Software performance design environments

Stephen Gilmore
Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland

Joint work with Catherine Canevet, Jane Hillston and Perdita Stevens

Structure of this talk

- Modelling with UML and PEPA
- UML state diagrams
- Describing performance measures in UML
- Software architecture
- An example: extracting, solving, reflecting
- Related work
- Conclusions

Introduction

- The *Unified Modelling Language* (UML) is an effective and popular notation which is used to capture high-level designs for software systems.

- One aspect of software system design which is not typically captured in a UML document is a record of the likely (or desired) rate of *performance* of the major activities of the system.

- When this information is not included in the initial UML description of a system it increases the likelihood that the performance of the software system being developed will not be considered until very late in the software development process.

Motivation

- Our motivation for working with UML in the performance modelling process is to *introduce the benefits of performance analysis* without the complexities and conceptual challenges which are normally associated with formal description techniques such as process algebras.

- The UML modeller can compose models and solve these for performance results without needing to understand the PEPA language, its formal definition or even how their model is represented in PEPA.

Aims

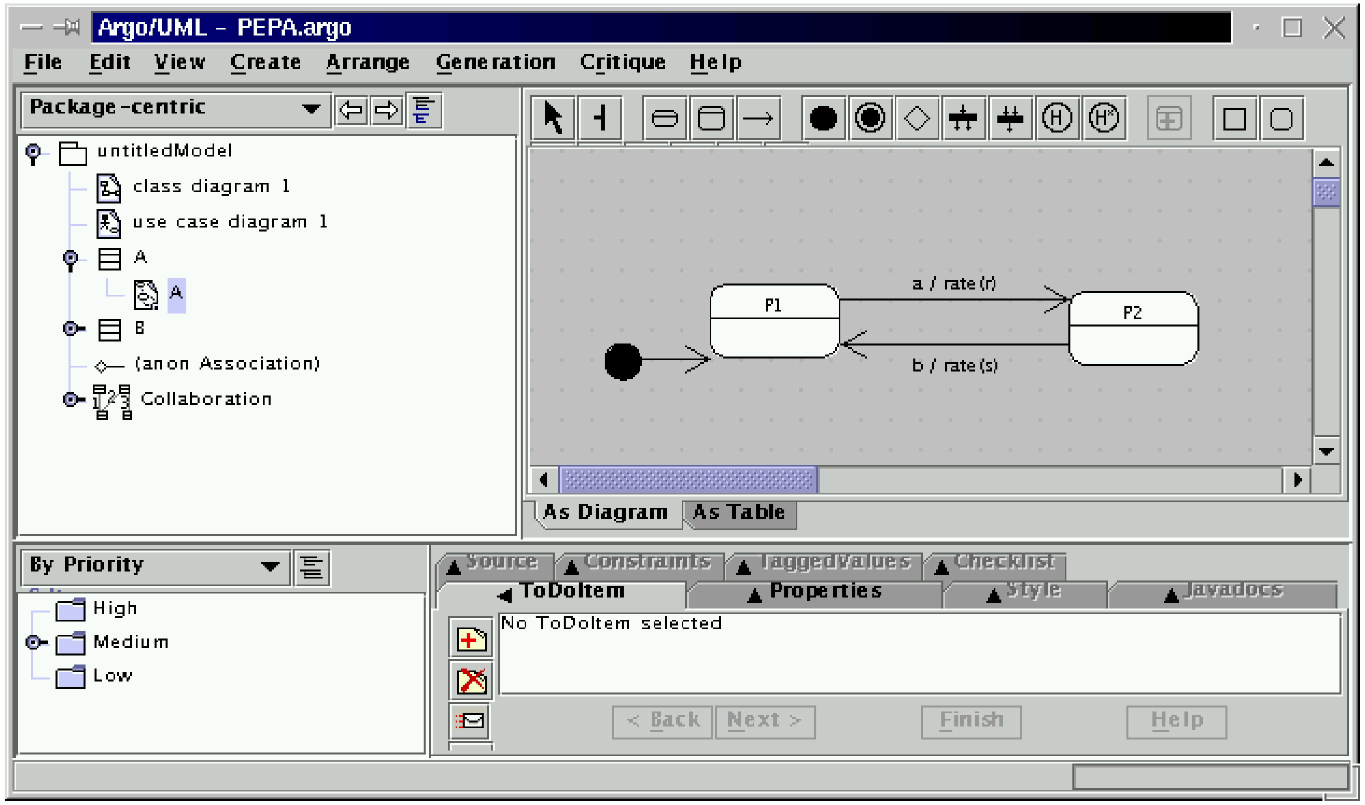
- We aim to achieve two goals in this work:

