Report SCE-01-09, Dept. of Systems and Computer Engineering, Carleton University, Ottawa Canada, Sept 25, 2001.

***Copyright C.M. Woodside*.**

Performance-related Completions for Software Specifications

Murray Woodside, Dorin Petriu, Khalid Siddiqui
Carleton University
Ottawa, Canada K1S 5B6
1-613-520-5721

{cmw | dorin | khs}@sce.carleton.ca

Sept 25 2001

ABSTRACT

To evaluate a software specification for its performance potential, it is necessary to supply additional information, not required for functional specification. Examples range from the execution cost of operations and details of deployment, up to missing subsystems and layers. The term "completions" is used here to include all such additions, including annotations, component insertions, environment infrastructure, deployment, communication patterns, design refinements and scenario or design transformations which correspond to a given deployment style. Completions are related to the purpose of evaluation, so they are tailored to describing the performance at a suitable level of detail. Completions for evaluating other attributes such as reliability or security are also possible. The paper describes how completions are added to a specification regardless of the language used (provided that it describes the system behaviour as well as its structure), and experience with completions in Use Case Maps.

General Terms

Performance, Design, Reliability, Documentation.

Keywords

Performance prediction, software specification, generative design.

1. INTRODUCTION

Developers may wish to evaluate non-functional properties of software specifications, in order to make satisfactory choices in systems which are increasingly complex. Properties such as performance, dependability and security may be evaluated at many different levels of specification, including Use Cases, execution scenarios, formal specifications, architecture, design (including executable behaviour specified by state machines), and prototypes. For any such specification, this work considers evaluation for performance; further we focus on quantitative evaluation by building a model, which is a common approach. The main ideas apply equally to evaluation for other properties.

There are many specification techniques which are performance-enabled to some degree. Scenarios were used in the pioneering work of Smith [34], and by many others, including the authors' work with Use Case Maps [37]. Formal specifications by Petri Nets and Process Algebras have been extended to Timed Petri Nets and Stochastic Process Algebras [14]. Executable design tools based on SDL [36] and the UML [5] have had annotations added for performance properties. Woodside et al [20, 31] made models from specifications in ROOM, similar to Real-time UML. Prototyping approaches with performance analysis were described by Menasce and Gomaa in [21], and by Bagrodia in [1]. Architecture evaluation has been described by Bass et al [3], making use of scenarios to explore any property of interest; an example with quantitative performance analysis was described by Nord [27]. Another approach to architecture performance evaluation was described by Balsamo et al in [2].

Existing performance–enabled specification languages provide

- descriptions of the deployment of the software on the hardware, and
- annotations for the CPU demands of operations, for the size of messages and databases, and for the latencies and bandwidth of communications.
- Annotations to describe the intensity of the workload (arrival rate, user population, event intervals)

Examples include proposals for considering time and performance in the UML (e.g. [19], [30]), in Process Algebras and Petri Nets (see [6]), modifications to SDL to support performance analysis (see [22] for examples), and annotations to Use Case Maps (which are used for developing concurrent architectures) [37].

However these annotated specifications are often insufficient for performance evaluation, because of their level of abstraction. If we compare a satisfactory high-level functional specification to the final deployed version whose performance we wish to evaluate, we can see four categories of elements in the final system, of which only one is fully defined in the specification:

- *specified elements* which are fully described in the specification, usually being the system elements under development,
- *implied elements*, which can be inferred from the description of the specified elements,
- *re-used elements* which are existing software components, represented in the specification by design stubs, (meaning, abstractions which just show their interfaces). Demand annotations to the stubs ignore the actual structure of these elements, which may be subsystems with many components and be distributed over many nodes.
- *system service layers* which provide communications, data access and system management, which aren't shown explicitly, but can be inferred.

The gap comes from missing elements that cannot be described by simple annotations. (By "elements" we mean the entities defined by the specification language, such as components, processes, roles, ports, gates, connectors, etc.) Some languages do have facilities, such as stereotypes in the UML, which can in principle be used to define these missing elements.

Performance evaluation needs to include the workload and delays for the implied, re-used and service layer elements. Deployment and demand annotations on the specified and stub elements in the specification (which is the present state of the art) may not be sufficient. The gap is the possibly complex and distributed structure of the missing elements, which may be critical to the evaluation. The width of the gap depends on the nature of the system (will it use such complex elements?) and on the depth of detail needed in the evaluation. For example some existing works model communications by a known overhead cost and a latency. However when CORBA is used to connect application components [23], a single message at the specification level may imply references to a remote directory, and multiple interactions with the ORB, as well as the messages between the components.

This work defines *completions* as a general means to capture performance concerns, to modify the specification and to describe all of the expected elements. Completions include the existing ideas of annotations, but also:

- self-contained fully-defined components or subsystems that replace stubs and that provide system services,
- self-contained subsystems that are implied by the specification (the second category above), according to interpretations or policies for interpretation, or to suitable annotations,
- patterns which transform the specification
- deployment, demand annotations and overhead allowances, as used in previous work.

Because they are for performance evaluation, they may be approximate in both functional and structural terms, or they may define workload and demand abstractions. In this way they are different from functional refinements of the specification, elaborations by generative programming [11], or software components in component-based software engineering [35].

A possible tool structure to build models with completions is suggested in Figure 1. There are completions that transform the specification, and others that are indicated within the specification but are incorporated only when the model is built. User input is provided to resolve any parameter values or decisions which are undefined when the model-building begins.

