A unified approach to performance modelling and verification

Stephen Gilmore and Leïla Kloul
Laboratory for Foundations of Computer Science
The University of Edinburgh

Motivation

- The need for safe and dependable computer systems is well-understood. In some systems correct functioning depends upon the ability to *perform effectively under heavy workload*. The analysis of such systems must consider both timing and behavioural information.

Motivation

- The need for safe and dependable computer systems is well-understood. In some systems correct functioning depends upon the ability to *perform effectively under heavy workload*. The analysis of such systems must consider both timing and behavioural information.

- *Performability = performance + dependability.*

Motivation

- The need for safe and dependable computer systems is well-understood. In some systems correct functioning depends upon the ability to *perform effectively under heavy workload*. The analysis of such systems must consider both timing and behavioural information.

- *Performability = performance + dependability.*

- It is better to know about problems early. If performance design flaws are found early in the development process then they can be corrected at a *relatively low cost*. In contrast, if they are found after the development process is long underway then they may be *expensive or even unrealistic* to repair.

Summary of this talk

- We describe a novel performability modelling approach which facilitates the efficient, and simple, solution of performance models extracted from high-level descriptions of systems.

Summary of this talk

- We describe a novel performability modelling approach which facilitates the efficient, and simple, solution of performance models extracted from high-level descriptions of systems.

- The notation which we use for our high-level designs is the *UML* graphical modelling language.

Summary of this talk

- We describe a novel performability modelling approach which facilitates the efficient, and simple, solution of performance models extracted from high-level descriptions of systems.

- The notation which we use for our high-level designs is the *UML* graphical modelling language.

- The technology which provides the efficient representation capability for the underlying performance model is the Multi-Terminal Binary Decision Diagram-based *PRISM* probabilistic model checker.

Summary of this talk

- We describe a novel performability modelling approach which facilitates the efficient, and simple, solution of performance models extracted from high-level descriptions of systems.

- The notation which we use for our high-level designs is the *UML* graphical modelling language.

- The technology which provides the efficient representation capability for the underlying performance model is the Multi-Terminal Binary Decision Diagram-based *PRISM* probabilistic model checker.

- The UML models are compiled through an intermediate language, the stochastic process algebra *PEPA*, before translation into MTBDDs for solution.

Contribution

- We provide a structured performability modelling platform by connecting a *specification environment* (SENV) and a *verification environment* (VENV) so that each may communicate with the other.

Contribution

- We provide a structured performability modelling platform by connecting a *specification environment* (SENV) and a *verification environment* (VENV) so that each may communicate with the other.

- The SENV and VENV are connected by a bridge which consists of two categories of software tools. These are:
  - *extractors* which translate designs from the SENV into inputs for the VENV, omitting any aspects of the design which are not relevant for the verification task at hand; and
  - *reflectors* which convert the results from the analysis performed by the VENV back into a form which can be processed and displayed by the SENV.

UML modelling

- A UML *model* is represented by a collection of diagrams describing parts of the system from different points of view; there are seven main diagram types. For example, there will typically be a *static structure diagram* (or *class diagram*) describing the classes and interfaces in the system and their static relationships (inheritance, dependency, etc.).

UML modelling

- A UML *model* is represented by a collection of diagrams describing parts of the system from different points of view; there are seven main diagram types. For example, there will typically be a *static structure diagram* (or *class diagram*) describing the classes and interfaces in the system and their static relationships (inheritance, dependency, etc.).
- *State diagrams*, a variant on Harel state charts, can be used to record the dynamic behaviour of particular classes. Interaction diagrams, such as sequence diagrams, are used to illustrate the way objects of different classes interact in a particular scenario.

UML modelling

- A UML *model* is represented by a collection of diagrams describing parts of the system from different points of view; there are seven main diagram types. For example, there will typically be a *static structure diagram* (or *class diagram*) describing the classes and interfaces in the system and their static relationships (inheritance, dependency, etc.).

- *State diagrams*, a variant on Harel state charts, can be used to record the dynamic behaviour of particular classes. Interaction diagrams, such as sequence diagrams, are used to illustrate the way objects of different classes interact in a particular scenario.

- As usual we expect that the UML modeller will make a number of diagrams of different kinds. Our analysis is based on *state and collaboration diagrams*.

Introducing performance information

- We have introduced performance information in the state diagrams such that each transition in these diagrams is labelled with a pair 'a / r' where a is the action type executed and r is an exponentially distributed rate associated with this action.

Introducing performance information

- We have introduced performance information in the state diagrams such that each transition in these diagrams is labelled with a pair '*a* / *r*' where *a* is the action type executed and *r* is an exponentially distributed rate associated with this action.
  - A customer arrival causes a change in the state of a queue so this would be one example of an action. Concretely, *arrive* / λ and *serve* / μ would be suitable arc adornments for a state diagram for a queue.

Introducing performance information

- We have introduced performance information in the state diagrams such that each transition in these diagrams is labelled with a pair '*a* / *r*' where *a* is the action type executed and *r* is an exponentially distributed rate associated with this action.
  - A customer arrival causes a change in the state of a queue so this would be one example of an action. Concretely, *arrive* / λ and *serve* / μ would be suitable arc adornments for a state diagram for a queue.

- State machines are *sequential components*. The collaboration diagram specifies the *concurrent composition* of instances of these state machines. Collaborating state machines synchronise on all of their common action types. Analysing such a UML model begins by mapping it to a model in the *PEPA stochastic process algebra*.

Analysing PEPA models

- PEPA is a process algebra with timed activities, choice, parallel composition and hiding. The analysis of a PEPA model derives a *Continuous-Time Markov Chain (CTMC)*. Many analysis tools are available for PEPA.

Analysing PEPA models

- PEPA is a process algebra with timed activities, choice, parallel composition and hiding. The analysis of a PEPA model derives a *Continuous-Time Markov Chain (CTMC)*. Many analysis tools are available for PEPA.

