This article appeared in the Proceedings of the First Int. Workshop on Software and Performance (WOSP98), pp. 31-41, Santa Fe, October 1998.

A Wideband Approach to Integrating Performance Prediction into a Software Design Environment

Murray Woodside*, Curtis Hrischuk*, Bran Selic**, Stefan Bayarov**
* Dept. of Systems and Computer Engineering, Carleton University, Ottawa
** ObjecTime Limited, Ottawa

Abstract

Performance predictions for software designs are needed to give early warnings of problems such as resource saturation or excessive delays. A wideband approach to performance prediction is one which can be applied to a wide range of design descriptions and parameters, ranging from incomplete early descriptions and estimates, to target system measurement data. The paper describes a practical development of such an approach, designed to be integrated with an existing software design environment called ObjecTime. It uses a performance model to make predictions about performance measures such as the average time an actor or processor is busy, the communication overhead of a scenario, or the delays in processing an input event. Automation shrinks the cost and time for model building, and synchronizes the design and model as the design develops. The wideband approach can be applied throughout the software life-cycle, from early sketchy designs until after deployment.

1.0 Positioning

There is an increasingly evident need for better methods and tools for predicting and improving the performance of software products. The difficulty is most acute in the early stages of design, or for extrapolation beyond the present status of the design (e.g., for scalability analysis). Performance models are the natural way to provide predictions for a range of environmental conditions or design alternatives and these predictions can be used to detect problems and suggest corrections. However, modelling is at present too slow and too costly. We describe a new, evolutionary, low cost wideband approach to performance analysis using models, including how it can be integrated into a typical high-level design environment. The domain of application of this environment is real-time interactive software, such as communications software. In this domain there are significant uncertainties in the execution demands and delays, and performance evaluation is statitisical, based on means, variances and percentiles. Hard deadlines, in which missing a deadline means a system failure, are not a major concern here.

Intuition is a poor guide to creating a design with good performance. There are too many factors, and it is confusing that some of them apply only for certain situations or data configurations. Real-time software designers often have only the vaguest idea of the performance impact of certain key design decisions. It is not uncommon in practice to find that performance predictions made by design teams are off by an order of magnitude and more! Even when formal performance predicting models of software are actually constructed they may not be trusted.

Under such circumstances, designing for performance is a question of trial and error where the intuition and experience of the design team plays a key role. Putative configurations are proposed, the corresponding performance models derived and evaluated, and the prediction results compared against requirements and previous results. To make this process more effective as much of it should be automated as possible. Automation can reduce the turnaround time and improve the reliability of the overall process.

In particular, the following opportunities for computer automation have been identified:

- Mapping of problem-specific configuration data into attributes of a model (resource functions).
- The derivation of the performance prediction model directly from the design and its execution traces.
- Evaluation of the model.
- Mapping and interpretation of the evaluation results in terms of problem-domain entities (traceability) for faster and more reliable problem diagnosis.
- Comparison of new versions to previous results.

This paper describes the first three aspects. The fourth (diagnosis) is a deep problem; comparison of results is less complex but is not discussed here.

A special opportunity arises with High-Level Design (HLD) tools. Using the tool the user creates a “design”,

Copyright 1998 by the Association for Computing Machinery. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or fee.

which in the case of ObjecTime is a description of a set of communicating "actor" processes, controlled by state machines with code fragments and by protocols for communications, and including data objects which exploit inheritance. A design may be rather sketchy, with just a few states and transitions, or the control actions may be defined in great detail. In either case it is possible to "execute" the design over a scenario by inserting events, stepping through the state machines, and executing whatever code or actions have been defined.

Performance prediction data from different sources are listed in Figure 1. Measurements made on the product or a code prototype (on the right) give solid and detailed test data and are a familiar and well-understood area [35]. Our goal is to supplement it by prediction by models, on the left, for earlier and more rapid evaluation. Modelling is more difficult and its usability needs to be improved. Where it has been used, modelling has a solid track record in identifying serious design problems and evaluating alternative solutions. (This case will not be argued here, but evidence is given for example in Smith [22] and by [11].)

Across the middle of Figure 1 is a range of techniques for obtaining performance estimates. Queueing models use estimates of the resource demand for each scenario to analyze contention for devices, and then to predict device saturation and response delays. Petri nets are sometimes used in a similar role. Moving to the right there are more sophisticated models, (layered queueing networks or extended queueing networks) and more detailed evaluation techniques (first analytic, then simulation, then symbolic execution of the software design, then measurements on target systems). Below each source is a relative (ball-park) estimate for the time it takes to obtain predictions based on one scenario; for the measurements there is a set-up time and cost to establish the desired configuration, both of which might be quite large. These times are notional, based loosely on experience and on small systems; for a complex case they may all be longer. The time to do a single measurement can add up to days, assuming that the measurement facilities are available for use.

Even with this wide range of sources of data available, the typical performance analysis practice is to do an expensive, isolated *one-of* analysis at the end of development, usually consisting of measurement activities, and then to tune the code. Smith has termed this the "fix-it-later" approach and documented the seriousness of the problems it creates [22].