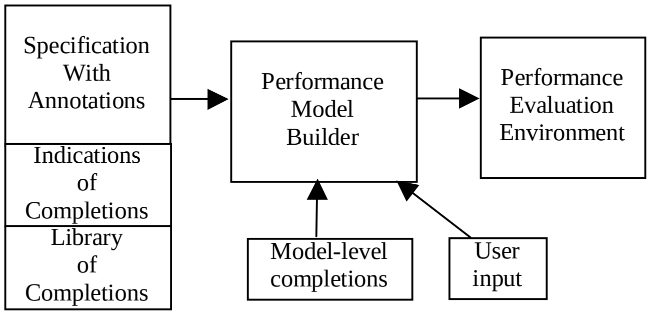
Specification With Annotations
Indications of Completions
Library of Completions
Performance Model Builder
Performance Evaluation Environment
Model-level completions
User input

Figure 1 Suggested tool architecture for Evaluation

Of existing specification languages only the UML profile proposed in [30], and Use Case Maps [37] go beyond simple demand and workload annotations, and even they are limited to stub replacement and service indications.

The purpose of this work is to investigate the general idea of completions and how performance completions can be introduced into specifications. This work only considers the capability to build a model; the accuracy and usefulness of the model will depend on the system, the goals of evaluation, the modeling technique and the quality of data available.

The potential uses of performance completions include verifying performance requirements, comparing design alternatives and evaluating flexible systems targeted to a range of environments.

2. EVALUATION OF SPECIFICATIONS.

To carry out an evaluation, we assume that we have

1. a specification language which accommodates performance annotations, as already discussed,
2. a procedure for building a performance model from the specification, as in Figure 1.

Figure 2 shows an example specification which illustrates different kinds of completions. It specifies a scenario for the use of a video server that is accessed through the Web. This is a Use Case Map (UCM) [9], [8] which shows a scenario as a line beginning at a filled circle and ending at a bar. Responsibilities along the path are indicated by X, and alternative paths by a Y-shaped branching out or joining together. Parallel paths fork out from a bar placed across the path, and may join at another similar bar. Components are indicated by boxes and parallelograms traversed by the path. Use Case Map specifications are suitable for presenting the ideas because they are highly visual and describe a high level of abstraction.

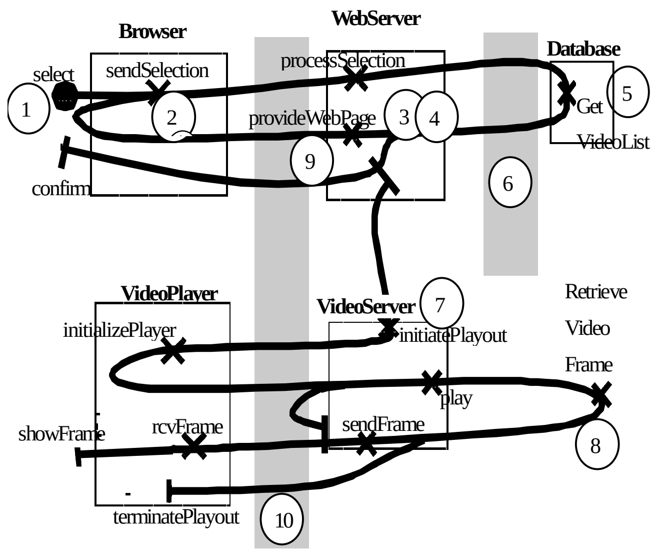

Figure 2. A Use Case Map specification for a video server, indicating execution paths, components, and some completions

In the scenario, a user accesses the video server to request a video (including a step to search one or more catalogues), and then the server connects to a video player component associated with the browser, and transmits the video. The performance concerns are

- the response times for the search and for confirming the request, and
- the jitter of the playback.

We consider the viewpoint of a designer of the video server software. A version of this example is used to explain the proposed UML Profile, with examples of UML annotations [30].

2.1 Performance Evaluation

Use Case Maps have well-defined performance annotations for the execution demand of responsibilities, services used by responsibilities (such as the database operations from the Web Server), loop counts, and the numbers of users initiating scenarios [12]. From an annotated UCM a layered queueing model can be generated automatically for performance prediction [24].

The important gap in this chain of operations is that only execution demands and delays that are attached to the UCM specification will enter the evaluation. Parameters cannot be attached to system components that are not included in the specification. Performance in this system is affected by elements that are only implied in Figure 2, such as the database.

2.2 Completions

The video server performance would be affected by all the following factors, which could be supplied as completions:

1. the users: how many at once, and how often they make a request,
2. the browser: its execution delays,
3. the web server execution demands in handling the request and sending the response,
4. the execution demands of the CGI script that creates the web page,
5. the database operations to get the list of videos to show the user,
6. the middleware that connects the web server to the database, such as (perhaps) a CORBA layer,
7. the video player software initialization execution at the users terminal,
8. the video storage subsystem that retrieves the frames of the video, possibly with special parallel structure in the storage,
9. the internet protocol software layers,
10. the assumptions about internet delays and their variability, to be used in the analysis.

The placement of these completions is indicated approximately in the diagram of Figure 2 by the numbered circles. A more precise notation to indicate that a completion is to be created is discussed in Section 4.