- **The PEPA Workbench** — *steady-state* and *transient* analysis
  - ▷ *MTBF* and *MTTF* computation

Analysing PEPA models

- PEPA is a process algebra with timed activities, choice, parallel composition and hiding. The analysis of a PEPA model derives a *Continuous-Time Markov Chain (CTMC)*. Many analysis tools are available for PEPA.
- **The PEPA Workbench** — *steady-state* and *transient* analysis
  - ▷ *MTBF* and *MTTF* computation
- **Möbius** — *steady-state*, *transient* analysis and *simulation*
  - ▷ *Instant-of-time* and *interval-of-time* measures

Analysing PEPA models

- PEPA is a process algebra with timed activities, choice, parallel composition and hiding. The analysis of a PEPA model derives a *Continuous-Time Markov Chain (CTMC)*. Many analysis tools are available for PEPA.
- **The PEPA Workbench** — *steady-state* and *transient* analysis
  - ▷ *MTBF* and *MTTF* computation
- **Möbius** — *steady-state*, *transient* analysis and *simulation*
  - ▷ *Instant-of-time* and *interval-of-time* measures
- **PRISM** — *steady-state*, *transient* analysis and *model-checking*
  - ▷ *custom performability verification*

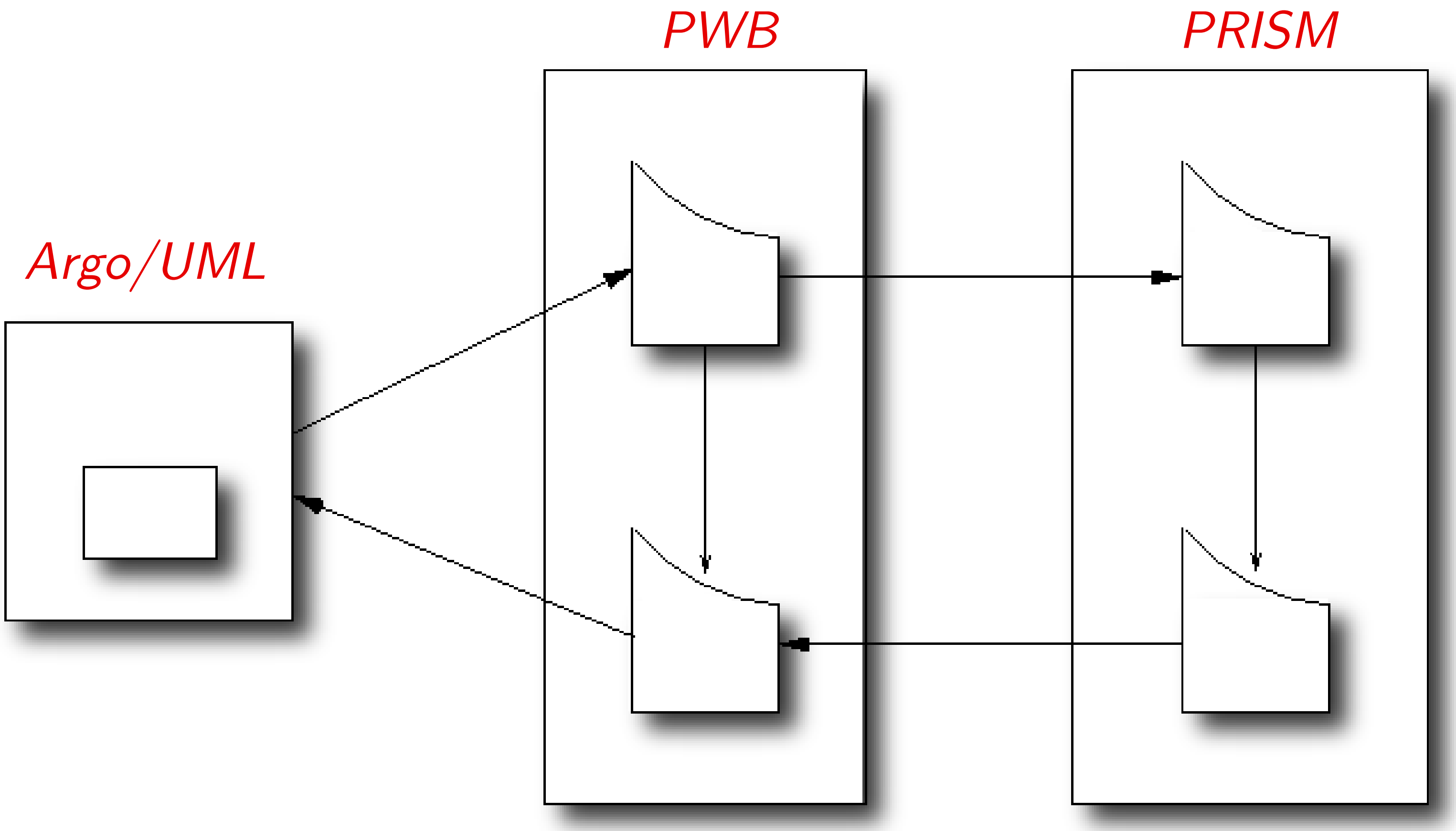
Analysing PEPA models

- PEPA is a process algebra with timed activities, choice, parallel composition and hiding. The analysis of a PEPA model derives a *Continuous-Time Markov Chain (CTMC)*. Many analysis tools are available for PEPA.
- **The PEPA Workbench** — *steady-state* and *transient* analysis
  - ▷ *MTBF* and *MTTF* computation
- **Möbius** — *steady-state*, *transient* analysis and *simulation*
  - ▷ *Instant-of-time* and *interval-of-time* measures
- **PRISM** — *steady-state*, *transient* analysis and *model-checking*
  - ▷ *custom performability verification*
- **IPC/DNAmaca** — *steady-state*, *transient* analysis and *passage-time densities*
  - ▷ *service-level agreements*

