied by the modeler. If it is supplied during the U2C transformation, it can be included as an attribute of the corresponding ExternalService ActiveResource object.

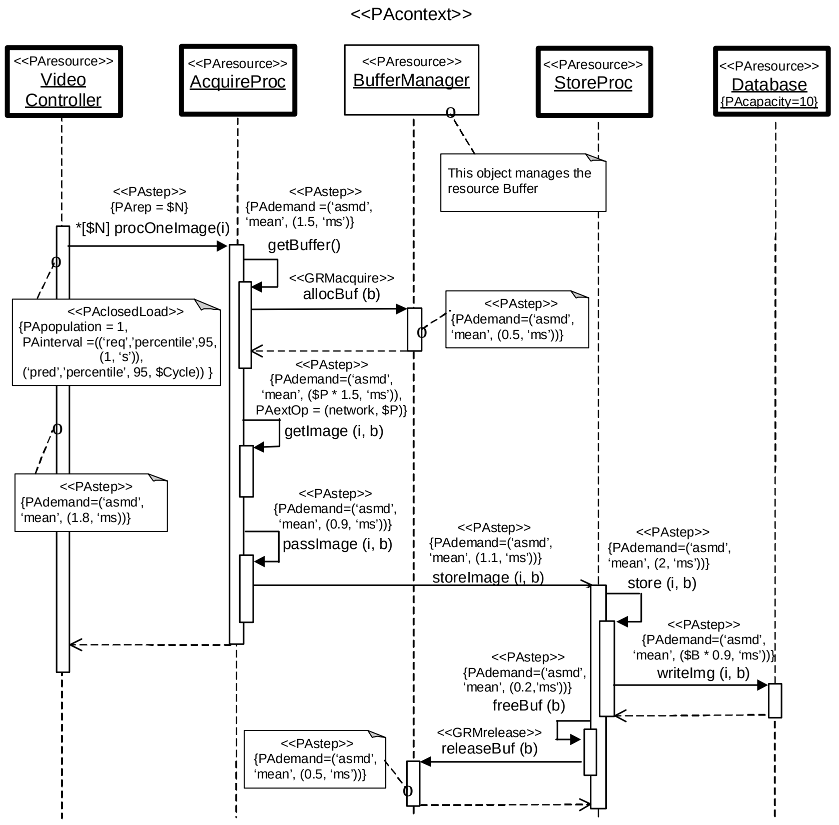
<<PAcontext>>
<<PAresource>> Video Controller
<<PAresource>> AcquireProc
<<PAresource>> BufferManager
<<PAresource>> StoreProc
<<PAresource>> Database {PAcapacity=10}
This object manages the resource Buffer
<<PAstep>> {PArep = $N}
*[$N] procOneImage(i)
<<PAstep>> {PAdemand =('asmd', 'mean', (1.5, 'ms')}
getBuffer()
<<GRMacquire>> allocBuf (b)
<<PAstep>> {PAdemand=('asmd', 'mean', (0.5, 'ms'))}
<<PAclosedLoad>> {PApopulation = 1, PAinterval =(('req','percentile',95, (1, 's')), ('pred','percentile', 95, $Cycle)) }
<<PAstep>> {PAdemand=('asmd', 'mean', ($P * 1.5, 'ms')), PAextOp = (network, $P)}
getImage (i, b)
<<PAstep>> {PAdemand=('asmd', 'mean', (1.8, 'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean', (0.9, 'ms'))}
passImage (i, b)
<<PAstep>> {PAdemand=('asmd', 'mean', (1.1, 'ms'))}
storeImage (i, b)
<<PAstep>> {PAdemand=('asmd', 'mean', (2, 'ms'))}
store (i, b)
<<PAstep>> {PAdemand=('asmd', 'mean', ($B * 0.9, 'ms'))}
writeImg (i, b)
<<PAstep>> {PAdemand=('asmd', 'mean', (0.2,'ms'))}
freeBuf (b)
<<GRMrelease>> releaseBuf (b)
<<PAstep>> {PAdemand=('asmd', 'mean', (0.5, 'ms'))}