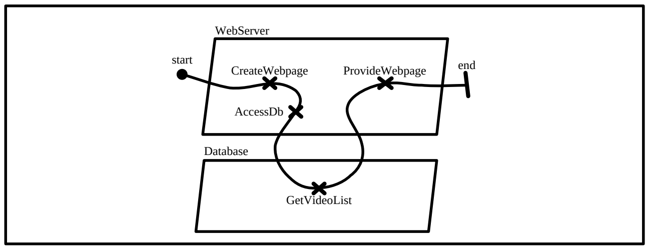

Figure 3a. The web server access to a database (detail)

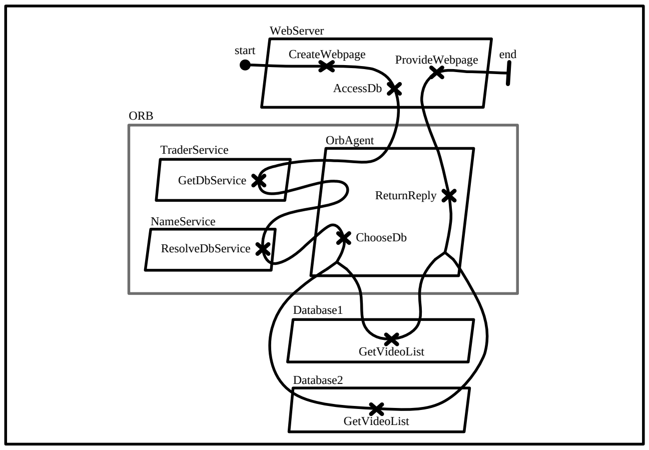

Figure 3b. Completions for middleware and database

Some of this information could be added as annotations to the design (UML annotations for this example are described in [30]), but others cannot. An example of a high-level completion which is too complex to be described by a simple annotation is indicated in Figure 3. Figure 3a shows just part of Figure 2, where the web server accesses a database. To complete the description of the access, Figure 3b adds:

- first, for circled item number 6 in Figure 2, a representation of a CORBA layer connecting the web server to the database (an ORB agent and services, shown as processes in the box in the middle), which discovers the appropriate database and its network address, and directs the request to the database
- also, for the database access indicated by circled item number 5 in Figure 2, it defines two alternative databases Database1 and Database2 with videolists, such that the ORB agent can resolve the request to go to one of them.

Further completions to show the internet delays, the structure of the database software and processors, and its storage devices and their organization, could be appropriate. Thus we can see that completions can be recursive (can require or imply further completions), and that the process of adding them can stop at a point chosen by the designer.

2.3 Evaluation Techniques

The performance model created from the specification and the completions may be based on simulation, queueing analysis, Markov Chains or other approaches. Some examples are surveyed by Dumke et al. in [14]. The present authors have used layered queueing, which is appropriate for systems with stochastic behaviour and service layers, and scales up well to large systems. An application of layered queueing to e-commerce servers is described in [13], and its use to analyse scalability of an IP telephony system is described in [17].

For time-critical systems with deadlines, schedulability analysis (e.g. [27]) can verify that responses can always be completed on time.

3. COMPLETENESS OF SPECIFICATIONS

The example in the previous section demonstrates that incompleteness can be a problem; how can we measure it? How can we say that a specification is sufficiently complete? *Performance Completeness* of a specification implies that it contains enough information to evaluate the performance of the system, to the desired accuracy. It must include sufficient information on:

- deployment data and workload intensity parameters
- system structure and behaviour,
- demand values by system elements for execution (CPU demand) and for services (from other elements, or from the system, such as IO demands),
- latency values

Information can be missing if it is not essential for meeting the goals of the evaluation.

A limited form of analysis called "bottleneck analysis" requires only the total demand for every device; the bottleneck device has the highest demand and limits the system capacity. Schedulability analysis which can guarantee that a system meets certain deadlines, on the other hand, requires precise information about everything.

The approach to a definition taken above has a certain circularity, in that one can be satisfied about completeness only after carrying out the analysis. More practical measures of levels of performance completeness can be defined, analogous to levels of coverage achieved in testing:

Levels of Performance Completeness (definition)

- *total completeness*, means the specification describes all activities of the system and its environment that are important to the performance goals
- *scenario completeness*, equivalent to total completeness but for a defined set of scenarios only,
- *time granularity completeness* which includes all operations with time demands greater than a defined granularity DT,
- *software/hardware granularity completeness*, in which some objects or subsystems are aggregated together.

4. INDICATING COMPLETIONS IN A SPECIFICATION

Completions need to be indicated for a specification, either within the document or as an attached commentary. Many languages have annotations or stereotypes which can be attached to specification elements (for instance, the tagged values and stereotypes in the UML [5]). These annotations which indicate completions add parameters of different kinds to the specification. We will define the completions in a way which is independent of any particular specification language, as a string defining a tuple which in turn indicates the completion. These tuples could be placed either in annotations within the specification, or in an ancillary document.

4.1 Replacement

A simple case is a completion which *replaces* a "stub" element in the specification by a subsystem (such as putting in the components and behaviour of a particular database in place of a "Database" stub in Figure 2 or 3a). The UCM language defines a "stub" element, but the term here refers to a design placeholder in any language. The replacement might be indicated by a tuple:

```
Replacement = ("substitution";
        element to be replaced;
        completion type:name:instance;
        parameter vector of elements type:name:value)
```

Elements to be replaced could include both components such as objects and processes, and connectors indicating communications, messages, calls or flows. The completion type and name are significant within an assigned library of completions, and the instance defines a name at the specification level for the instantiation of the completion. The type establishes the possible associations (roles or ports) of the completion (which must be compatible with the stub in the specification) and the parameters (defined by type:name) that can be assigned. Some generic types for completions are:

- a *service*, which replaces a stub which provides an operation through a call-return interface. Several services may be provided by the one server, so the server may be a parameter (and another completion). Web servers, database servers, file servers and directory servers are common examples of server completions and services include queries, updates, and transactions.

- a *layer*, which interposes between two components and modifies a service. The ORB in Figure 2 has this role,
- a *two-port filter*, which replaces a connection in the specification. A filter processes data and passes it on. Communications channels and real-time interfaces are common examples of filter completions.
- a *multicast filter* in which one send is propagated to many receivers.