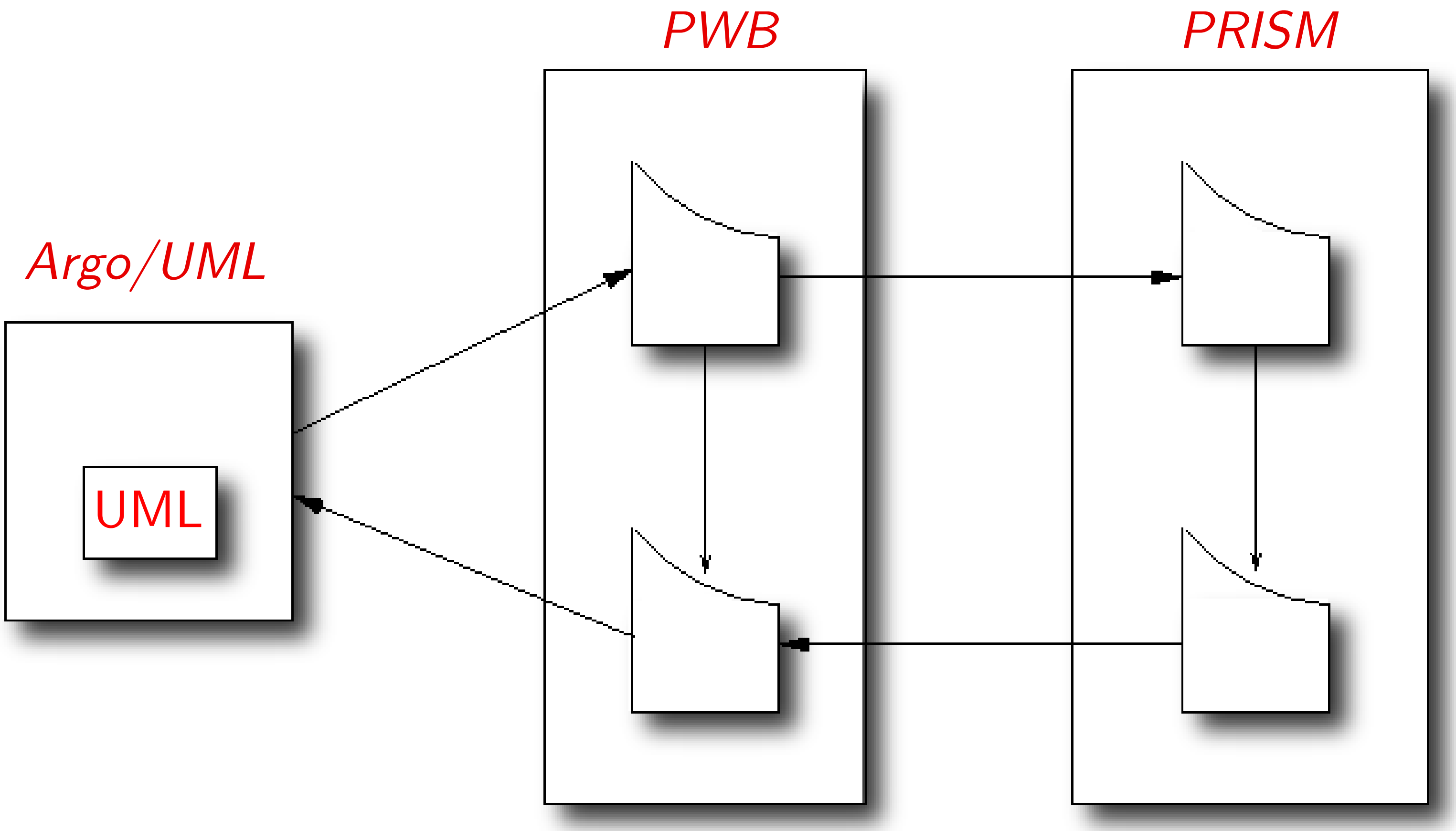
Analysing PEPA models

- PEPA is a process algebra with timed activities, choice, parallel composition and hiding. The analysis of a PEPA model derives a *Continuous-Time Markov Chain (CTMC)*. Many analysis tools are available for PEPA.
- **The PEPA Workbench** — *steady-state* and *transient* analysis
  - ▷ *MTBF* and *MTTF* computation
- **Möbius** — *steady-state*, *transient* analysis and *simulation*
  - ▷ *Instant-of-time* and *interval-of-time* measures
- **PRISM** — *steady-state*, *transient* analysis and *model-checking*
  - ▷ *custom performability verification*
- **IPC/DNAmaca** — *steady-state*, *transient* analysis and *passage-time densities*
  - ▷ *service-level agreements*

