

Service-Level Agreements for Service-Oriented Computing

Stephen Gilmore
LFCS, Edinburgh

Joint work with Allan Clark and Mirco Tribastone

Outline

Service-oriented computing

Example: Virtual University

Conclusions

Outline

Service-oriented computing

Example: Virtual University

Conclusions

Service-oriented computing

- ▶ A modern approach to distributed computing.
- ▶ Applications are built by composing services.
- ▶ Services are replicated across a number of servers.
- ▶ Providers *publish* services in registries.
- ▶ Users *discover* services and *bind* to them.

Uncertainties in service-oriented computing

- ▶ We do not know which service instances will be used.
- ▶ The service instances have different performance characteristics.
- ▶ The service instances may have different functionality.
- ▶ Plus all of the usual problems of distributed systems. . .

Uncertainties in service-oriented computing

- ▶ We do not know which service instances will be used.
- ▶ The service instances have different performance characteristics.
- ▶ The service instances may have different functionality.
- ▶ Plus all of the usual problems of distributed systems. . .

. . . Break out the chocolate biscuits :-)

Analysis of service-oriented computing

- ▶ Put all possible descriptions of service behaviours together in one big model.

Analysis of service-oriented computing

- ▶ Put all possible descriptions of service behaviours together in one big model.
 - ▶ Hope your favourite large state-space method can cope. :-)

Analysis of service-oriented computing

- ▶ Put all possible descriptions of service behaviours together in one big model.
 - ▶ Hope your favourite large state-space method can cope. :-)
- ▶ Separate out service bindings into different cases. Analyse cases separately. Re-combine results.

Analysis of service-oriented computing

- ▶ Put all possible descriptions of service behaviours together in one big model.
 - ▶ Hope your favourite large state-space method can cope. :-)
- ▶ Separate out service bindings into different cases. Analyse cases separately. Re-combine results.
 - ▶ Scalable analysis of scalable systems. :-)

What we need

- ▶ A way of specifying uncertainty about durations.

What we need

- ▶ A way of specifying uncertainty about durations.
 - ▶ Use exponential distributions.

What we need

- ▶ A way of specifying uncertainty about durations.
 - ▶ Use exponential distributions.
- ▶ A way of specifying uncertainty about rate parameters.

What we need

- ▶ A way of specifying uncertainty about durations.
 - ▶ Use exponential distributions.
- ▶ A way of specifying uncertainty about rate parameters.
 - ▶ $r = \{0.1, 0.3, 0.5, 0.7\}$

What we need

- ▶ A way of specifying uncertainty about durations.
 - ▶ Use exponential distributions.
- ▶ A way of specifying uncertainty about rate parameters.
 - ▶ $r = \{0.1, 0.3, 0.5, 0.7\}$
- ▶ A way of specifying uncertainty about bindings.

What we need

- ▶ A way of specifying uncertainty about durations.
 - ▶ Use exponential distributions.
- ▶ A way of specifying uncertainty about rate parameters.
 - ▶ $r = \{0.1, 0.3, 0.5, 0.7\}$
- ▶ A way of specifying uncertainty about bindings.
 - ▶ $Server = \{UEDIN :: Server, UNIPI :: Server\}$

Languages and tools

- ▶ SRMC (Sensoria Reference Markovian Calculus)

Languages and tools

- ▶ SRMC (Sensoria Reference Markovian Calculus)
 - ▶ Use `srmc` (Sensoria Reference Markovian Compiler) to compile to PEPA

Languages and tools

- ▶ SRMC (Sensoria Reference Markovian Calculus)
 - ▶ Use `srmc` (Sensoria Reference Markovian Compiler) to compile to PEPA
- ▶ PEPA (Performance Evaluation Process Algebra)

Languages and tools

- ▶ SRMC (Sensoria Reference Markovian Calculus)
 - ▶ Use `srmc` (Sensoria Reference Markovian Compiler) to compile to PEPA
- ▶ PEPA (Performance Evaluation Process Algebra)
 - ▶ Use `ipc` (Imperial PEPA Compiler) to compile to Hydra