The process of inserting a completion into a specification is essentially the same as binding a component into a design using a system such as RESOLVE [35].

In Figure 3b the database design stub of Figure 2 is replaced by a pair of databases and a choice between them. This could be defined by the following tuples:

```
(“substitution”; Database; subsystem:MultiDB:DB(1:2); -)
(“substitution”; DB(1); process:Oracle-DBS:Database1;
        int:records:25000)
(“substitution”; DB(2); process:MSQL-DBS:Database2;
        int:records:5500)
```

The first tuple introduces the choice and two placeholders DB(1) and DB(2). The second and third define the completions for the placeholders, and give the sizes of the databases. The tuples presumably do not have parameters for the execution workload of an Oracle database system, or an MSQL system, because those can be part of the completion itself.

4.2 Rules

Direct replacement can be implied rather than explicitly indicated, using rules which identify both the elements to be replaced, and the completion and its parameters. An example might be to replace all web servers in the specification by an Apache web server. Because a rule may cause multiple replacements it is not suitable to give a single instance name, so a group name is defined for all the instances, with a symbol # to be replaced by a number 1, 2, …

Replacement Rule: (“substitution-by-rule”;

> rule name as a string;
>
> {set of properties of set E of elements to be replaced as a comma-separated list of strings};
>
> completion type:name:instance#;
>
> parameter vector of elements type:name:value)

The properties referred to in the rule could be defined in special property tags or comments attached to specification elements, or (conceivably) they could be inferred by reasoning on the roles and context of the elements in the specification. In the video server, such a rule might be expressed as:

```
(“substitution-by-rule”; webServerRule; “ANY web server”;
        process:ApacheServer:ApacheWebServer#; -)
```

The web server element in Figure 2 would have to have a “web server” property. In this example the execution parameters of the web server are not specified in the tuple; they may be specified internally in the completion in some way.

4.3 Styles

A set of rules like this could represent the normal practices of a designer or design group, and such a set will be called a *completion style*. It consists of either a set of rules, or a set of groups of rules, or both:

Style: (“style”;

> style name (a string);
>
> {set of Replacement Rules, a comma-separated list of rule names};
>
> {set of Substyles, a comma-separated list of substyle names})

Styles can be hierarchical, with substyles to govern subsystems or aspects of the system, with rules that apply under some condition. A substyle can be defined by:

Substyle: (“substyle”;

> substyle name (a string);
>
> condition for application of the substyle;
>
> {set of Rules, a comma-separated list of rule names})

In the context of the example of Figure 1 and 2, a style might include rules for using Apache for any web server, sockets with TCP/IP for communications, and Linux NFS for any file server. A more sweeping style could be to always include an audit process in a system, which examines system state and initiates recovery in certain error conditions. The audit process does have to be specified, but its use could be indicated as part of an overall style for the system.

4.4 Communications Completions

Communications gives rise to slightly different completion rules because it links entities together, and is attached to connectors rather than to entities themselves. In general, a completion for communications may also involve more sophisticated additions to the specification, such as:

- scenario modifications to represent the protocol, for instance with stages for connection, directory search and authentication, data transfer,
- protocol operations at the sender and receiver
- other components and their operations, such as directory service,
- network contention delays
- latencies across network links

It may be bound into the system to serve a particular pair of processes, or several processes, or all the messages in the system, with multiple roles as senders and receivers. The elements to be added to the specification have to be configured according to these choices and to the deployment of processes. Protocol operations may be combined with the other application demands of application processes, or modeled separately with a communications front-end. In these cases the rules for configuring the completion are part of its definition, and they may depend on some of its parameters. For a two-party communications substrate a completion can be identified by the tuple:

Two-Party Communications Rule: (“comm2”;

rule name (a string);

initiator role element; responder role element;

completion type:name:instance;

parameter vector of elements type:name:value)

The completion instance may include elements shared with other communication paths. The parameters may govern the configuration and deployment of the subsystem, and its capacity and demand values.

A set of two-party paths by socket communication over a LAN provides an example of a completion to handle a simple message delivery. Each path has a specification of its completion with its own initiator and responder roles, but they all share the same completion instance. This completion will have to accommodate incrementally added usages when it is inserted into the specification. For each connection there is a message size description and protocol overhead, and for the entire LAN there is a speed parameter (such as 10 Mbit, 100 Mbit, or 1 Gbit per second), and (perhaps) a level of additional contending traffic.

In Figures 1 and 2, completions for internet communications between the user’s browser and video player, and the service site including the web server and video server, could be indicated by a tuple:

(“comm2”; “CommByTCPSocket”; “ANY user-node-element”;
“ANY server-node-element”;
network:TCPconnection;internet;
real:accessSpeedMegabits:100)

This identifies processes with the properties of being a “user-node-element” and a “server-node-element” and links them by a path with a TCP protocol stack at each end of each connection, suitable access interface, and a 100 Mbit access rate. It implies acknowledgements for messages, and waiting for TCP window flow control.

The CORBA layer shown in Figure 3b is an example of a more complicated subsystem which provides a service linking two entities, and replaces a connection in the specification. The layer could be quite complex, and (for example) provide transaction semantics and guaranteed delivery. The completion might be indicated by:

(“comm2”; “CommByCORBA”; “element WebServer”;
“element Database”; layer:CORBA-Orbix:ORB; -)

This identifies particular elements for the participants. The ORB completion has the structure shown in Figure 2b, and connects to whatever replaces the Database.

A more general multiparty communications completion requires a rule that identifies instances of elements by their roles in the completion. Examples include sending and receiving, but also specialized sub-roles such as sending queries, sending updates, and other roles such as subscribing to receive updates.