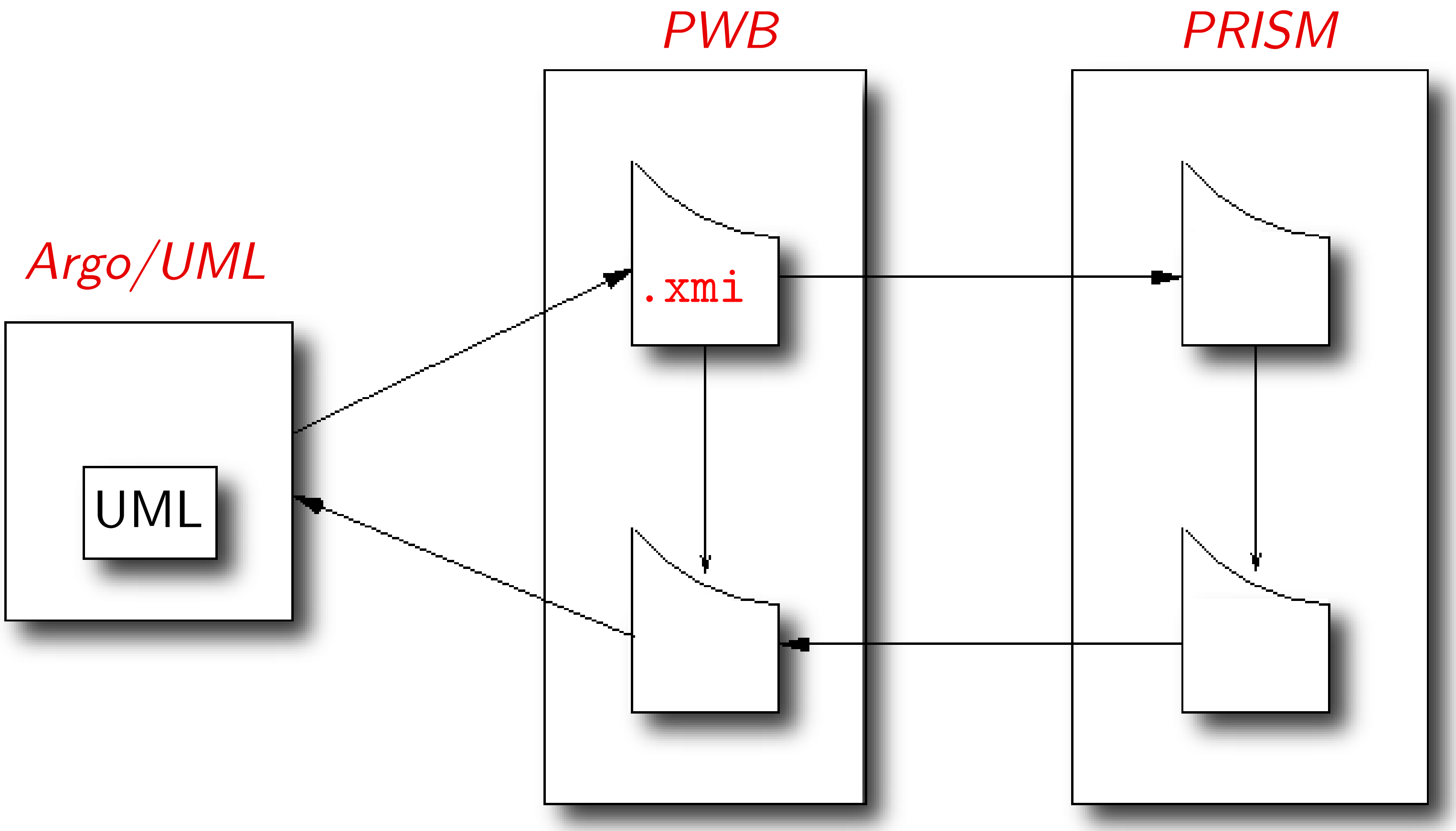
Software architecture



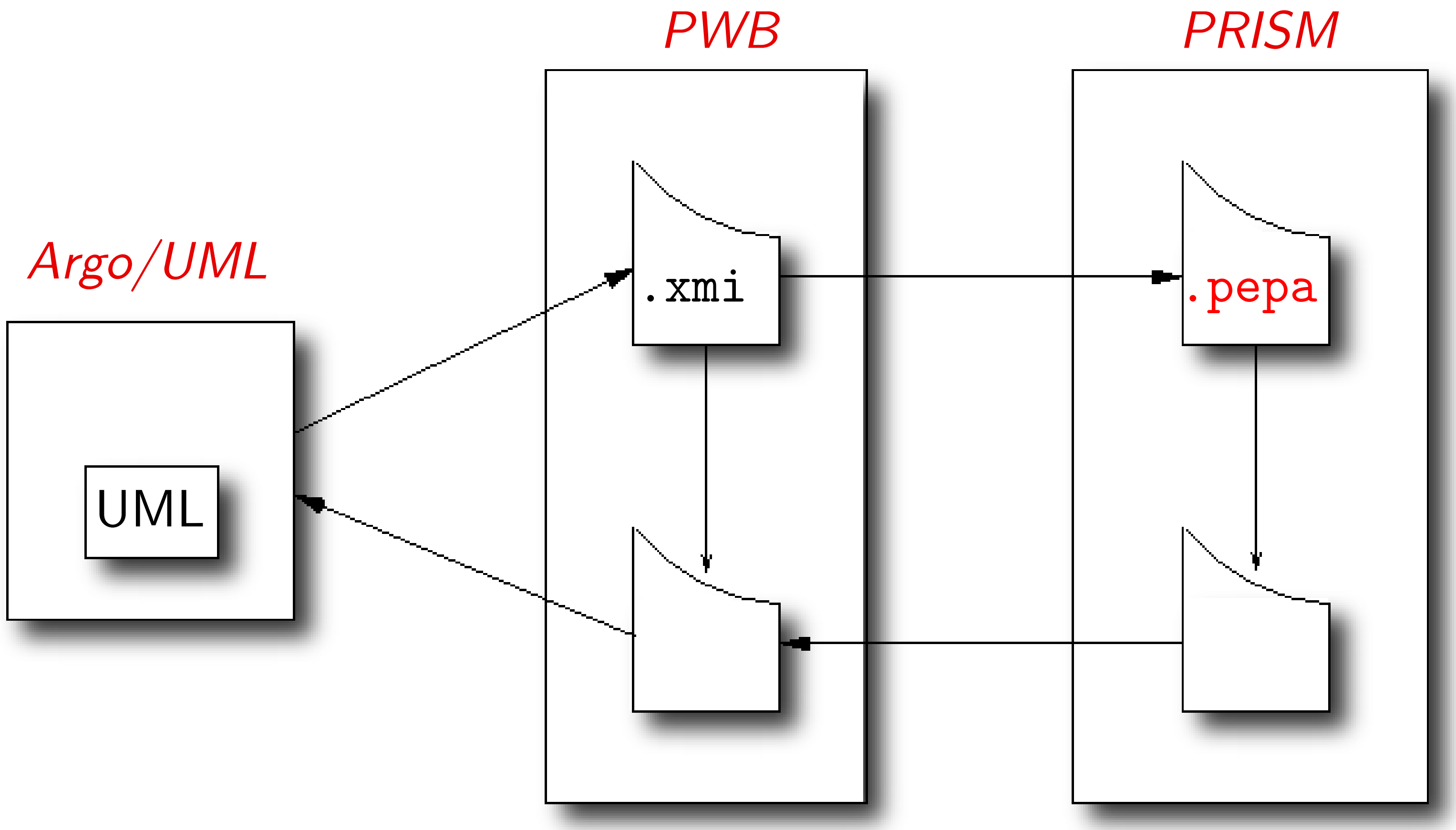
Software architecture



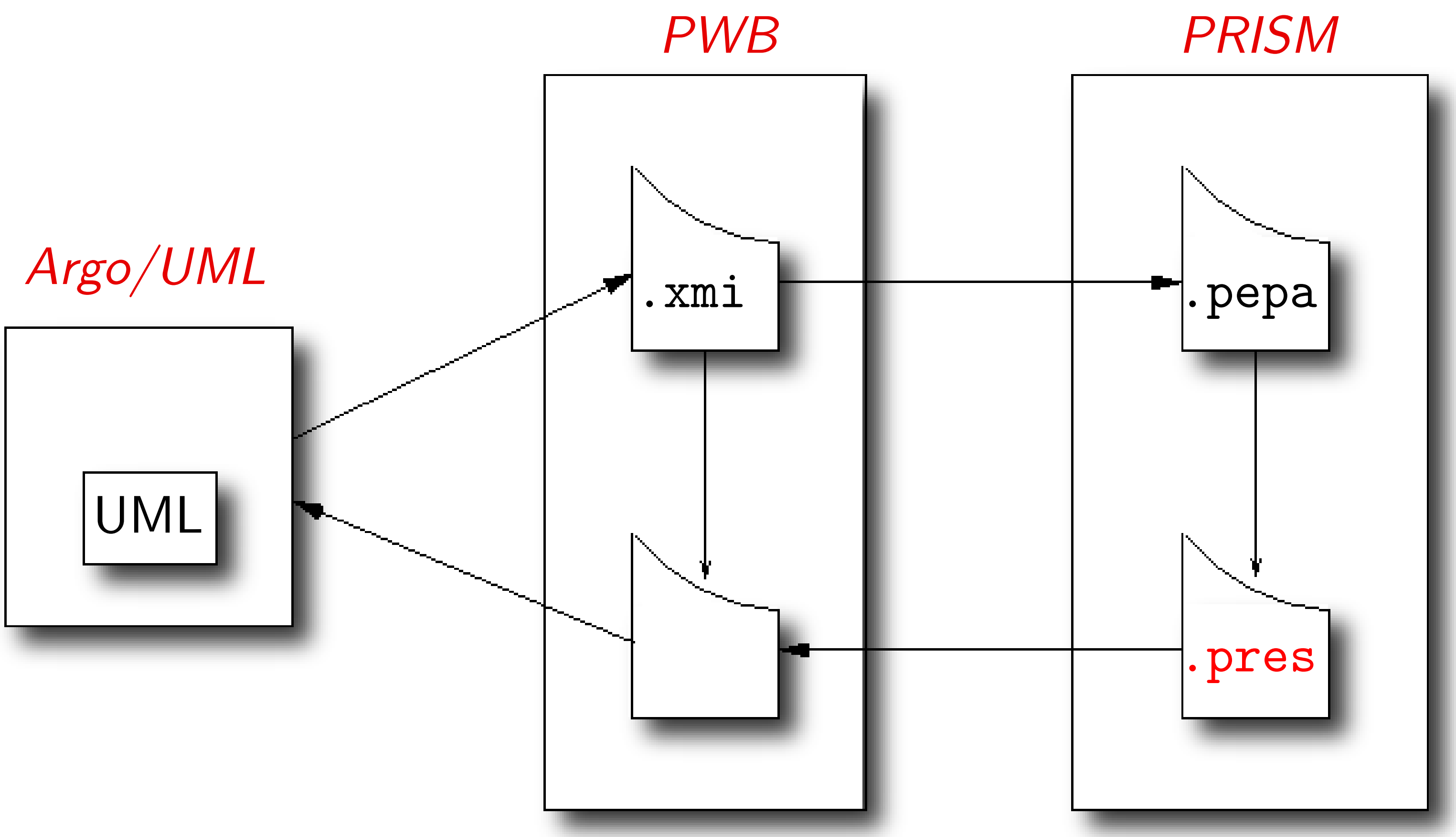
Software architecture



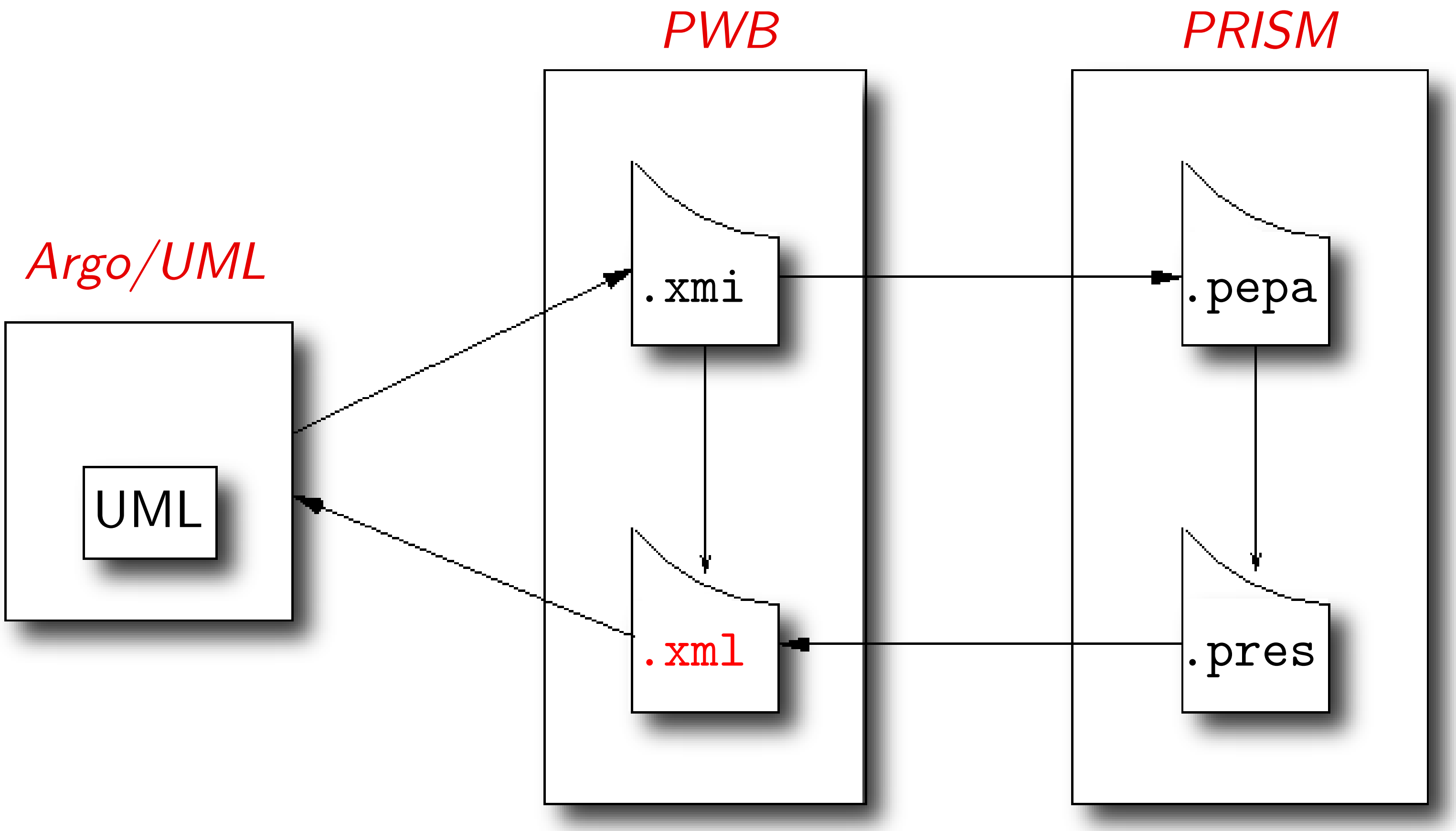
Software architecture



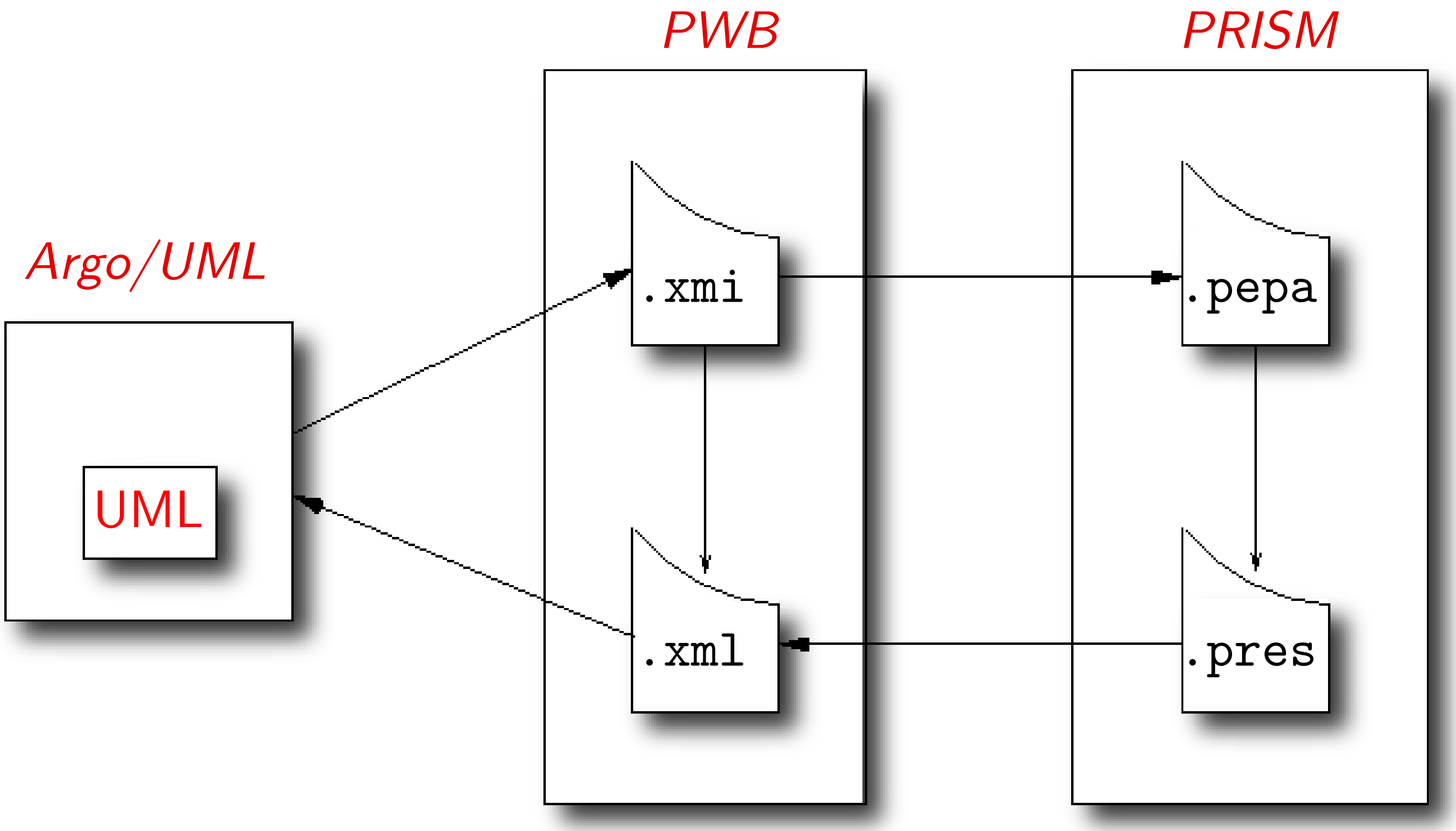
Software architecture



Software architecture



Software architecture



Specifying system metrics

- A timed UML model describes the behaviour of the system under study. Metrics over the system can be specified using additional system components or logical formulae.

Specifying system metrics

- A timed UML model describes the behaviour of the system under study. Metrics over the system can be specified using additional system components or logical formulae.

- A long-run performance measure can be specified by using *stochastic probes* to capture the property of interest. The probe can be described as simply another UML state machine diagram or using a special-purpose description language due to Bradley and Argent-Katwala.

Specifying system metrics

- A timed UML model describes the behaviour of the system under study. Metrics over the system can be specified using additional system components or logical formulae.