Languages and tools

- ▶ SRMC (Sensoria Reference Markovian Calculus)
 - ▶ Use `srmc` (Sensoria Reference Markovian Compiler) to compile to PEPA
- ▶ PEPA (Performance Evaluation Process Algebra)
 - ▶ Use `ipc` (Imperial PEPA Compiler) to compile to Hydra
- ▶ Hydra (HYpergraph-based Distributed Response-time Analyser)

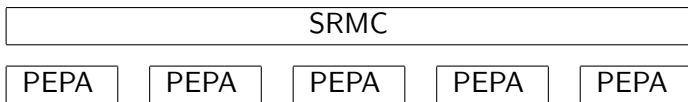
Languages and tools

- ▶ SRMC (Sensoria Reference Markovian Calculus)
 - ▶ Use `srmc` (Sensoria Reference Markovian Compiler) to compile to PEPA
- ▶ PEPA (Performance Evaluation Process Algebra)
 - ▶ Use `ipc` (Imperial PEPA Compiler) to compile to Hydra
- ▶ Hydra (HYpergraph-based Distributed Response-time Analyser)
 - ▶ Use `hydra-s` to compute response-time quantiles

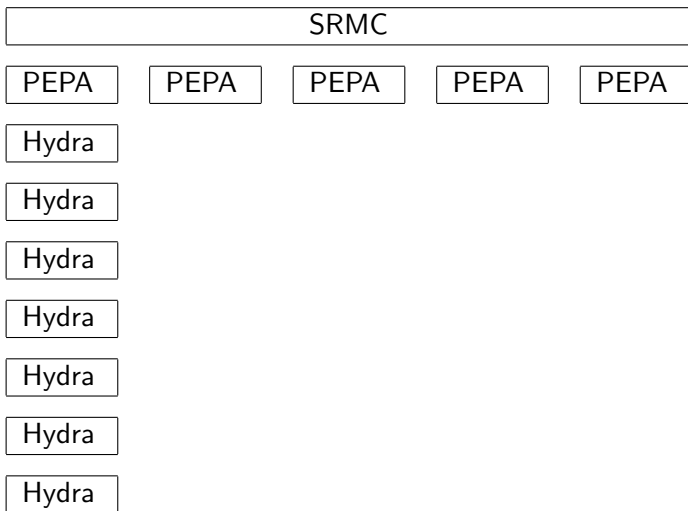
Evaluation model

SRMC

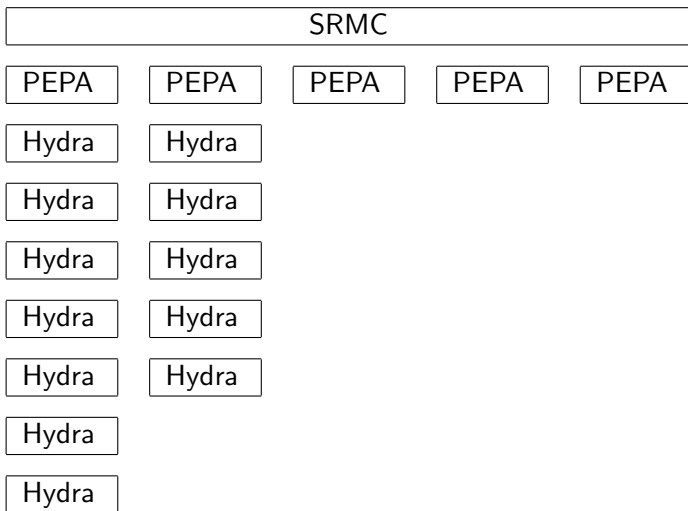
Evaluation model



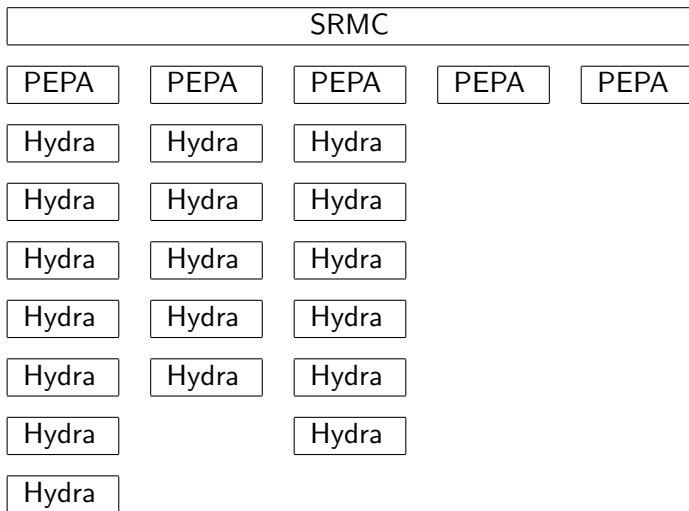
Evaluation model



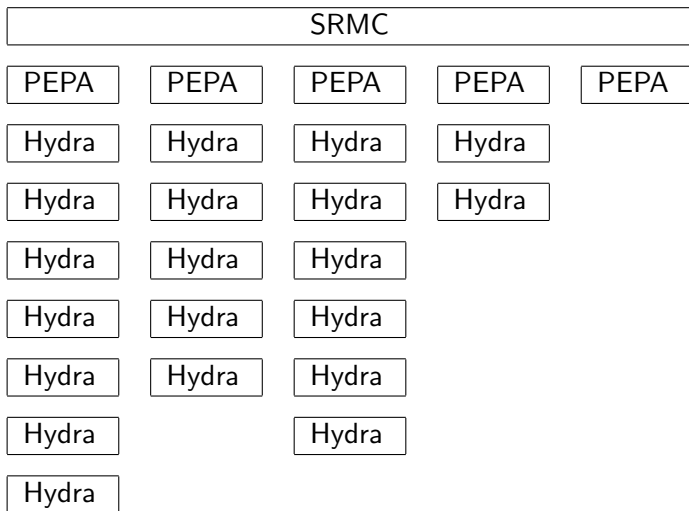