Multiparty Communications Rule: (“commN”;

rule name (a string);

{Set of bindings of form role:element};

completion type:name:instance;

parameter vector of elements type:name:value)

The set of possible roles are a property of the type of the completion, and there may be more than one element bound to one role.

An interesting example of a multiparty communications completion is communication via a global shared memory presented as a distributed tuple space, with the following supposed properties [28]:

- there are separate sets of processes which create tuples and write values into the space, which subscribe to updates for certain tuples, and which query the space for values. Only certain interfaces of these process (which we have called here “role instances”) are bound to the tuple space. Other roles of the same processes might use socket communications.
- the tuple space has multiple servers, deployed on certain nodes (which are parameters of the completion), and each process interacts with the nearest server. These interaction bindings could be made explicitly, or could be inferred from the overall deployment in configuring the completion.
- the tuple space servers execute operations to synchronize their representation of the tuple space, so they all have a correct and complete representation.

This could be indicated by

(“commN”; “CommByMyTupleSpace”; {“sender:SENDER1”, “receiver:RECEIVER1”, “receiver:RECEIVER2”...}; layer:MyTupleSpace:TS; -)

4.5 Variables in Completion Indications

As they have been described, the indications may have parameters of any type (such as numeric, boolean or strings). It is also useful to allow parameter values indicated by variable names, which must be given values before the evaluation model is built. These unresolved parameters provide flexibility in coordinating and adapting completions to the circumstances of each evaluation. Where a system attribute is shared among many completions, the same parameter could be defined globally and appear in several completion indications.

The syntax suggested for the variable names is to begin them with a $ sign:

Variable name = $string

to identify variable names from other elements of a completion. This assumes that the specification tool does not use this convention for another conflicting purpose (if it does, then another convention could be substituted). Thus, in indicating the TCP completion above, a variable $netspeed could be used instead of 100 Megabits/sec.

("comm2"; "CommByTCPSocket"; "ANY user-node-element";
"ANY server-node-element";
network:TCPconnection;internet;
real:accessSpeedMegabits:$netspeed)

This value will have to be resolved before a model can be built. One way is to have a statement

$netspeed = 100

included in the specification with the completions, global to all completions, as already mentioned; another way is query the user at the time a model is built, as indicated in Figure 1.

Where a completion itself contains a further completion or annotation, it may be defined by a parameter name from the outer completion. Thus in the example of the database completions above, the MSQL-DB completion type could (just as an example) be defined as a process allocated to processor $DBPROCR, with relative execution speed $CPUSPEEDFACTOR. Its execution demand could be defined by an equation such as:

$CPUDemand = (0.3 + 0.01 * $records)/$CPUSPEEDFACTOR

In evaluating the CPU demand value to calibrate a performance model, the $records parameter has been supplied as a parameter in the indication of the completion given above (with value 5500). If the processor and its speed factor are not known, they would be presented as a query to the user while building the performance model.

Parameters can therefore be used to link together different levels, in defining completions in terms of additional completions. Also, keeping undefined parameters to be resolved by the user is an excellent way to allow flexibility at the time the evaluation model is created. However, if parameters from a variety of completions are presented at the user interface in Figure 1, an indication of where they were defined will be necessary to avoid name conflicts. The completion instance name could be prepended as CompletionInstance.$variable

The MSQL-DBS completion instance is Database2, so a user reference to the CPU speed would take the form

Database2.$CPUSPEEDFACTOR

4.6 Sets of Completions

In the rules described above, the specification of a single completion name or parameter value could be replaced by the specification of a set of candidates, so that either an installation rule can be used by the model-builder to choose the most appropriate, or the user can be queried for a choice, or the evaluation can scan over the entire set (and over combinations of such sets for different completions) in order to evaluate multiple configurations for a single design specification.

The more general form of the completion-defining tuple would indicate a set of values for a single element, as a comma-separated list contained in curly braces:

completion type:{set of names}:instance
parameter type:name:{set of values}

The indication described above for a TCP connection completion could be generalized to:

("comm2"; "CommWithChoice"; "ANY user-node-element";
"ANY server-node-element";
network:{TCPconnection, RPC}:internet;
real:accessSpeedMegabits{10, 100, 1000})

in which the protocol used is either sockets with TCP (defined in theTCPconnection completion) or a form of Remote Procedure Call (defined in the RPC completion) and the access speed to the internet is 10, 100 or 1000 Mbit/sec. The intention of the multiple choices could be either to make a choice later, or to evaluate all six combinations.

4.7 Integration with Specification Tools

Different specification languages may be able to incorporate these indications in different ways, so the tuple syntax defined above is just meant to identify the information. In the UML for example, elements can be stereotyped and annotated with tags. One way to indicate completions would be to use tags with strings which contain the tuples just defined. A proposed UML profile for performance [30] defines many tag types to define scenarios with performance measures, demands and workload parameters; these definitions could be extended with typed tags for each tuple type, and structured parameter fields.

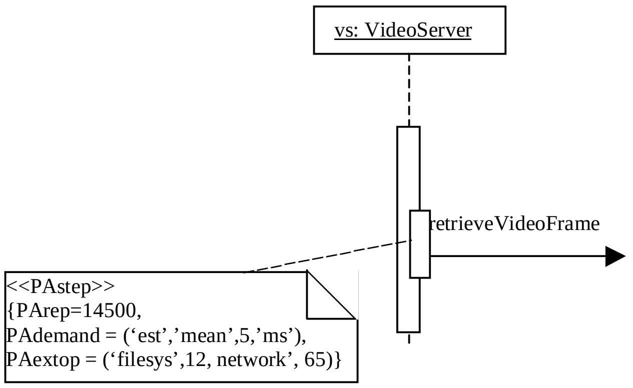
vs: VideoServer
retrieveVideoFrame
<<PAstep>>
{PArep=14500,
PAdemand = ('est','mean',5,'ms'),
PAextop = ('filesys',12, network', 65)}

Figure 4 UML tags annotated on a Sequence Diagram

Figure 4 shows a fragment of a UML Sequence Diagram to specify the behaviour of the same video server as in Figure 2, showing UML tag values defined in the profile, which define 14500 repetitions (video frames), and CPU demand of 5 ms, 12 file operations and 65 network packets per video frame, for the videoserver step which retrieves frames and sends them..