The goal of the present *wideband approach* is to create a consistent framework from left to right of Figure 1. It can detect performance problems as early as possible and, in the event expectations are not met, to provide useful information for tuning. It starts early, retaining information gathered from many sources, so that a more accurate, detailed, and coherent picture develops as the design evolves. It incorporates data from CASE tool designs or code prototyping, from test execution, and from measurements made on the finished product, as indicated in Figure 2. This work uses the *Layered Queueing Network* model (LQN) [14, 9] but the approach could be adapted to other models. The early information jump-starts the analysis and also makes it cheaper to continue the analysis as the design progresses. Using common tools and viewpoints at each stage gives continuity and traceability. The idea of performance viewpoints and how to exploit them was described in [13].

2.0 The Keystone Abstraction: an LQN Model

Central to the *wideband approach* is the use of an abstract performance model of software execution that captures an actor's resource demands, and contention. (*Actor* is the name we will use for software modules.) Resources include the usual hardware devices, actors that provide services to other actors, buffers, critical sections, and so on. A Layered Queueing Network (LQN) model is a generalization of queueing networks that represents these in a way familiar to software developers, in terms of requests for service between actors and the queueing of messages at an actor. An LQN can represent any queueing network model as a special case . In an LQN, actors are modeled as *tasks* which can accept service request messages at an LQN *entry*. An entry describes and models a service provided by an actor, and also the associated resource demands. If a task is a kind of object, an entry is a kind of method.

Prediction by Models				Evaluation by Measurement		
Queueing model	Layered Queueing Model		Behavior simulation (ObjecTime) (HLD tool)	Code prototype	Lab-scale system	Fully deployed application
	analytic	simulation				
0.1 sec	2 sec	5 min	5 hours	20 min to days (ignoring setup time)		
(analytic model)		(simulation model)		(instrumented execution and monitoring)		

Fig. 1: A Wide Spectrum of Sources of Performance Evaluations

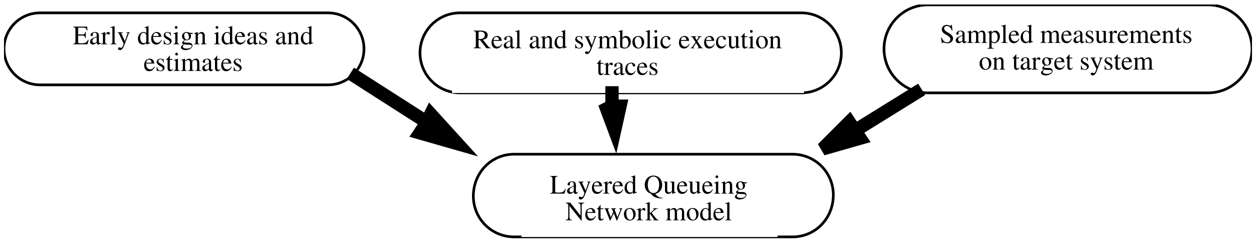

Fig. 2: Flows of Information into the Performance Model

Requests for service are characterized as entry-to-entry interactions sent in a synchronous (i.e., *send-wait-reply*), or asynchronous (i.e., *send-and-continue*) fashion. The interaction types affect performance because they affect task blocking, queueing delays at devices and message queues, parallelism, and resource contention.

The unit of concurrency for an LQN model is a scheduled concurrent operating system process or thread that we refer to as a *task*. One or more actors in the design can be mapped into one task, at the designer's choice, and the choice is also made in the model. To analyze the maximum possible concurrency, each actor is a task; to analyze the minimum concurrency, all actors are mapped into one task. Most real cases lie in between.

2.1 An Example System

Figure 3 shows the structure of a simplified network management system. There is an actor representing the *Visual Interface*, an actor for a *Management Application* that must collect data from across the system, as well as two *Device Agent Actors* to represent a multiplicity of agents deployed in the system which access local data and send it to the Management Application. Additional system features such as directories, ORBs, and other agents are left out of the design for the sake of simplicity.

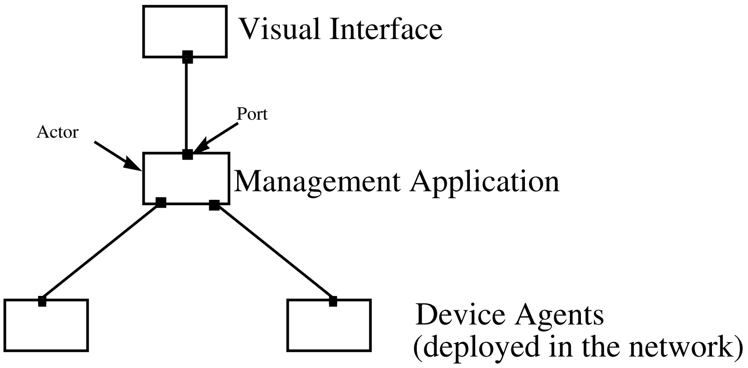

Fig. 3: A Network Management System as an Example

The behavior of the Visual Interface is to request data from the Management Application and to display the results. The behavior of the Management Application is to poll a set of agents with a delay between each poll, storing the results. The Device Agents simply accept a request and return a result. All communication is synchronous. The Visual Interface and Management Application both execute on the same processor. The expected, budgeted resource demand for the service each actor provides is shown in Table 1.