Fig. 6. The Acquire/Store Video scenario for the Building Security System

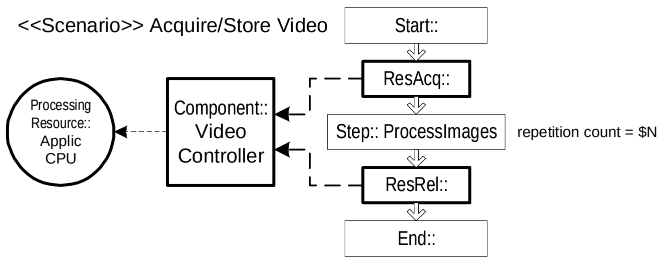
<<Scenario>> Acquire/Store Video
Start::
ResAcq::
Step:: ProcessImages
repetition count = $N
ResRel::
End::
Component:: Video Controller
Processing Resource:: Applic CPU

Fig. 7. The CSM representation of the information in Figures 5 and 6: the high-level loop

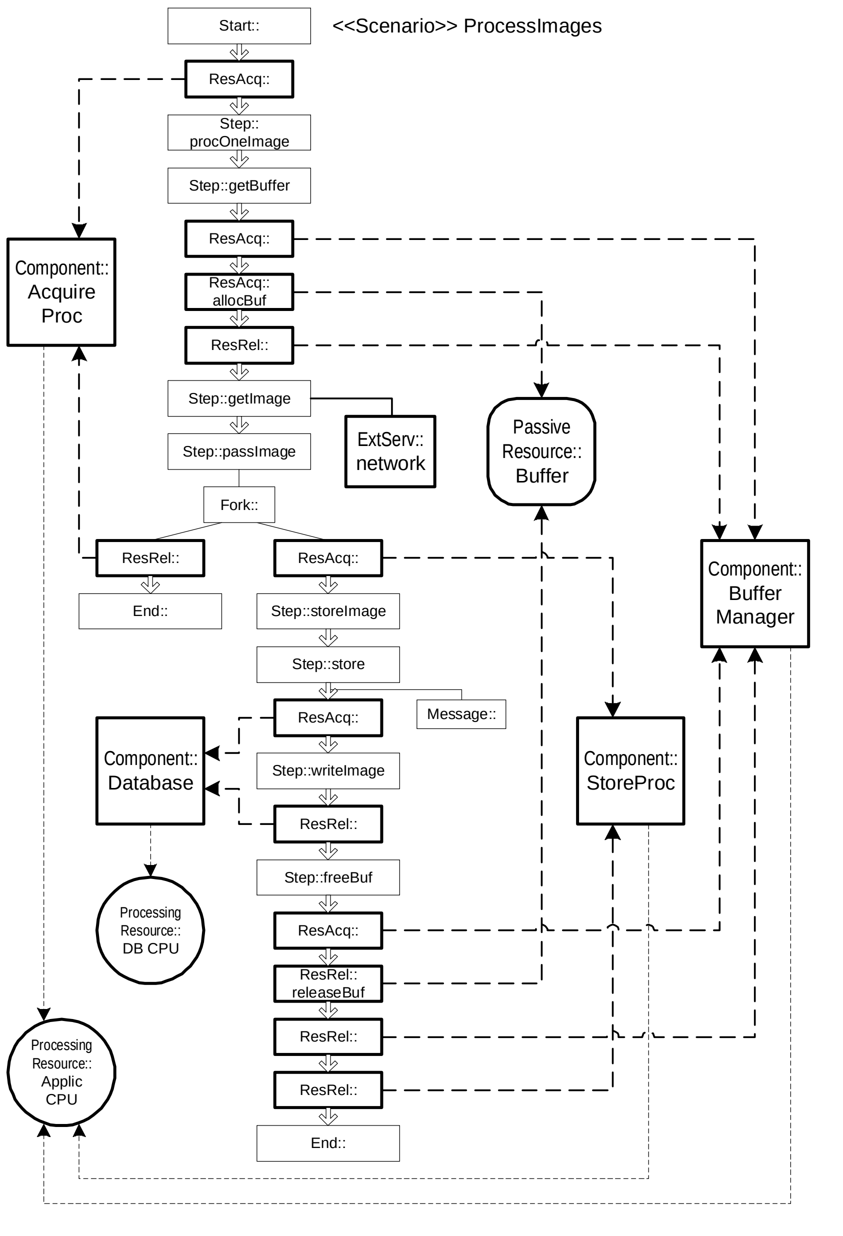

Fig. 8. The CSM representation of Figures 5 and 6: refinement of the loop step

Figs. 7 and 8 show the corresponding CSM model, using some abbreviations for CSM types (such as ResAcq for ResourceAcquire) which should be clear. The loop which is indicated by the parameter *$N on the procOneImage message in Fig. 6, has been realized as a CSM Step processImages with a repetition parameter $N in Fig. 7. It is refined by a subscenario also named processImages, in Fig. 8. Figs. 7 and 8 use some special graphical notation. They use bold outlines for Resources and for ResourceAcquire and ResourceRelease Steps, and bold arrows for dependencies between them. There is also a special hollow arrow representing a Sequence path connection and its associations with its predecessor and successor Steps.

It can be confirmed that Figs. 7 and 8 capture the sequential scenario information from the Sequence Diagram, and the resource relationships from it and the Deployment Diagram, without loss of information.

A place has also been provided in the CSM representation for a description of the Message sent to the database to log the video frame. The UML specification does not provide message parameters, but a modeler could provide them for the CSM.

The power of the CSM representation can be observed in its clear representation of the Buffer buffer pool resource, which is important in this system. In the UML specification its existence is defined in Fig. 5, and its use is indicated by the GRMAcquire and GRMrelease stereotypes on the messages to the Buffer Manager in Fig 6, and the note that associates the Buffer Manager with the Buffer resource. The U2C transformation assembles the information and connects the acquisition and release points to the resource. If the UML models failed to define the buffer pool resource, or to associate it correctly with the messages to allocate and deallocate buffers, the missing information could be discovered during the U2C transformation and supplied through queries to the modeler.