  1. to show how UML models enhanced with performance information can be mapped into PEPA; and

  2. to show how the results of the performance analysis of the PEPA models which are produced by this process can be presented to the UML modeller in the terms of their UML model.

- We concentrate primarily on one type of UML diagram here: *state diagrams*.

State diagrams in ArgoUML



No ToDoItem selected

Problems in specifying performance measures

- Performance measures can be described by building a *reward structure* on the model.
  - However, associating locations in the equilibrium probability distribution with syntactic states of the model *exposes details of the representation* such as orderings of components in the PEPA system description.
  - This would present the UML modeller with *much of the difficulty* of working directly with Markov Chains.
- Higher-level description languages for specifying performance measures exist, such as PML_μ and *CSL*, but these notations would be formidable for a typical UML developer to use.

Specifying performance measures simply

- For this reason we *aggregate* the state space of the system over the *local states of each component*. This has two beneficial effects:

  1. it avoids the need for any descriptions of state-space representations, whether high-level or low-level; and

  2. instead of working with a large set of very small numbers the modeller works with a small set of numbers which are orders of magnitude larger.

- We can express more complex performance measures by *defining UML components* which expose the configurations of interest in the model. Behaviourally, such components may be redundant, but they are necessary for expressing performance measures.

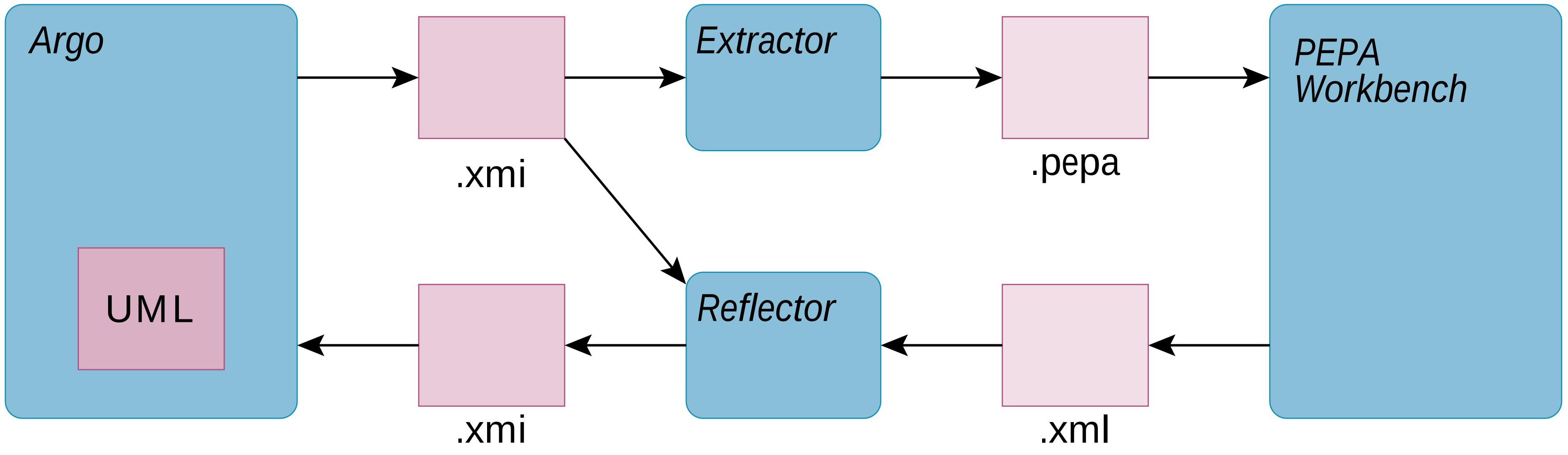
Example

- A component such as the following might be redundant in the model considered as a whole but *utilisation* can be directly read from the model solution as the percentage of time that the component spends in the *Busy* state.