Evaluation model



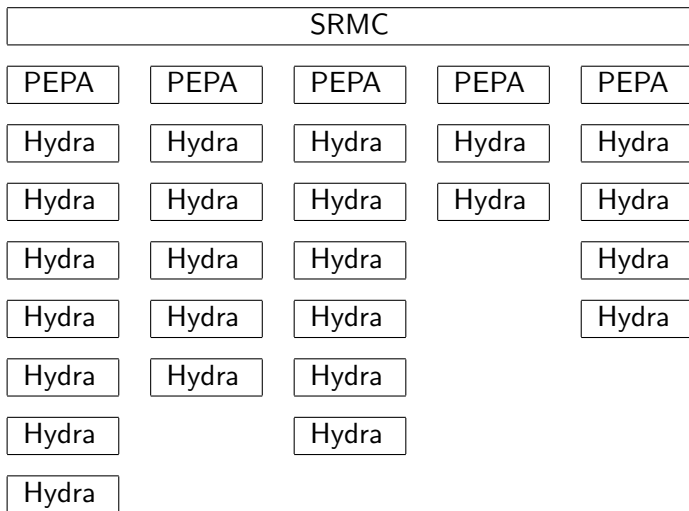
Evaluation model



Evaluation model



Evaluation model



Outline

Service-oriented computing

Example: Virtual University

Conclusions

A Virtual University

The Sensoria Virtual University (SVU) is a (fictitious) virtual organisation formed by bringing together the resources of the universities at Edinburgh (UEDIN), Munich (LMU), Bologna (UNIBO), and Pisa (UNIFI). The SVU federates the teaching and assessment capabilities of the universities allowing students to enrol in courses irrespective of where they are delivered geographically.

Content upload and download

Students download *learning objects* from the content download portals of the universities involved and upload archives of their project work for assessment. By agreement within the SVU, students may download from (or upload to) the portals at any of the SVU sites, not just the one which is geographically closest.