In the Use Case Map Navigator, some performance-related completions have been integrated into the language and the tool interface. Each specification element has a performance-related window for this information, including path choice probabilities, arrival rates at start points, deployment of processes to processors, and the workload demands of responsibilities. The window for the performance attributes of the GetVideoList responsibility in Figure 2 is shown in Figure 5.

Figure 5 UCM Navigator window for annotating a service

5. DEFINING THE COMPLETIONS

A performance completion is a transformation applied to the specification, which is tailored to the goal of predicting performance. There are similarities to design elaborations and refinements that are used in generative programming [11], but the completions themselves may be quite different. Performance completions need to describe only those scenarios which are important for performance evaluation, typically those with heavy use, and others whose timing is important for reasons such as system safety or security. In this way a large part of the functionality of a service or communications subsystem may be *missing altogether* from a performance completion (for instance, relating to system initialization, and some kinds of exception handling). Further, the description of the subsystem provided in the completion may be *approximate*. Functions may be lumped together if they are always performed together, and alternative paths may be combined and averaged.

The actual completion is the change that is inserted (preferably automatically, to avoid excessive effort) into the specification. This creates an expanded specification, which may not need to be read by the designer (but only used to create a performance model). There are three kinds of change, which we will call annotation, refinement and transformation.

5.1 Additional Annotation Completions

The simplest form of completion is an *additional annotation* in the specification. Effectively this converts the indication of the completion (which is already a kind of annotation) into one or more other annotations for performance properties or further completions. For example a completion indication for a particular database system might be converted into annotations giving execution demand figures and IO demands for that system. A parameter for database size on the completion indication could be used to compute the demand figures for this instantiation of the database.

5.2 Refinement Completions

A *refinement* is described by a sub-specification, which can replace a design stub, a connection, or an operation (which may be indicated in UCMs as a responsibility). It should have interfaces that match the path attachment points in the scenario.

A simple refinement is a server that accepts a request and sends a reply. It can be used wherever this behaviour pattern appears, which is often. A refinement to complete a service request could be described by a specification as a scenario, like the web server in Figure 1, or by information about the server and the services it offers, which defines:

ServerType	
Deployment	
Service1:	Execution demand per request
	List of: {Lower-level service; requests}
Service2: ….	

This is sufficient for understanding the performance impact of the server, at a coarse level of detail. Further servers would have their own descriptions for the services they offer, and their deployment.

A more powerful refinement is a subsystem like the CORBA layer in Figure 2b. It has internal behaviour and elements which may be system-wide, but it still has to conform to the interfaces, which in this case handle requests in one direction and responses in the other.

A different kind of refinement is an elaboration of behavoiur to show the sequence of responsibilities necessary to carry out a single high-level responsibility. Such a path eleboration is strictly at the level of the specification, and is indicated by a sub-specification. In UCMs there is a notion of sub-map which plugs into a point in the scenario to refine it; other specification languages provide similar abstractions (a process definition in process algebras or SDL, for instance).

5.3 Transformations

The most general form of completion requires transforming the specification in some defined way in order to progress in the details. This goes beyond additional behaviour specified in a subsystem behind an interface, and actually modifies behaviour in the original specification.

Transformations can be based on pattern substitution, in which one pattern of elements and interactions is replaced by another one. General theories, prescriptions, tools and experience for doing this are found in the graph transformation literature, using the ideas surveyed in [29].

An example of pattern substitution is the replacement of a multicast connection in a specification by a tuple space subsystem (described briefly above), in which the receivers request updates to certain tuples. In the completed specification the receivers have to actively request data from a tuple-space server, instead of passively waiting for the data to arrive, and then they have to test whether it has changed. The transformation begins by identifying the roles of senders and receivers, and then modifying their behaviour as illustrated in Figure 6. Tools for formal graph replacement are described in the reference.

.

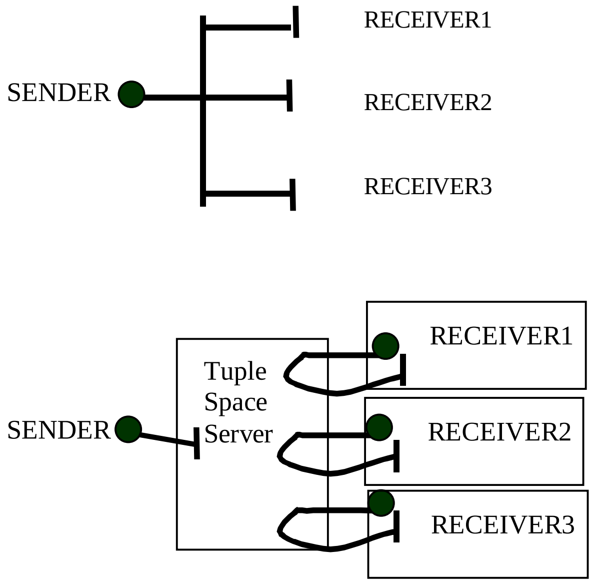

Figure 6 A transformation from a multicast to multiple retrievals from a Tuple Space

Another similar pattern-based refinement would arise in the use of a polling discipline for requests which are specified as being sent to a server (in polling, the server seeks out the requests instead of waiting for them).