The resources that limit the performance of the system are partly the processors that run the actors shown, but there is also a layered software resource in the Management Application. It waits for a response to each Agent query, which delays its responding to the Visual Interface, even when the processor is lightly loaded. The capability to replicate the Management Application is a software design decision. If it is not replicated then it, rather than any processor, may be the system bottleneck. If replicas are used, implanted as lightweight threads, several of them can be active at once and they can maintain a higher rate of processing.

Description	Budgeted Resource Demand (msec.)
The visual interface's CPU demand to update the screen	1.0 per agent + (10.0 for overhead)
Management application's CPU demand to respond to the visual interface's request	0.25 per agent + (2.0 for clean-up after sending the reply)
Management application's CPU demand to query an agent	3.0 per query
Round-trip delay until the management application receives an agent's reply	4.5 per query
Minimum time between polls to an agent	1000.0

Table 1: Budgeted Resource Demands and Polling Delay

We can test our reasoning by examining the performance predictions, where a model is used that varies the number of agents and number of Management Application replications. The performance predictions are graphed in Figure 4. As shown, the contention effects can cause the average waiting time by the Visual Interface to become quite large unless the Management Application is replicated four or five times. From the average number of busy Management Application replications we find that a replication value of five should be sufficient. Thus the software must be designed to support replicated Management Application servers.

Other performance issues that the LQN may address include the correct setting of task priorities to achieve a set of quality-of-service guarantees such as response time percentiles, the impact of actor replication, the impact of combining some actors into one task (limiting their concurrency but reducing overhead of operation), the sensitivity of performance to a change in the workload or the number of users, or various kinds of scalability.

3.0 Process and Automation

The design evaluation process described here is based on scenarios, and this differentiates it from methods which simply embed parameters in the design objects (e.g. [15]). The use of scenarios is valuable because it directly identifies the kinds of response being analyzed, may identify several types of response, and allows the analyst to focus on just the most important cases from a performance point of view (as described in [12]). It is also fits in well with Use Case analysis in early design work, which many software engineers have adopted.

Figure 5 traces the process. The engineer defines the scenarios and the performance requirements for each one (1), and then defines and validates the software design (2),(3) using the HLD tool (e.g., ObjecTime). Resource values are entered in a special facility described in Section 5 (4). The core of the performance machinery is the low-cost, automated construction (5) of an LQN model from the high level design and the set of scenarios. A scenario is just a sequence of expected events following from a certain input stimulus (or set of stimuli), and a certain initial state.