SRMC description of the UEDIN server

```
UEDIN::{  
  lambda = 1.65;  mu = 0.0275;  gamma = 0.125;  delta = 3.215;  
  avail = { 0.7, 0.8, 0.9, 1.0 };  
  
  UploadPortal::{  
    Idle = (upload, avail * lambda).Idle + (fail, mu).Down;  
    Down = (repair, gamma).Idle;  
  }  
  
  DownloadPortal::{  
    Idle = (download, avail * delta).Idle + (fail, mu).Down;  
    Down = (repair, gamma).Idle;  
  }  
}
```

SRMC description of the UNIBO server

```
UNIBO::{  
  lambda = 1.65;  mu = 0.0275;  gamma = 0.125;  delta = 3.215;  
  slambda = 1.25;  sdelta = 2.255;  avail = { 0.8, 0.9, 1.0 };  
  
  UploadPortal::{  
    Idle = (upload, avail * lambda).Idle + (fail, mu).Down  
           + if avail > 0.8 then (supload, avail * slambda).Idle;  
    Down = (repair, gamma).Idle;  
  }  
  
  DownloadPortal::{  
    Idle = (download, avail * delta).Idle + (fail, mu).Down  
           + if avail > 0.8 then (sdownload, avail * sdelta).Idle;  
    Down = (repair, gamma).Idle;  
  }  
}
```

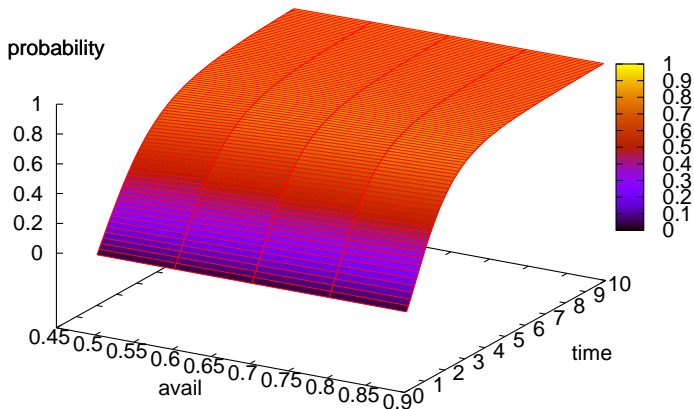
SRMC description of the Upload portal

```
SVU::UploadPortal =  
  { UEDIN::UploadPortal::Idle, LMU::UploadPortal::Idle,  
    UNIBO::UploadPortal::Idle, UNIPi::UploadPortal::Idle };
```

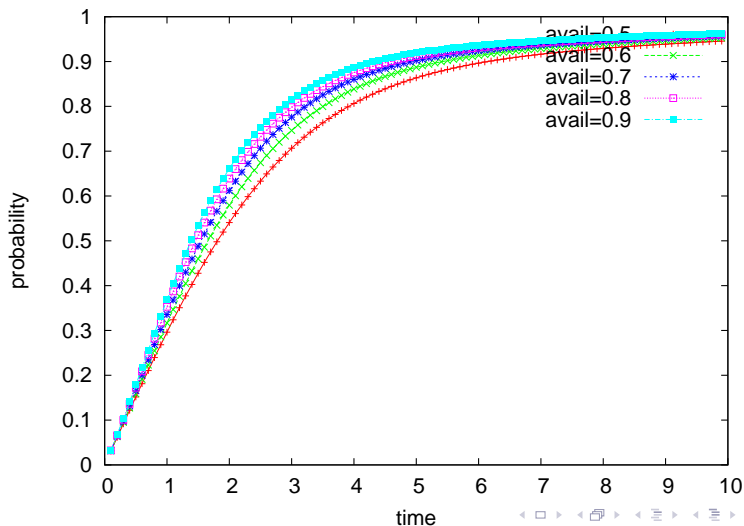
SRMC description of the Client

```
Client::{  
  Idle = (start, 1.0).Download;  
  Download = (download, _).(download, _).(download, _).Upload  
            + (sdownload, _).(sdownload, _).(sdownload, _).Upload;  
  Upload = (upload, _).(upload, _).Disconnect  
          + (supload, _).(supload, _).Disconnect;  
  Disconnect = (finish, 1.0).Idle;  
}
```