5.4 Parameters of Completions

A completion may define and use a number of parameters, using concrete values or variables with the naming convention $variable described in the previous section. The values of variable parameters may be defined by the indication when it is bound into the specification, or by the completion itself, and they may be used in defining further (nested) completions.

Where a completion defines a value for a parameter it is treated as a default value which can be overridden by a value given in an indication.

Processors and other devices may have to be named as annotations in the specification of a completion. They are assumed to be indicated by strings which name the processors, and these can also be variables.

5.5 Insertion of Completions

The process of inserting a completion into a specification includes compatibility checks (to see if the interfaces are compatible) and parameter bindings. Compatibility checks may be more complex if the specification is more detailed. For instance if the specification describes data types transferred across an interface into a design stub, then a completion for the stub should be capable of accepting these types. In a sophisticated system the interface types might be exploited to create parameter values for the completion (for instance, communications overhead depends on message sizes, which might be defined as part of the specification of a service request.

6. RELATED WORK AND IDEAS

Other approaches to evaluating designs and architectures for performance either use predictive quantitative models, as we do here, or make qualitative analyses, which we shall not attempt to analyze. The problem of adding performance information to specifications has been handled either by using annotations alone, or by including submodels within the performance modelling environment for existing re-used components.

Smith in her pioneering work used a scenario specification which was dedicated to performance (rather than to software specification), with user-supplied demand parameters in place of annotations [33]. The user must capture and add up all the demands of an operation, on all devices. Smith’s modeling language has a high-level operation abstraction to act as a stub that can be substituted by a submodel, providing a limited capability for replacement completions. Her approach has been reflected in other work also, such as [10] and [18].

Attempts to obtain performance models from standard software specification languages such as SDL or the UML are roughly at the stage of defining annotations. Completions which go beyond this have not been considered, as far as we know, except for service demands in the proposal [30].

Performance models from other specification languages such as Petri Nets and Process Algebras have been widely studied [14], often by generating Markovian models at the state-transition level as described by Pooley [26]. One example that has been put to practical use in design is the PROTOB system [7], another is HIT [4].

Menasce and Gomaa described a language to specify the components and interactions of a system, with a tool to generate a queueing model. In building the model, they used a submodel for a database “stub” [21]. The Hyperformix Strategizer modeling tool uses pre-built performance sub-models for software and hardware components such as databases, networks and storage subsystems. Their approach is quite similar to that taken here, but is not designed for user-defined completions.

The present authors and our collaborators have for many years attached performance parameters to a software architecture model to define layered queueing models [15]. In [16], an attempt was made to provide a way to build these models from any executable software design tool, and this was realized in [38] for the ObjecTime Developer tool, which implements the ROOM methodology.

Models have been built from UCM specifications [37], [32] and from UML specifications [25]. For completions, we have found it useful to pass the description of certain kinds of subsystem completions through to the performance model, which can then incorporate pre-built submodels of the subsystems, as indicated in Figure 1 [32]. In this work, at least up to this point, the operations shown in Figure 1 have some automated steps, such as generating the performance model from the UCM, and some manual steps, such as invoking further completions.

7. CONCLUSIONS

Completions are essential for evaluation of non-functional properties, because specification languages have been (properly enough) designed to describe functional properties first. In work to date, a beginning has been made with annotations for workloads, deployment and resource demands. The idea of completions generalizes this to describe more complex information which is properly missing in many specifications, such as the functional behaviour of service layers. This idea is similar to software components, but is specialized for the particular evaluation (in this case, for performance).

With completions, we make a specification complete for an evaluation goal, (*performance completeness*) as well as in a functional sense.

Completions must be *indicated* within a specification, and this may be done using an annotation mechanism in the language, or by an added description which has been defined here using six classes of tuples. The classes of tuples include rules which can be used to infer the need for a particular completion from the semantics of the specification, and styles of development embodied in set of rules. The question of how to infer completions appears to open rich research possibilities. Rules for describing communications have more complex semantics than those for other elements. Indications may identify sets of completions, either for later resolution or for evaluation of multiple cases.

Completions must also be defined in a way that can be used to transform the specification, or to contribute to the performance model as it is being built. The general problem of transforming the specification can be approached by graph transformation guided by graph patterns. Special cases include generating annotations, substitution of a design stub by a sub-specification, and invocation of a service described by a submodel in the performance tool. Parameters defined in the indication may be used in calculations of parameters of the completion itself, or propagated into deeper-level completions. Unresolved parameters can be queried to the user before completing the model.

8. REFERENCES

[1] R. L. Bagrodia and C.-C. Shen, "MIDAS: Integrated Design and Simulation of Distributed Systems," *IEEE Trans. on Software Engineering,* vol. 17, no. 10 pp. 1042-1058, October 1991.

[2] S. Balsamo, P. Inverardi, and C. Mangano, "An Approach to Performance Evaluation of Software Architectures," in *Proc. of First International Workshop on Software and Performance (WOSP98),* October 1998, pp. 178-190.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice.* Addison-Wesley, 1998.

[4] H. Beilner, J. Mater, and N. Weissenberg, "Towards a Performance Modelling Environment: News on HIT," in *Modeling Techniques and Tools for Computer Performance Evaluation.* 233 Spring Street, New York, N.Y. 10013: Plenum Press, 1989, pp. 57-75.

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[6] E. Brinksma, H. Hermanns, and J.-P. Katoen, Eds., *Lectures on Formal Methods and Performance Analysis.,* Berlin: Springer, 2001.