5. The Transformations to Generate a Performance Model

The implementation of the two transformation steps in Fig. 2 (U2C and C2P) can exploit a variety of established techniques.
U2C is a model transformation as envisaged in the proposals for Queries, Views and Transformations (QVT) at the OMG, such as [1]. As performance is platform-dependent, other QVT transformations to incorporate platform-specific elements may also be a part of U2C, or a preliminary to it. Work with QVT is for the future. At the time of writing we are creating a direct ad hoc transformation from an XMI representation of the UML design, to an XML representation of the CSM, following the CSM metamodel.

5.1. Transformation U2C, from UML to a Core Scenario Model

The transformation that is currently being developed reads XMI files representing the UML design model, builds a data structure to represent the UML objects, and uses it to create a DOM (Domain Object Model) tree for the CSM, which can be output in XML. The formal definition of this U2C transformation, based on the UML metamodel, will be described in a planned report on the transformation. Here, the status of the transformation and its general approach will be described. At present, only Deployment Diagrams and Activity Diagrams are included; Sequence Diagrams will be approached later. The Activity Diagram for the example (corresponding to Figure 6) is given in [9].

The first step is to create resource and component objects for the resources and components in the UML description, using the stereotypes (possibly provided by notes) in the UML, and the one-to-one correspondences shown in Table 2. Attributes are obtained from tagged values. Then each Activity Diagram (AD) in the UML is examined. A CSM Scenario is established, and the AD Partitions (swimlanes) are identified with Components in CSM, constructed from the Deployment information. If this is not straightforward through component names, the user is asked about creating Component objects for them. From the Initial PseudoState, a Start Step is created. If a State following this has a <<PAworkload>> stereotype, the workload parameters become attributes of the Start Step. The rest of the AD is used to create a set of Steps and Connectors from the correspondences in the second part of Table 2. If an activity is of type CompositeState, with another diagram to refine it, the Step in CSM has a reference to a nested Scenario created for the second diagram.