- A long-run performance measure can be specified by using *stochastic probes* to capture the property of interest. The probe can be described as simply another UML state machine diagram or using a special-purpose description language due to Bradley and Argent-Katwala.

- More complex metrics use *Continuous Stochastic Logic (CSL)*.

Specifying system metrics

- A timed UML model describes the behaviour of the system under study. Metrics over the system can be specified using additional system components or logical formulae.

- A long-run performance measure can be specified by using *stochastic probes* to capture the property of interest. The probe can be described as simply another UML state machine diagram or using a special-purpose description language due to Bradley and Argent-Katwala.

- More complex metrics use *Continuous Stochastic Logic (CSL)*.

$$\phi \quad ::= \quad \mathit{true} \mid \mathit{false} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi]$$

$$\psi \quad ::= \quad \mathrm{X}\,\phi \mid \phi\,\mathrm{U}\,\phi \mid \phi\,\mathrm{U}^{I}\,\phi$$

Specifying system metrics

- A timed UML model describes the behaviour of the system under study. Metrics over the system can be specified using additional system components or logical formulae.

- A long-run performance measure can be specified by using *stochastic probes* to capture the property of interest. The probe can be described as simply another UML state machine diagram or using a special-purpose description language due to Bradley and Argent-Katwala.

- More complex metrics use *Continuous Stochastic Logic (CSL)*.

$$\phi \;::=\; true \mid false \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi]$$

$$\psi \;::=\; \mathrm{X}\,\phi \mid \phi\,\mathrm{U}\,\phi \mid \phi\,\mathrm{U}^{I}\,\phi$$

Specifying system metrics

- A timed UML model describes the behaviour of the system under study. Metrics over the system can be specified using additional system components or logical formulae.

- A long-run performance measure can be specified by using *stochastic probes* to capture the property of interest. The probe can be described as simply another UML state machine diagram or using a special-purpose description language due to Bradley and Argent-Katwala.

- More complex metrics use *Continuous Stochastic Logic (CSL)*.

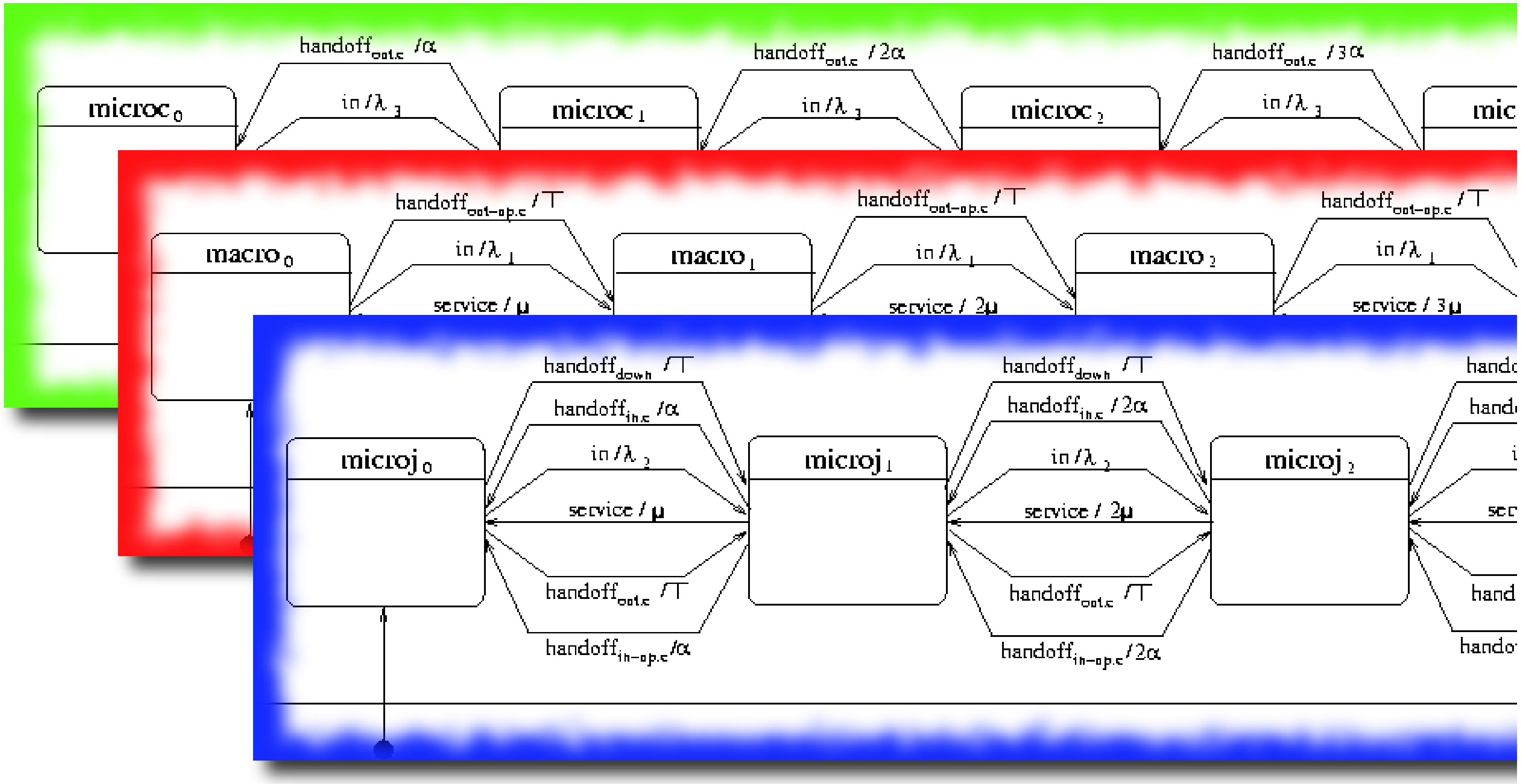
$$\phi \quad ::= \quad \mathit{true} \mid \mathit{false} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi]$$