[7] G. Bruno and G. Marchetto, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *IEEE Transactions on Software Engineering,* vol. SE-12, no. 2 February 1986.

[8] R. J. A. Buhr and R. S. Casselman, *High-Level Design of Object-Oriented and Real-Time Systems: A Unified Approach with Use Case Maps.* Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[9] R. J. A. Buhr, "Use Case Maps as Architectural Entities for Complex Systems," *IEEE Transactions on Software Engineering,* vol. 24, no. 12 pp. 1131 - 1155, 1998.

[10] V. Cortellesa, "Deriving a Queueing Network Based Performance Model from UML Diagrams," in *Proc. Second Int. Workshop on Software and Performance (WOSP2000), ACM order no. 488003,* Ottawa, Canada, 2000, pp. 58-70.

[11] K. Czarnecki and U. W. Eisenecker, *Generative Programming.* Addison-Wesley, 2001.

[12] D. Cameron et. al., "Draft Specification of the User Requirements Notation," Canadian Contribution to ITU-T,, Sept. 2001.

[13] J. Dilley, R. Friedrich, T. Jin, and J. A. Rolia, "Measurement Tools and Modeling Techniques for Evaluating Web Server Performance," in *Proc. 9th Int. Conf. on Modelling Techniques and Tools,* St. Malo, France, June 1997.

[14] R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds., *Performance Engineering.,* Berlin: Springer, 2001.

[15] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, "Performance Analysis of Distributed Server Systems," in *The Sixth International Conference on Software Quality (6ICSQ),* Ottawa, Ontario, 1996, pp. 15-26.

[16] C. Hrischuk, J. A. Rolia, and C. M. Woodside, "Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype," in *Proc. of Third Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '95),* Durham, NC, January 1995, pp. 399-409.

[17] P. Jogalekar and M. Woodside, "Evaluating the Scalability of Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 6 pp. 589-603, 2000.

[18] P. Kahkipuro, "UML Based Performance Modeling Framework for Object Oriented Systems," in *UML99*, Berlin, 1999.

[19] 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.

[20] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, "Automated Performance Modeling of Software Generated by a Design Environment," *Performance Evaluation*, vol. 45, no. 2-3 pp. 107-124, 2001.

[21] D. Menasce and H. Gomaa, "A Method for Design and Performance Modeling of Client/Server Systems," *IEEE Transactions on Software Engineering*, vol. 26, no. 11 pp. 1066-1085, 2000.

[22] A. Mitschele-Theil and B. Muller-Clostermann, "Performance Engineering of SDL/MSC Systems," *Journal on Computer Networks and ISDN Systems*, vol. 31, no. 17 pp. 1801-1815, 1999.

[23] T. J. Mowbray and T. Zahavi, *The Essential CORBA.* Wiley, 1995.

[24] D. B. Petriu, "*Layered Software Performance Models Constructed from Use Case Map Specifications*," Master's thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, 2001.

[25] D. C. Petriu and X. Wang, " Deriving Software Performance Models from Architectural Patterns by Graph Transformations," in *Theory and Applications of Graph Transformations, TAGT'98., Lecture Notes in Computer Science 1764*, H.Ehrig, G.Engels, H. J. Kreowski, and G. Rozenberg, Eds. Berlin: Springer, 2000, pp. 475-488.

[26] R. Pooley, "Software Engineering and Performance: a Roadmap," in *The Future of Software Engineering, part of the 22nd Int. Conf. on Software Engineering (ICSE2000)*, Limerick, Ireland, June 2000, pp. 189-200.

[27] R.L. Nord and B. C. Cheng, "Using RMA for Evaluating Design Decisions," in *Second IEEE Workshop on Real-time Applications*, Washington, DC, July 1994.

[28] A. Rowston and A. Wood, "An efficient distributed tuple space implementation for networks of heterogenous workstations," Department of Computer Science, University of York, Technical Report YCS-270, 1996.

[29] G. Rozenberg, Ed. *Handbook of Graph Grammars and Computing by Graph Transformation.*, Singapore: World Scientific, 1997.

[30] B. Selic, A. Moore, M. Bjorkander, M. Gerhardt, B. Watson, and M. Woodside, "Response to the OMG RFP for Schedulability, Performance and Time," Obect Management Group, OMG document ad/01-06-14, June 12,2001.

[31] B. Selic, M. Woodside, C. Hrischuk, and S. Bayarov, "A Wideband Approach to Integrating Performance Prediction into a Software Design Environment," in *Proc. of First International Workshop on Software and Performance (WOSP98)*, October 1998, pp. 31-41.

[32] K. H. Siddiqui and C. M. Woodside, "Performance Aware Software Development (PASD) Using Execution Time Budgets," in *Proc. MICON 2001*, Ottawa, Canada, Aug. 2001.

[33] C. U. Smith, *Performance Engineering of Software Systems.* Addison-Wesley, 1990.

[34] C. U. Smith, "Software Performance Engineering," in *Encyclopedia of Software Engineering.*, 1993.

[35] C. Szyperski, *Component software; beyond object-oriented programming.* New York: ACM Press, 1998.

[36] Telecommunication Standard Sector of ITU, "Specification and Description Language (SDL)," ITU-T Recommendation Z.100, Int. Telecommunications Union, March 1993.

[37] W. C. Scratchley and C. M. Woodside, "Evaluating Concurrency Options in Software Specifications," in *Int. Conf on Modelling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, College Park, MD, Oct. 1999, pp. 330-338.

[38] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, "Automated Performance Modeling of Software Generated by a Design Environment," *Performance Evaluation. To appear* 2001.