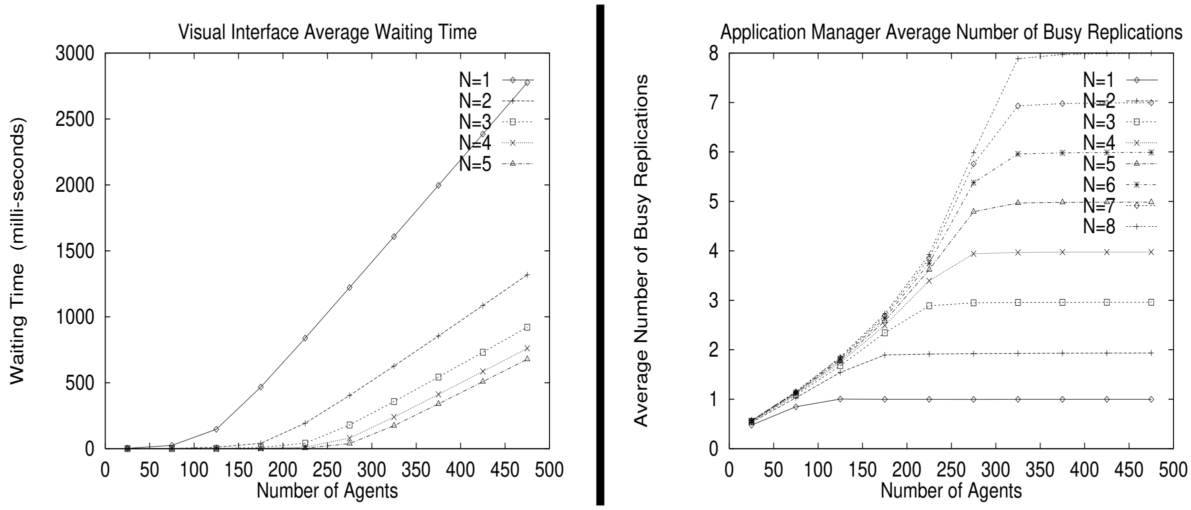

Fig. 4: Performance Impact of Replication

·

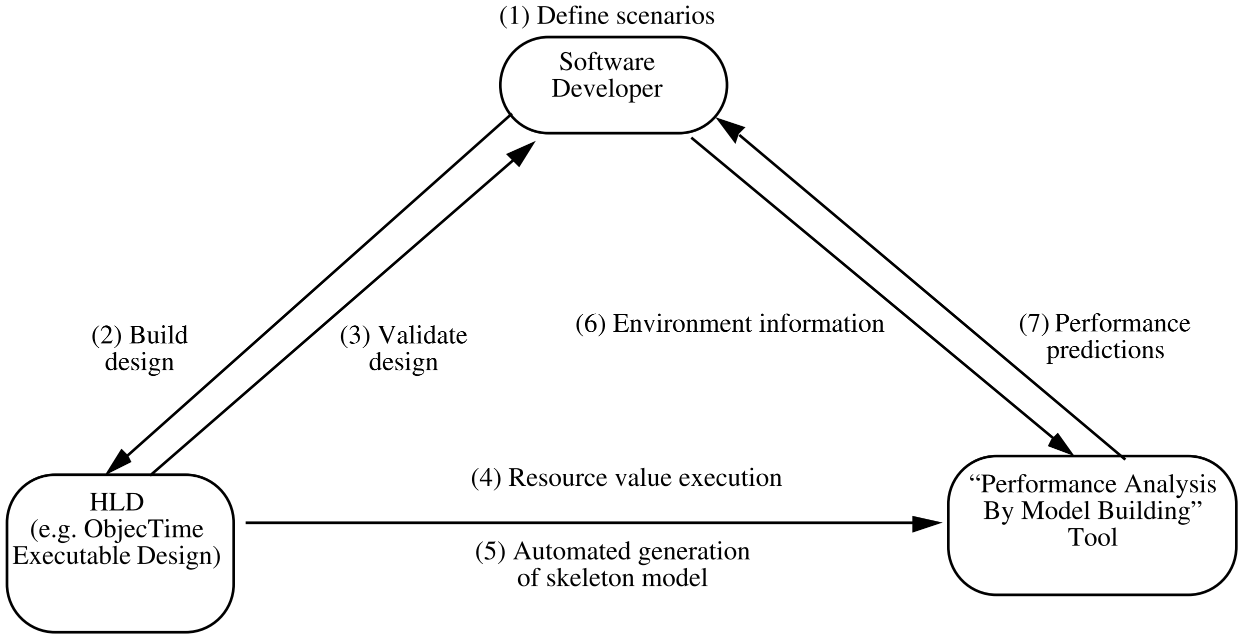

Fig. 1: The Developer's Use of PAMB

Typically in the design tool we trigger the scenario just by applying the input stimulus, and the executable design goes through the scenario as a reaction. Finally the analyst or designer specifies an execution environment (6) and gets performance predictions back (7). The remainder of this paper will be described in terms of a particular HLD tool, ObjecTime, which is used with the ROOM development method [16].

4.0 The "Performance Analysis by Model Building" (PAMB) Tool

PAMB implements an automated model-building technique which uses traces as its input [3, 4, 5]. Traces are a low cost source of behavior information because they can be automatically generated from a variety of sources. Traces reveal the dynamic details of a design that are difficult to determine from source code or documentation but are important to performance, such as: data dependent branching, the identity of tasks involved in anonymous or dynamically bound interactions, and the involvement of the polymorphism and inheritance hierarchy of an object-based system.

A special tracing format called an *angio trace* is used, which was introduced in [2, 3]. As described in [2, 3], an angio trace follows the causal path of a scenario from actor to actor (or task to task, in fully deployed software) and allows the interaction semantics (asynchronous, RPC or "forwarding") to be extracted and the interaction architecture to be detected.

Building a model with PAMB proceeds as follows, with reference to Figure 1,

(1) and (2), Generate the angio traces for the selected scenarios by executing the ObjecTime design. A Message Sequence Chart (MSC) [1] can be generated from each scenario, and used to visualize the results.

(3) Abstract the trace into an intermediate graph form [5].

(3) Detect the task, entries and interactions by matching templates with parts of the intermediate graph. This gives a partial description, called a *sub-model*, for the one scenario.

(3) Build a "skeleton" LQN model by attaching the submodels together.

(3) Finish the model by introducing processors and other devices and assigning parameter values which configure the model for its target environment.

The last step introduces parameters for resource demands and the target system's environment. Some of these parameter may be variables which will be systematically assigned values to conduct a what-if analysis, sensitivity analysis, or trend extrapolation. Resource demands are