$$\psi \quad ::= \quad \mathrm{X}\,\phi \mid \phi\,\mathrm{U}\,\phi \mid \phi\,\mathrm{U}^{I}\,\phi$$

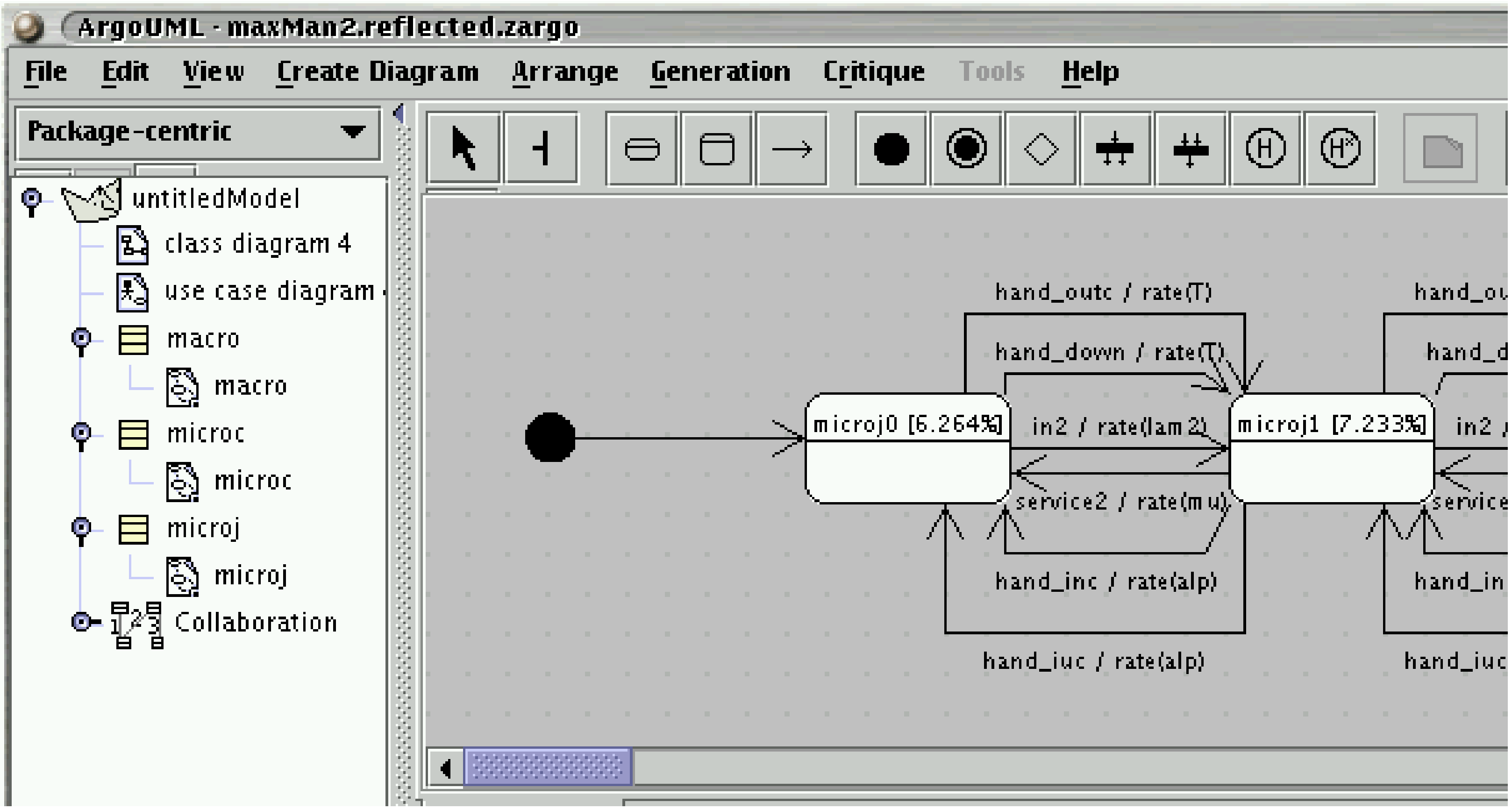
Example: hierarchical cellular network

- We conducted a case study with the tool to investigate its use in practical modelling.
- We modelled a *hierarchical cellular phone network* consisting of two tiers of cells, a level of macrocells overlying a level of microcells.
- In this study, we considered the *Manhattan model* where the reuse pattern is based on a five squared microcell cluster, a central cell surrounded by four peripheral cells.
- We considered a *Fixed Channel Allocation scheme* where a fixed number of channels are distributed among the different cells.