$$
\begin{array}{l}
/\!*\ \text{An idle resource can be acquired}\ *\!/ \\
\mathit{Idle} \overset{\mathit{def}}{=} (\mathit{acquire}, \top).\mathit{Busy} \\
\\
/\!*\ \text{A busy resource can be released}\ *\!/ \\
\mathit{Busy} \overset{\mathit{def}}{=} (\mathit{release}, \top).\mathit{Idle}
\end{array}
$$

- We present this example in PEPA notation here but the modeller working with our method would describe this using the equivalent UML state diagram.

Software architecture



The PEPA Workbench

- The PEPA stochastic process algebra is supported by a range of tools including the *PEPA Workbench*, the *Möbius Modeling Framework* and the *PRISM* probabilistic symbolic model checker.

- We adapted the Java edition of the PEPA Workbench to interoperate with our *Extractor* and *Reflector* tools. The PEPA Workbench processes an input PEPA model which the Extractor generates from an input UML model in XMI format. It writes its results as an XML document which is processed by the Reflector tool.

- It would be impractical to use the PEPA Workbench user interface together with ArgoUML so we added a command-line interface to the PEPA Workbench, allowing it to be invoked from a script.

Extractor

- The Extractor takes as its parameter the original XMI file generated when saving a UML model with ArgoUML and returns an equivalent PEPA input. We use the *Minidom XML parser* to parse the XMI file. Once we have the XMI as a DOM object, we can access individual tags by name.

- When processing the XMI file we look for all the elements that we will need to provide in the PEPA model which we produce as output (`statemachine`, `context`, `transition`, `collaboration`, `base`, . . .).
  - We print for each state machine a defining PEPA expression of the following form: `#source=(workload,rate).target+...` For each state, if it is a source, we find its transitions with the correct workload, rate and target.
  - Finally we print the collaboration line (e.g. `P1 <a,b> P3`).