Table 2. Objects in CSM, created in direct correspondence to objects in a UML Deployment or Activity Diagram

Type of Object in UML	**Stereotype in STP Profile**	**Type of Object in CSM**
Deployment Diagram		
Node	PAresource	Passive Resource
Node	PAhost	Processing Resource
Component	PAresource	Passive Resource
Component	none	Component
Activity Diagram		
SimpleState	PAstep	Step
CompositeState	PAstep	Step with nested Scenario
PseudoState: Initial	none	Start Step
PseudoState: Fork	none	Fork PathConnector
PseudoState: Join	none	Join PathConnector
PseudoState: Branch	none	Branch PathConnector
PseudoState: Merge	none	Merge PathConnector
FinalState	none	End Step

The sequential relationships in the Scenario are constructed by linking the Steps through IDrefs to PathConnectors, as described by the attributes shown in Table 1. They are based on the Transitions in the Activity Diagram, and their source and target States. If both are States (SimpleState or CompositeState) the Transition provides a Sequence PathConnector linked to the corresponding Steps. If one is a Fork, Join, Branch or Merge PseudoState, the Transition provides only a pair of links (in both directions) between the corresponding PathConnector and the Step for the other. If both are PseudoStates then a dummy Step is introduced between the PathConnectors in the CSM.

A loop can be described in the Activity Diagram by a CompositeState activity (referencing another diagram for a subscenario), stereotyped as <<PAstep>> with a repetition count tagged value. This can be translated directly to the CSM representation as a high-level Step with a nested Scenario and a repetition count attribute. However another possible representation of a loop in an AD is a sequence of activities delimited by a Merge at the loop head (where the repetition begins) and a Branch at the loop end (with a Transition back to the Merge for the next repetition); we can call this an “informal loop”. Informal loops must be detected and then transformed to the subscenario structure just described, but this has not yet been addressed.

In a diagram with swimlanes representing different Components, a Transition from one swimlane to another implies releasing one Component Resource and acquiring the other. A Resource Release Step, Sequential PathConnector and Resource Acquire Step are introduced into the sequence for that Transition, possibly with Sequential PathConnectors at the beginning and end, depending on the source and target.

External Operations (with operation counts) are created in CSM for every ExternalService in the Activity Diagram, named in a tagged value attached to a <<PAStep>> stereotype. They are placeholders for calls to services defined outside the design, and are intended to be described by submodels introduced at the CSM level or at the performance model level (as in [12] for instance).

Since most of the CSM metamodel corresponds one-to-one to elements in the domain model underlying the SPT Profile, most of the U2C transformation is straightforward. Most of the transformations described above have been implemented and tested at this point; some details are incomplete.

Clearly some parameters or resource specifications may be missing, and the process of CSM extraction will include reports to the user, and provision of default values. Some translations have not yet been addressed, including informal loops (mentioned above) and the “Message” objects in CSM (which are not derived from the current Profile). Our current approach to any gap or ambiguity is to ask the tool user to resolve it.

Traceability from objects in the UML design to objects in the CSM (and on into the performance model) is presently provided only by the use of names, such as the names of Resources, Components, and Steps. Some UML tools provide a unique object identifier which could be carried into the CSM along with the name, to provide a stronger traceability.

5.2. Transformation to a Performance Model

There is not space here to describe the possible C2P transformations in any detail, but typical performance models use Queuing Networks, Layered Queues, or timed Petri nets. For systems with only processing devices and no nested resources Smith has shown how to create a Queueing Network model from a similar scenario model which she calls an Execution Graph [11] (she also deals with nested resources on an ad hoc basis). For nested resources, a systematic algorithm which traverses a scenario model almost identical to CSM, and generates a layered queueing (LQN) performance model [3] was described in [10]. The LQN that emerges for the BSS example in this paper was described in [13], with an analysis that illustrates the use of the model to study bottlenecks and scalability.