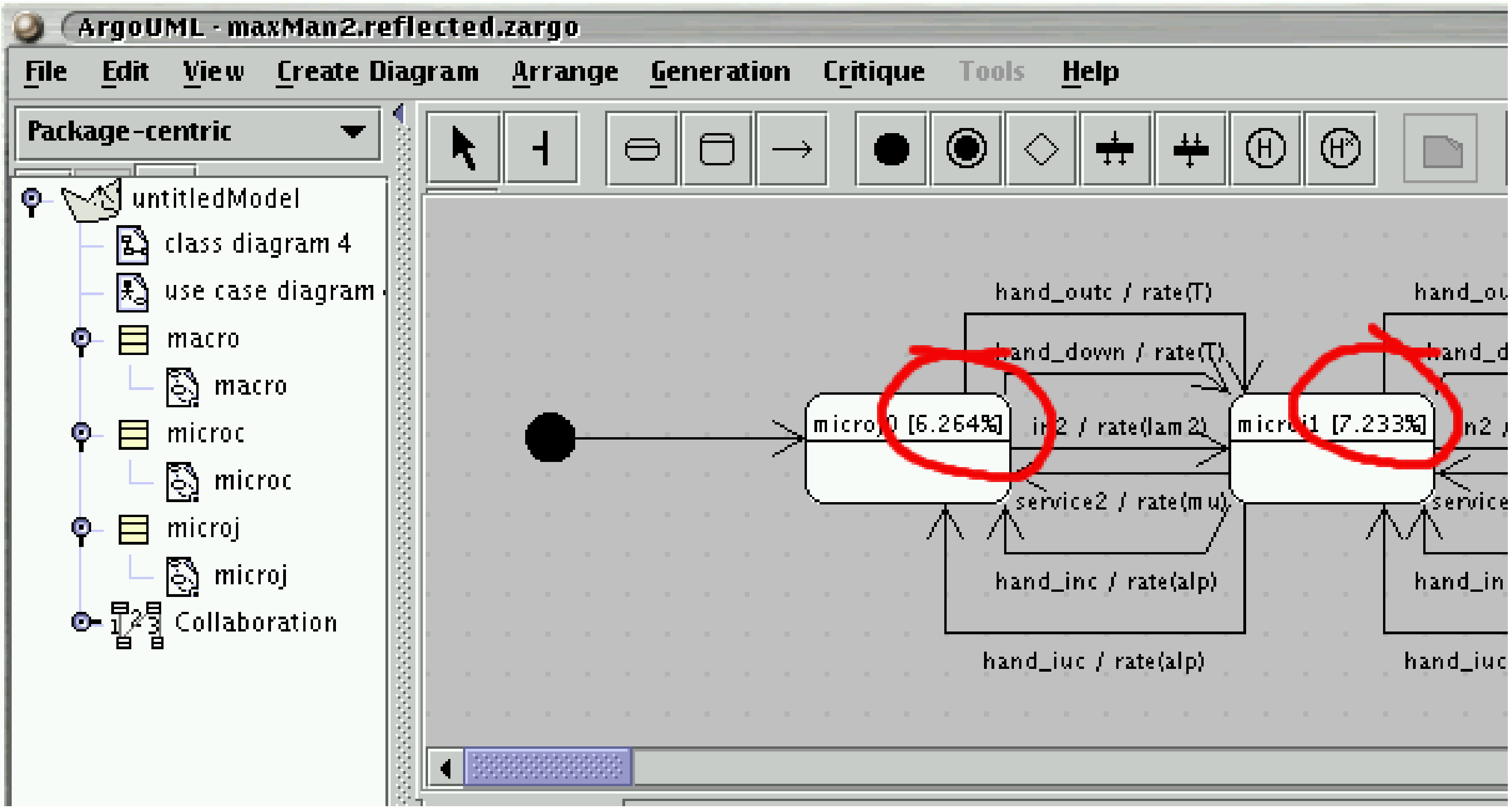
State diagrams



Results reflected in Argo/UML



Results reflected in Argo/UML



Related work

- Petriu and Shen automatically extract a *layered queueing network model* from an input UML model with performance annotations in the format specified by the *OMG Profile for Schedulability, Performance, and Time*.

Related work

- Petriu and Shen automatically extract a *layered queueing network model* from an input UML model with performance annotations in the format specified by the *OMG Profile for Schedulability, Performance, and Time*.
- López-Grao, Merseguer and Campos map UML diagrams into *Generalised stochastic Petri nets* which can be solved by *GreatSPN*.

Related work

- Petriu and Shen automatically extract a *layered queueing network model* from an input UML model with performance annotations in the format specified by the *OMG Profile for Schedulability, Performance, and Time*.
- López-Grao, Merseguer and Campos map UML diagrams into *Generalised stochastic Petri nets* which can be solved by *GreatSPN*.
- Lindemann, Thümmler, Klemm, Lohmann, and Waldhorst map state and activity diagrams into *generalised semi-Markov processes* which can be solved by *DSPNexpress-NG*.

Related work

- Petriu and Shen automatically extract a *layered queueing network model* from an input UML model with performance annotations in the format specified by the *OMG Profile for Schedulability, Performance, and Time*.
- López-Grao, Merseguer and Campos map UML diagrams into *Generalised stochastic Petri nets* which can be solved by *GreatSPN*.
- Lindemann, Thümmler, Klemm, Lohmann, and Waldhorst map state and activity diagrams into *generalised semi-Markov processes* which can be solved by *DSPNexpress-NG*.
- One feature of our work which is distinctive from the above is the role of a *reflector* in the system to present the results of the performance evaluation back to the UML modeller in the terms of their input model.

Conclusions

- This approach to modelling allows the modeller to access a powerful and efficient solution technology without having to master the details of unfamiliar modelling languages such as process algebras and reactive modules. Our experience of using the PEPA and PRISM tools has been uniformly good.

Conclusions

- This approach to modelling allows the modeller to access a powerful and efficient solution technology without having to master the details of unfamiliar modelling languages such as process algebras and reactive modules. Our experience of using the PEPA and PRISM tools has been uniformly good.

- One of the decisions which we have had to take in this work was the choice of UML diagrams and metaphors to employ. In part our choice in this was restricted by the degree of support offered by our UML tool (ArgoUML).

Conclusions

- This approach to modelling allows the modeller to access a powerful and efficient solution technology without having to master the details of unfamiliar modelling languages such as process algebras and reactive modules. Our experience of using the PEPA and PRISM tools has been uniformly good.

- One of the decisions which we have had to take in this work was the choice of UML diagrams and metaphors to employ. In part our choice in this was restricted by the degree of support offered by our UML tool (ArgoUML).

- We hope that we have gone some way to providing automated support for computing simple performability measures and to circumventing an unnecessary notational hurdle if this was acting as an impediment to the understanding and uptake of modern performability analysis technology.

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.

The authors thank Gethin Norman and David Parker of the University of Birmingham for the implementation of the PEPA process algebra combinators in the PRISM model checker.