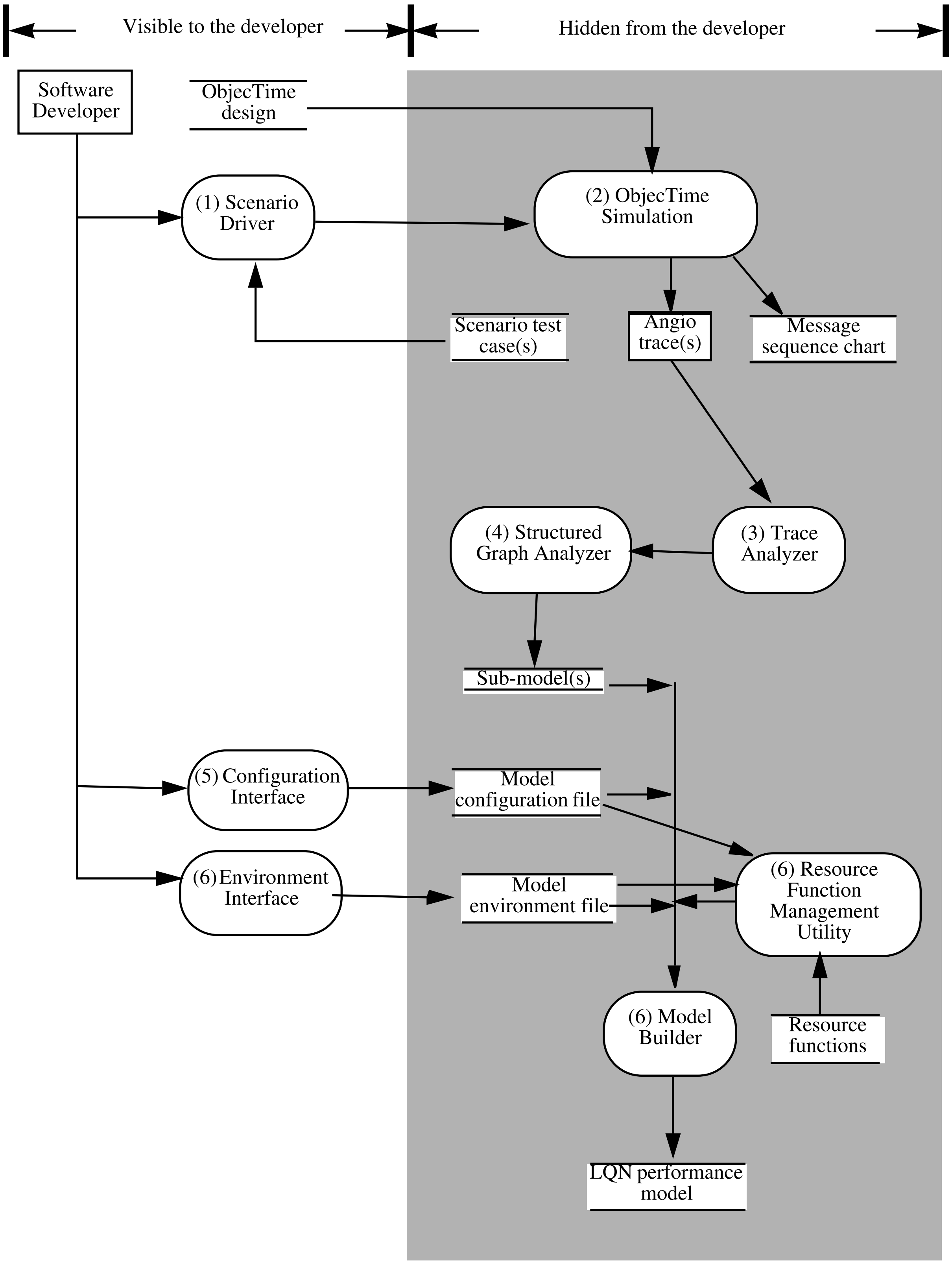

Fig. 1: Building a Model with PAMB and the ObjecTime HLD Tool

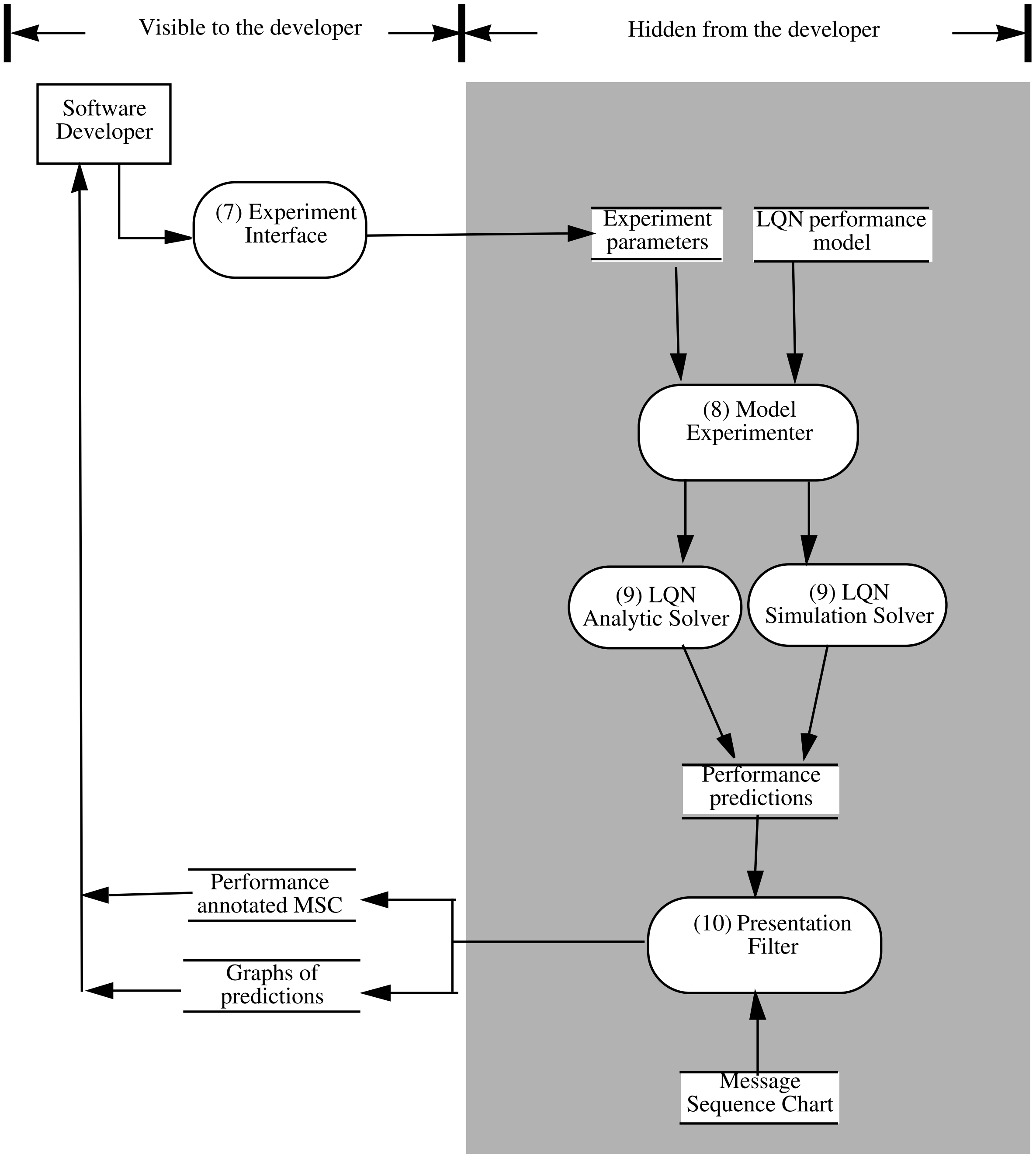

Fig. 1: **Evaluating and Interpreting the Model's Performance Predictions**

introduced for each actor's activities using a set of resource functions supplied by the Resource Function Management Utility (described in section 5.0). An event in the execution of the actor can record an *activity identifier* which is assigned representative resource demands by a *resource function*. Parameters for the target system's environment, include: workload intensity (e.g. invocations per second by scenario), device attributes (e.g. scheduling policy, service rate), task attributes (e.g. replication value), and resource assignments (e.g. actor to task assignment and task to CPU assignment).

As well as being used to build a model, PAMB has an interface to vary a model's parameters, evaluate the model, and present the resulting performance predictions. The steps in this process are shown in Figure 1, which are:

(1) The remaining parameter variables are assigned a value, or a list of values to evaluate, by the developer.

(1) and (9) For each variable assignment a model is generated and evaluated by analytic or simulation means.

(3) Once the suite of models are evaluated the performance predictions are gathered, filtered, and sorted by scenario, task, and device. They are presented to the software developer as MSCs with performance annotations, or as a set of graphs.

If external entities need to be incorporated into the model the software developer can describe them in the design and their behavior will be characterized in the angio traces for inclusion in the model. The model can be enhanced by adding plug-ins for entities that are not represented in detail in the design (like protocol stacks, ORBs, networks or buses) but will be present in the target environment. The developer can also adjust the model to represent different deployments, using different plug-ins.