Varying the availability of the server



Varying the availability of the server



Service-level agreements for SOC

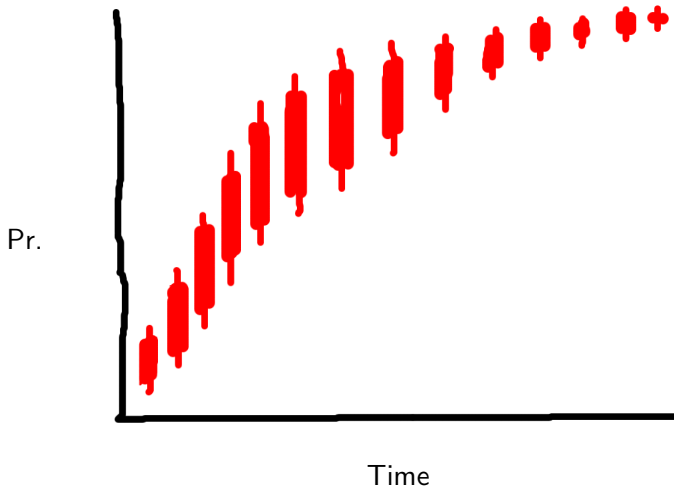
The type of service-level agreement which we would attempt to state for service-oriented computing systems would include a *confidence interval*, a *path* through the system, a *time bound* and lower and upper *probability bounds*.

Service-level agreements for SOC

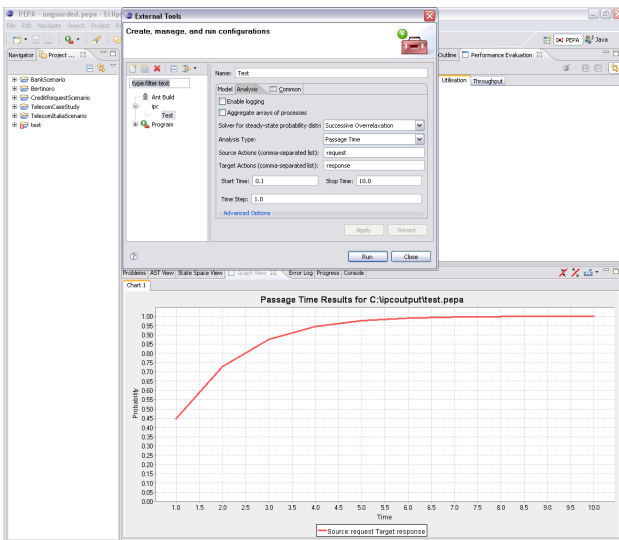
The type of service-level agreement which we would attempt to state for service-oriented computing systems would include a *confidence interval*, a *path* through the system, a *time bound* and lower and upper *probability bounds*.

For example: “Ninety-five percent of requests receive a response within 3 seconds with probability between 87.25% and 92.65%” .

Artist's impression of the results



Eclipse user interface



Outline

Service-oriented computing

Example: Virtual University

Conclusions

Further work

We would like to use SMRC as a *virtual process calculus* in the sense that it federates the resources of other calculi.

- ▶ We intend to develop an SRMC to FSP mapping so that we can check safety and liveness properties of models using LTSA.
- ▶ We hope to develop an SRMC to Stochastic FSP mapping so that we can simulate models with non-exponential distributions.
- ▶ We are interested in extending ipc to map to the SMARTA Laplace-transform-based response-time analyser so that we can analyse models with non-exponential distributions.

TODOs

- ▶ Develop good static analysis for SRMC to prevent analysis of flawed models.
- ▶ Extend dynamic analysis in Hydra to report errors found during state-space generation.
- ▶ Combine dynamic analysis from all runs with IPC and Hydra to feed back to the SRMC level.
- ▶ Extend Condor support in IPC to support SUN Grid Engine also.

Conclusions

- ▶ We addressed the inherent uncertainty in service-oriented computing by analysing by cases. We perform parameter sweep for each case. We evaluate these in parallel using Condor.
- ▶ The analysis methods scale well with increasing problem size.
- ▶ We build on tried and trusted compilers and analysers.
- ▶ Hopefully a “real world” approach to a “real world” problem.