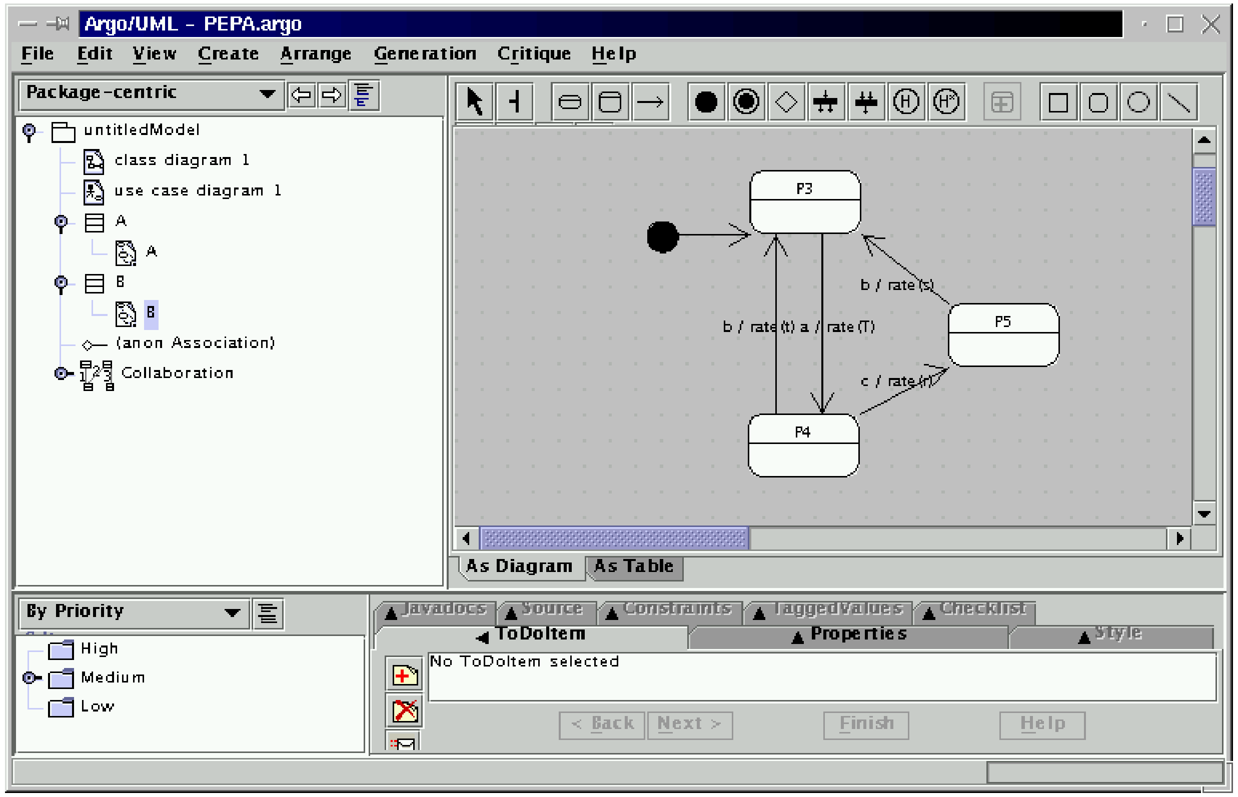
Reflector

- The Reflector takes as its parameter the original XMI file and the XML file that contains all the results from the PEPA Workbench. It returns a modified XMI file, which when loaded in ArgoUML will show the modified model.

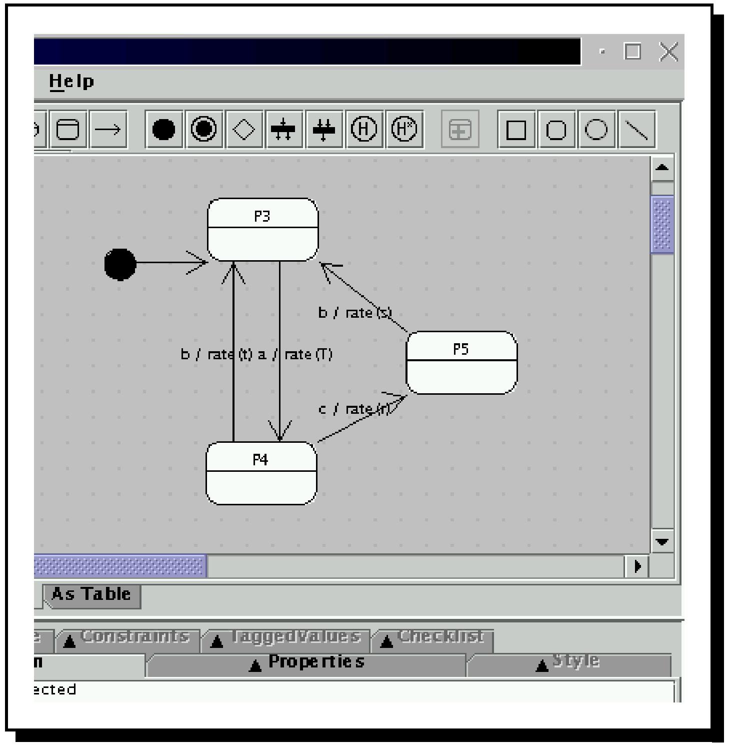
- We use Minidom to parse the original XMI file and also to build the modified XMI file by creating the branches and leaves of the document. The `xml.dom.minidom` module supports a very simple interface for adding new XML tags and data to an XML document.

- We modify our original XMI file using the results from the PEPA Workbench where for each `statemachine`, for each `state`, we add the corresponding probability.