The use of the angio trace as the input behavior description allows the wideband approach to accommodate many situations, such as multiple ObjecTime designs that interact, a multi-site development effort, or components for which the source design is not available (e.g., due to intellectual property rights). An angio trace can also be extracted from many sources, such as annotated specifications (e.g. Use-Cases [57] or Message Sequence Charts [1]).

5.0 Resource Data and its Management

Resource data is needed for CPU demands of operations in the design. At this stage, we have only considered operations by code generated automatically by the ObjecTime tool, running on a run-time library called the *MicroRTS* (which is available for many platforms). As shown in Figure 8, the MicroRTS insulates the applications from the underlying operating system. It provides a small number of object classes and primitive operations used by an ObjecTime design. These basic services are standardized across target platforms so they can be measured and serve as resource functions, being available early in the design process.

A *Resource Function Management Utility* (RFMU) manages the resource functions, as well as providing tools to systematically make measurements. Figure 1 shows how it communicates with the tools to provide resource demand parameters for a model, passing a resource function identifier (with optional parameter values) and returning a resource demand value.

Data gathering for the RFMU is automated by a *test harness* which runs the measurement *test suite,* to measure, collect, and store the resource demand data about the MicroRTS primitives. This approach was taken because there are a large number of standard target platforms to test (over 60), where a *target* was considered to be a particular RTOS, a processor model and a specific compiler. The test harness systematically runs the test suite on each target by automating the compilation, target loading, running, and output result collection. It is developed in Expect/Tcl scripts [7, 8], and provides an easy way to add a new test model or target platform.