Other authors have created performance models using different intermediate models. Shen and Petriu extracted a graph model of the scenario from UML activity diagrams and used graph transformations to create an LQN [8]. Kahkipuro [4] defined his own intermediate model expressing the UML information he needed to create a model similar to a LQN. Cortelessa et al define a kind of Execution Graph to extract the scenario [2], and derived queueing models. Lopez-Grao et al [5] derive a timed Petri net in an interesting way. They create subnets for the individual Steps with labels that allowed them to compose the fragments of scenarios, bottom-up, to arrive at a model for the entire behaviour. However, their approach is suitable only for the particular style of Petri nets they use, and at this point their model does not address processor contention.

6. Conclusions

The Core Scenario Model defined here has the capability to open up the use of predictive models for performance of software designs, by providing a core of information needed in building performance models of different kinds. It provides a kind of "Unified Performance Model", at one remove from the actual modeling formalisms.

This paper described the model and showed, using an example, how it captures the required information. CSM provides a bridge between UML and the SPT Profile, and existing techniques to generate performance models which are based on the queueing and layered queueing formalisms. While it is not demonstrated here, it seems clear that Petri net models may be obtained with equal ease, and that new features introduced in UML 2 are easily accommodated.

Acknowledgments
This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its program of Strategic Grants. The U2C transformations outlined here are the work of Kathleen Shen and will be described in detail elsewhere.

References

[1] DSTC et al, *MOF Query/Views/Transformations: Second Revised Submission*, OMG document ad/04-01-06, Jan. 2004.

[2] V. Cortellesa, "Deriving a Queueing Network Based Performance Model from UML Diagrams," *in Proc. Second Int. Workshop on Software and Performance (WOSP2000)*, Ottawa, Canada, 2000, pp. 58-70

[3] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, "Performance Analysis of Distributed Server Systems," *Proc. 6th Int. Conf. on Software Quality (6ICSQ)*, Ottawa, Ontario, 1996, pp. 15-26.

[4] P. Kahkipuro, "UML-Based Performance Modeling Framework for Component-Based Systems," in *Performance Engineering*, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds. Berlin: Springer, 2001.

[5] J. P. Lo'pez-Grao, J. Merseguer, and J. Campos, "From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering," in *Fourth Int. Workshop on Software and Performance (WOSP 2004)*, Redwood City, CA, Jan. 2004, pp. 25-36.

[6] Object Management Group, "UML Profile for Schedulability, Performance, and Time Specification," OMG Adopted Specification ptc/02-03-02, July 1, 2002

[7] Object Management Group, *Meta Object Facility (MOF) 2.0 Core Specification*, OMG Adopted Specification ptc/03-10-04, Oct. 2003.

[8] D. C. Petriu and H. Shen, "Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specifications," in *Proc. 12th Int. Conf. on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation*, London, England, 2002.

[9] D. C. Petriu and C. M. Woodside, "Performance Analysis with UML," in *"UML for Real"*, ed. B. Selic, L. Lavagno, and G. Martin, Kluwer, 2003, pp. 221-240.

[10] Dorin Petriu, Murray Woodside, "Software Performance Models from System Scenarios in Use Case Maps", *Proc. 12 Int. Conf. on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation (Performance TOOLS 2002)*, London, April 2002.

[11] C. U. Smith and L. G. Williams, *Performance Solutions.* Addison-Wesley, 2002.

[12] Xiuping Wu and Murray Woodside, "Performance Modeling from Software Components," *in Proc. 4th Int. Workshop on Software and Performance (WOSP 2004)*, Redwood Shores, Calif., Jan 2004, pp. 290-301

[13] Jing Xu, Murray Woodside, Dorina Petriu "Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time", *Proc. 13th Int. Conf. on Computer Performance Evaluation, Modeling Techniques and Tools (TOOLS 2003)*, Urbana, Illinois, USA, Sept 2003, pp 291 – 310.