A simple example (before)



A simple example (extracting)



```
%r = 2.0;
%s = 2.0;
%t = 2.0;

#P4 = (c,r).P5 + (b,t).P3;
#P5 = (b,s).P3;
#P3 = (a,infty).P4;
#P1 = (a,r).P2;
#P2 = (b,s).P1;

P1 < a,b > P3
```

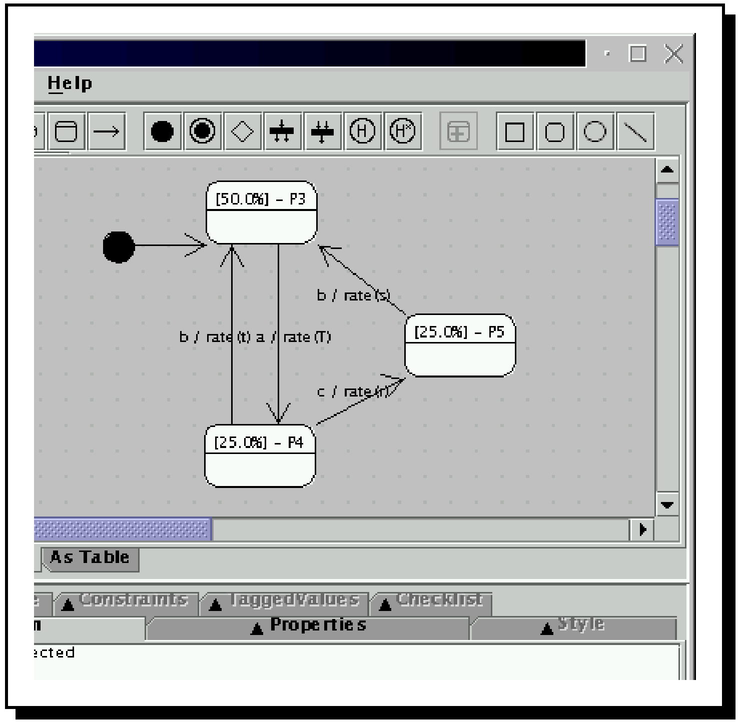
A simple example (solving)

```
%r = 2.0;
%s = 2.0;
%t = 2.0;

#P4 = (c,r).P5 + (b,t).P3;
#P5 = (b,s).P3;
#P3 = (a,infty).P4;
#P1 = (a,r).P2;
#P2 = (b,s).P1;

P1 < a,b > P3
```

