Software Engineering and Performance: A Road-map

Rob Pooley
Department of Computing and Electrical Engineering, Heriot-Watt University
Edinburgh EH14 4AS, United Kingdom
+44 131 451 3367
rjp@cee.hw.ac.uk

Performance analysis, performance engineering

• The study of the performance of computer systems and networks attempts to understand and predict their time dependent behaviour.

• The overall process of estimating performance is often referred to as performance analysis.
Integration within the engineering process is *performance engineering*.

- As well as modelling, this involves measurement of real systems.
- Recent work has tried to combine estimation of a system’s performance with estimates of its reliability, giving a joint measure, called performability.

**Performance engineering techniques**

- Performance engineering is essentially an experimental approach to predicting the likely performance of systems.
- It can involve
  - *building and then monitoring a typical system, under the workloads of interest*,
  - *or using models which reproduce the time dependent behaviour of an unrealised system*,
    - *either driven by a trace of the expected workload*
    - *or by a workload represented as a set of random variables (stochastic modelling).*
The advantages of modelling include:

• estimates are made where a system does not exist yet or is too costly to buy to monitor;

• the workloads possible with a model may not be easy to generate on a real system;

• models can generate almost any measures we wish, which monitoring may not match;

• a model can test conditions which would damage the real system.

Typical representations for performance models

• queueing networks,

• Petri nets (especially timed and stochastic extensions),

• a variety of proprietary simulation languages and notations

• most recently, timed and stochastic extensions to formal description languages and process algebras.
Figure 1: Typical queueing network

Figure 2: Typical stochastic Petri net
Solution

- The main solution techniques are analytic, numerical and simulation.

- When we solve a model we can obtain an estimate for a set of values of interest within the system being modelled, for a given set of conditions which we set for that execution.

- The failure of traditional engineering and computer science education to build in an effective introduction to notions of probability, as a key abstraction and approximation technique have tended to leave research in these areas outside the mainstream of software engineering.

- Most engineers are happy to replace some complex behaviour with a simple delay when estimating time dependent behaviour. Most will happily use upper and lower bounds to estimate operational tolerances. Surprisingly few are capable of generalising this to use appropriate stochastic variables for such estimates.
Software performance engineering

- Performance analysis techniques have suffered a lack of acceptance in the wider software design community.
- Several reasons have been put forward for this.
  - The most compelling is the reluctance of designers to learn the specialised formalisms required by queueing network analysis and Markovian numerical solution techniques.
  - Both stochastic Petri nets and stochastic process algebras are attempts to move closer to a designer's world view. Unfortunately, both remain rather academic.
Explicit efforts to appeal to software engineers and designers include

- Smith's Software Performance Engineering outlined a methodology for integration of performance estimation within the software engineering process,
- Beilner's HIT introduced a notation based on abstract data types and layered design,
  - from which performance estimates could be derived automatically,
  - using a variety of solution techniques
- Hughes introduced a notion of hierarchical decomposition of measures, related to hierarchical design methods.

The performance engineering roadmap
Design of software systems

- Software design has suffered itself from a lack of agreement on an approach and a notation which is best suited to the creation of effective applications. Fashions have moved on frequently, with functional, structured approaches being challenged by object oriented and latterly component based approaches. In some application domains formal proof has been seen as paramount, with languages such as LOTOS growing from formal concurrency research using process algebras. For the performance engineering researcher, this has meant a bewildering range of rapidly moving targets.

Progress

- We take an optimistic view of where software design is going, however.
- This is based on an analysis of design trends which sees convergence on three major, closely related concepts:
  
  Component based design
  
  Architectures and patterns
  
  UML
Component based design

- provides a high level abstraction of the building blocks used in system construction;
- allows interfaces and connections between units of the system to be identified in a manner similar to the hardware systems where performance analysis is already well accepted;
- allows investment in the understanding of component behaviour, since component reuse is a major part of this approach.

Architectures and patterns

- support the view of systems as communicating components and map easily onto the way performance models define systems;
- design uses communicating state machines;
- performance modelling uses communicating stochastic processes.
UML

- provides a widely accepted notation for a component based approach to design,
- will become more formalised as it becomes used in critical systems;
- along with a widely used notation come widely used computer assisted software engineering (CASE) tools.

UML version of a Harel Statechart
Integrating current design methods with performance engineering

- The first major meeting devoted exclusively to software performance was the 1st international Workshop on Software and Performance (WOSP) in Santa Fe in October 1998.
- This concluded that the definition of an integrated means of representing performance within widely used design notations, notably UML, was an essential prerequisite to wider adoption of SPE within the software community.
- The recent UML ’99 conference included papers on this issue.
Work in this area to date

• simulation of UML models,
• generation of queueing network models from UML deployment diagrams and collaboration diagrams.
• Mappings from UML to
  – layered queueing network models (LQNs),
  – stochastic Petri net models (SPNs)
  – stochastic process algebra models, particularly Hillston’s PEPA,
• All show the potential for using UML’s logical and behavioural notations to define the structure of performance models.

How to do it

• Figure shows a UML collaboration diagram for a simple two player boardgame and the statechart representing the internal behaviour of one class of object, Board, used to build this collaboration.
• Annotations to the statecharts expressing time are defined within UML today.
• Further developments in incorporating timing within message sequences and other external behaviour diagrams are being debated. This will take us to the point where we have performance CASE tools.
• That is today’s state of the art.
• Our problem is which way forward from there.
An agenda for progress

- There has never been a more optimistic point for expanding the use of performance engineering within the wider software engineering community.
- This will be accelerated by the emergence of much more elaborate systems for implementation of applications.
- The growing popularity of object broker architectures and World Wide Web based applications will force users to ask developers hard questions about response times and scalability.

Steps needed

- 1. To create a well understood formalism, probably based on UML, allowing performance annotations to design models.
- 2. To create a methodology which embeds performance questions within the software lifecycle in terms of widely used approaches.
- 3. To integrate solution tools for performance measures transparently within extended design tools, such as object oriented CASE tools.
- 4. To develop ways of returning performance results from specialised tools in terms of the design models from which they were derived.
- 5. To integrate performance modelling measures within a performance monitoring and testing framework in a consistent manner.