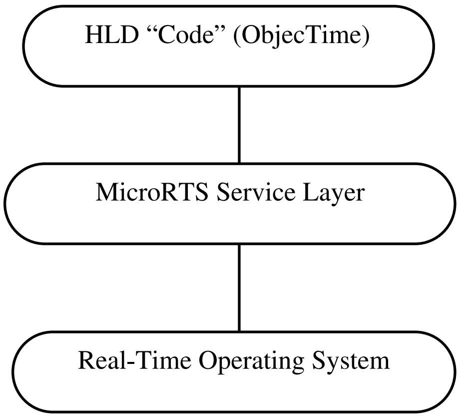

Fig. 5: Software Layers in the Target Environment

The test suite measures the resource demands of all of the MicroRTS primitives. The measurements are made using the RTOS’s standard timer utility. This does impose constraints on the usage of the resource functions (e.g., granularity) but it is sufficient for a first cut. As we gain experience, the test suite will be fine-tuned and, if needed, better measurement tools will be used.

The measurement test suite is broken into the three main categories of MicroRTS services:

(1) Communication services: the methods for asynchronous and synchronous communication between actors on the same task, on different tasks, or in different ObjecTime designs.

(3) Data services: like most object-oriented systems, variables are objects that belong to a class, and basic data manipulation methods exist for the class (e.g., instantiating or deleting a variable).

(3) Frame services: an ObjecTime design has a detailed structural component for which separate classes and methods are provided; they belong to the Frame services. This service layer provides functionality to incarnate (i.e., create) or destroy optional actors, to import or deport actor incarnations, as well as other services.

A developer can use the RFMU to measure user-defined operations, store the measurements, and reuse them as resource functions.

6.0 Exploiting the Approach

Not only does the wideband approach accommodate information from a multitude of sources but it can be applied throughout the life-cycle of the design. It can begin early in the specification and requirements stage for feasibility tests, continue with periodic design assessment and tuning, and then go on into post-deployment performance considerations (e.g., capacity planning). A brief description of the application of the approach is given, including how it can focus the effort of tuning the system performance.

6.1 Specification and Requirements

The specification and requirements stage describes the performance expectations for individual scenarios, and defines a workload mix of scenarios for performance modeling. The performance expectations can be average values for device utilization (e.g., network, processors, disks), throughput, or end-to-end response times. Performance design constraints can be allocated as well, such as budgeting the actor interaction overhead per scenario.

6.2 Design Development

The actor software architecture heavily influences the performance of the system so it is important to get feedback on its performance characteristics before a large commitment is made. It is also useful to know when a design is satisfactory, to prevent design churn. Using PAMB, developers get immediate feedback on the overhead of the actor software architecture. More investigative approaches with fully developed models can be done at key points in the design process, when feedback on the performance is needed to select between design alternatives or assess the feasibility of a given design.

To help individual designers proceed, performance expectations should be broken down for individual subsystems, with references back to the system requirements to give traceability. This is useful during performance reviews, where performance expectations are compared with predictions.

A smooth transition to measurement-based evaluation can be made if the target platform c a provide angio traces to be fed into PAMB. The target system can also improve the calibration of user resource functions. The predictions themselves can be supplemented and calibrated from target measurements in the model.

6.3 Tuning the Performance of a Design

Design tuning may involve altering the actor software architecture or interactions, delegating work from one actor to another, or using a different allocation of actors to tasks. The general tuning process follows steps #4 through to #8 of Figure 6. Once the model is generated (steps #1, #2, and #3) and evaluated (step #4), the performance analysis is complete if the performance predictions meet the performance objectives (step #5).

If performance is unsatisfactory, designers can usually respond based on their deep understanding of the task. Certain improvements can be tested without changing the design definition at all: one can change the environment (step #7), by reallocating tasks to processors, adjusting a task's priority, or providing faster hardware.

The goal of the PAMB tool and process is to be usable by most designers, with modest assistance from a performance expert. In difficult cases where a specialist is still needed, the wideband approach helps by providing a model and data ready made, with which the software developers and performance expert can communicate efficiently, focusing on the problem and the principles of software performance analysis rather than model building.

6.4 Post-Deployment

Software is never static but is constantly undergoing (re)construction. The wideband approach can be used to study proposed features, by combining a sub-model from a deployed system with sub-models generated from a design of new features. Other uses of a model of the deployed system include capacity planning for installations, and performance management.

7.0 Status and Conclusions

A prototype PAMB tool has been developed which automatically constructs models from ObjecTime designs, evaluates them, and presents the resulting performance predictions. The interfaces shown in Figures 6 and 7 are text-driven in the prototype. The presentation of results is by graphs. The prototype in an early form has successfully analyzed an industrial angio trace with over 10,000 events generated by a system with over 50 actors.

There are at least three requirements for a practical process for predicting and improving a software design's performance.

- The first one is to provide predictions with a low cost and small delay, and this is satisfied by the automation in the PAMB tool prototype.
- The second is to reduce the need for performance expertise and this is also satisfied to a large degree. Partly this is due to the use of

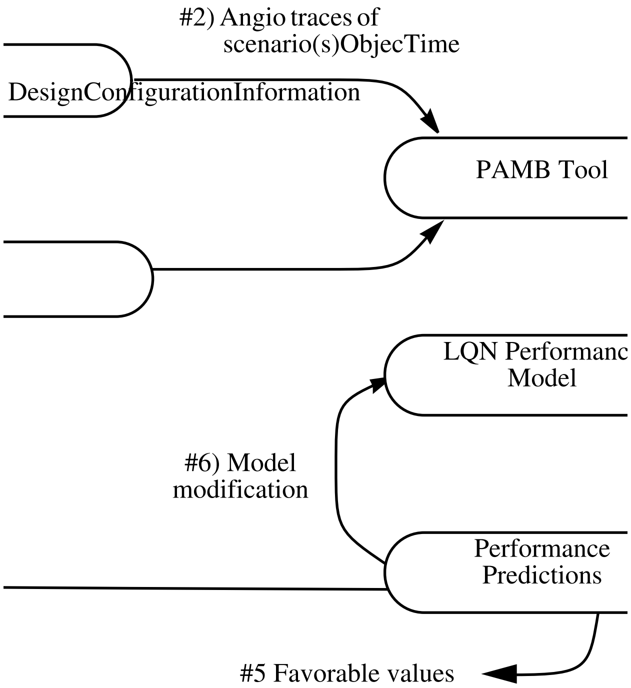

Fig. 6: **Tuning with the Wideband Approach**

layered models, which show the performance results in terms of software entities. When expertise is needed, there is already a framework to find a solution: the performance consultants are only involved when difficult problems are found, the developer remains responsible for the application aspects of the design, and the model is the common vehicle for discussion between the parties.

- The last requirement (identified but not addressed here) is the need for a methodology to exploit these tools to diagnose problems and identify solutions. Although a solution may be obvious once a performance problem is identified, some developer training is needed (e.g., for interpreting predictions or in design tuning techniques) and diagnosis aids are needed. This is an exciting area for research.

The "wideband" attribute of this work derives from the ability to derive models from many sources. For insatnce, similar ideas have been applied to another HLD tool and designs in the standardized SDL language [17]. The trace based model building has also been applied to UNIX execution.

With the wideband approach, developers can become aware of a design's performance. This awareness will replace the existing knowledge void that the developer currently has about performance, due to a lack of methodology and tools. This may be the most significant contribution of the wideband approach because it empowers the developer to identify and fix performance problems before they are set in concrete.

Acknowledgments

This research was funded by ObjecTime Limited, by TRIO, the Telecommunications Research Institute of Ontario, by OCRI, the Ottawa-Carleton Research Institute, and by the Industrial Research Chairs program of NSERC, the Natural Sciences and Engineering Research Council of Canada.

References

[1] CCITT. "Recommendation Z.120: Message Sequence Chart (MSC)." Technical report, Geneva.

[2] C. Hrischuk. "The automatic generation of software performance models using prototypes." Master's thesis, Dept. of Systems and Computer Engineering, Carleton University, 1995.

[3] C. Hrischuk, J. Rolia, and C. M. Woodside. "Automatic generation of a software performance model using an object-oriented prototype." In *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'95)*, pages 399–409, 1995. http://www.sce.carleton.ca/ftp/pub/rads/syncWthread.ps.Z.

[4] C. Hrischuk, C. M. Woodside, J. Rolia, and R. Iversen. "Trace-based load characterization for generating software performance models." Technical Report SCE-97-05, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada K1S 5B6, 1997.

[5] C. E. Hrischuk, *Trace-based Load Characterization for the Automated Development of Software Performance Models*, PhD thesis, Carleton University, Ottawa, Ontario, Canada, http://www.sce.carleton.ca/rads/rads.html, 1997. In preparation.

[6] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, New York, New York, 1991.

[7] D. Libes, *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*, O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1994.

[8] J. K. Ousterhout, *Tcl and Tk Toolkit,* Addison-Wesley, 1994.

[9] J. Rolia and K. Sevcik, "The method of layers," *IEEE Transactions on Software Engineering*, 21(8):689–700, August 1995.

[10] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy,

and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632 USA, 1991.

[11] F. Sheikh, J. Rolia, P. Garg, S. Frolund, and A. Shepperd, "Layered modeling of large scale distributed applications." In *Proceedings of the 1st World Congress on Systems Simulation*, pages 247–254, September 1997.

[12] C. U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley Publishing Co., New York, NY, 1990.

[13] C. M. Woodside, "A three-view model for performance engineering of concurrent software," 21(9):754–767, Sept. 1995.

[14] C. M. Woodside, J. E. Neilson, D. Petriu, and S. Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software," *IEEE Transactions on Computers*, 44(1):20–34, Jan. 1995.

[15] R. Henke, H. Konig, A. Mitschele-Thiel, "Derivation of Efficient Implementations from SDL Specifications Employing Data Referencing, Integrated Packet Framing and Activity Threads", Eighth SDL Forum, North-Holland, Sept. 1997.

[16] B. Selic, G. Guleckson, P. Ward, "Real-Time Object-Oriented Modeling", Wiley, 1994.

[17] H. El-Sayed, D. Cameron, M. Woodside, "Automated Performance Modeling from Scenarios and SDL Software Design", Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE), Kyoto, April 20-21, 1998.