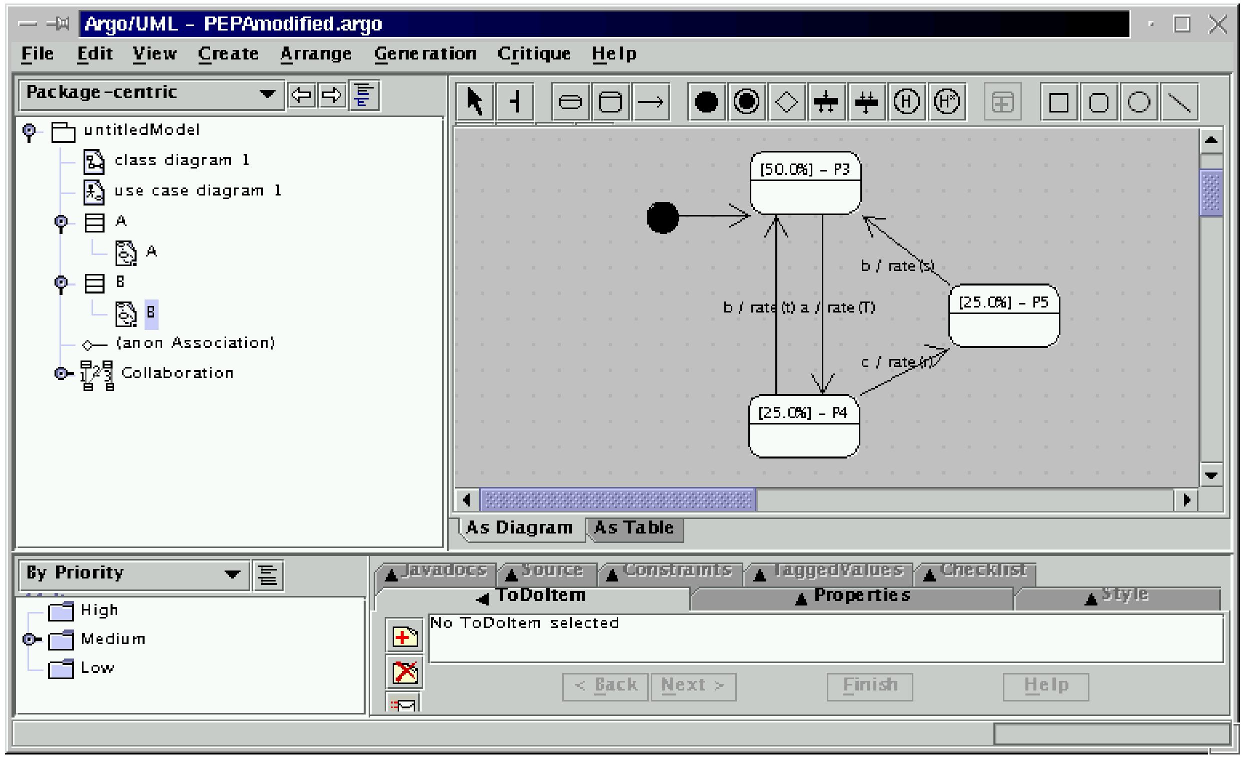
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE PEPA_Workbench_Results SYSTEM "PepaResults.dtd">
<PEPA_Workbench_Results>
  <Component Name="0">
     <State Name="P1">
       <Probability>0.5</Probability>
     </State>
     <State Name="P2">
       <Probability>0.5</Probability>
     </State>
   </Component>
   <Component Name="1">
     <State Name="P5">
       <Probability>0.25</Probability>
     </State>
     <State Name="P3">
       <Probability>0.5</Probability>
     </State>
     <State Name="P4">
       <Probability>0.25</Probability>
     </State>
   </Component>
</PEPA_Workbench_Results>
```

A simple example (reflecting)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE PEPA_Workbench_Results SYSTEM "PepaResults.dtd">
<PEPA_Workbench_Results>
  <Component Name="0">
     <State Name="P1">
       <Probability>0.5</Probability>
     </State>
     <State Name="P2">
       <Probability>0.5</Probability>
     </State>
   </Component>
   <Component Name="1">
     <State Name="P5">
       <Probability>0.25</Probability>
     </State>
     <State Name="P3">
       <Probability>0.5</Probability>
     </State>
     <State Name="P4">
       <Probability>0.25</Probability>
     </State>
   </Component>
</PEPA_Workbench_Results>
```



A simple example (after)



No ToDoItem selected

Related work

- Pooley previously discussed generating PEPA models from a combination of sequence diagrams, collaboration diagrams, and a combined collaboration/state diagram.

- Mitton and Holton undertake an alternative mapping between PEPA and UML statecharts.

- Thomas, Munro, King and Pooley combine PEPA models with graphical notations for visualising derivation graphs, component interfaces and other aspects of the system under study.

- Petriu and Shen which maps UML models via XMI into layered queueing network (LQN) performance models.

Conclusions

- We have presented a method of deriving performance information from UML models.
- The method is supported by a tool set which comprises some existing modelling tools (ArgoUML and the PEPA Workbench) and other translators which we have written to connect them (the Extractor and the Reflector).
- The translators which interconnect the applications are general-purpose and can be adapted to work with other modelling tools.
- We have applied our approach to a range of small examples. We plan to investigate its usefulness when applied to larger examples.

Acknowledgements

This work has been supported by the *DEGAS (Design Environments for Global ApplicationS)* project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing.