Service-Level Agreements for Service-Oriented Computing

Clark, A., Gilmore, S., & Tribastone, M. (2009). Service-Level Agreements for Service-Oriented Computing. In A. Corradini, & U. Montanari (Eds.), Recent Trends in Algebraic Development Techniques. (pp. 21-36). (Lecture Notes in Computer Science). Springer Berlin / Heidelberg. 10.1007/978-3-642-03429-9_3

Service-Level Agreements for Service-Oriented Computing

Allan Clark, Stephen Gilmore and Mirco Tribastone

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

Abstract. Service-oriented computing is dynamic. There may be many possible service instances available for binding, leading to uncertainty about where service requests will execute. We present a novel Markovian process calculus which allows the formal expression of uncertainty about binding as found in service-oriented computing. We show how to compute meaningful quantitative information about the quality of service provided in such a setting. These numerical results can be used to allow the expression of accurate service-level agreements about service-oriented computing.

1 Introduction

Dynamic configuration is the essence of service-oriented computing. Service providers publish their services in a public registry. Service consumers discover services at run-time and bind to them dynamically, choosing from the available service instances according to the criteria which are of most importance to them. This architecture provides robust service in difficult operational conditions. If one instance of a service is temporarily unavailable then another one is there to take its place. It is likely though that this replacement is not fully functionally identical. It might have some missing functionality, or it might even offer additional functionality not found in the temporarily unavailable service instance.

However, even in the case of a functionally-identical replacement matters are still not straightforward when non-functional criteria such as availability and performance are brought into the picture. It is frequently the case that the functionally-equivalent replacement for the temporarily unavailable service will exhibit different performance characteristics simply because it hosts a copy of the service on another hardware platform. This impacts on essentially all performance measures which one would think to evaluate over the system configuration.

The world of distributed systems in which service-oriented computing resides is resource-sharing in nature. In such systems we have the additional complication that services may only be partially available in the sense that they are operational, but heavily loaded. In principle, all of their functionality is available, but only at a fraction of the usual level of performance. This becomes a pressing concern when service providers wish to advertise service-level agreements which

provide service consumers with formal statements about the quality of service offered. For example, a service provider might believe that 90% of requests receive a response within 3 seconds, but how can they check this?

Analytical or numerical performance evaluation provides valuable insights into the timed behaviour of systems over the short or long run. Prominent methods used in the field include the numerical evaluation of continuous-time Markov chains (CTMCs). These bring a controlled degree of randomness to the system description by using exponentially-distributed random variables governed by rate constants to characterise activities of varying duration. Often generated from a high-level description language such as a Petri net or a process algebra, CTMCs are applied to study fixed, static system configurations with known subcomponents with known rate parameters. This is far from the operating conditions of service-oriented computing where for critical service components a set of replacements with perhaps vastly different performance qualities stand ready to substitute for components which are either unavailable, or the consumer just simply chooses not to bind to them. How can we bridge this gap and apply Markovian performance evaluation to the assessment of service-level agreements about service-oriented computing?

In the present paper we propose a new Markovian process calculus which includes language constructs for the formal expression of uncertainty about binding and parameters (in addition to the other dimension of uncertainty about durations modelled in the Markovian setting through the use of exponentially-distributed random variables). We put forward a method of numerical evaluation for this calculus which scales well with increasing problem size to allow precise comparisons to be made across all of the possible service bindings and levels of availability considered. Numerical evaluation is supported inside a modelling environment for the calculus. We demonstrate the approach by considering an example of a (fictional) virtual university formed by bringing together the resources of several (real) universities. Our calculus is supported by a freely-available software tool.

Structure of this paper: In Section 2 we introduce our new Markovian calculus. In Section 3 we present an example service-oriented computing system, a "virtual university". In Section 4 we describe the analysis which can be performed on our process calculus models. In Section 5 we explain the software tools which we use. We present our conclusions in Section 6.

2 SRMC: Sensoria Reference Markovian Calculus

SRMC is a Markovian process calculus in the tradition of PEPA [1], Stochastic KLAIM [2], and Stochastic FSP [3]. On top of a classical process calculus, SRMC adds *namespaces* to allow the structured description of models of large size, and *dynamic binding* to represent uncertainty about component specification or the values of parameters. As a first step in machine processing, namespaces and dynamic binding can be resolved in order to map into a Markovian calculus

without these features such as PEPA (for performance analysis [4, 5]). Going further, rate information can also be erased in order to map into an untimed process calculus such as FSP (for analysis of safety and liveness properties [6]).

Namespaces in SRMC may be nested. Dynamic binding is notated by writing in the form of a set all of the possible values which may be taken. The binding records that the value is one of the values in the set (but we are not sure which one). The following example uses the name `UEDIN` for a location, the name `Server` for the server located there, the constant `processors` for the number of processors which the Edinburgh server has, and the constant `availability` for the availability of the server (which is between 50% and 100%).

```
UEDIN::{
   Server::{
      processors = 2;
      availability = { 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 };
   }
   ...
}
```

Outside the namespace scope one refers to the first constant using the fully qualified name `UEDIN::Server::processors` and to the second using the name `UEDIN::Server::availability`.

In addition to being able to give names to numerical constants and values it is also possible to give names to processes (in order to describe recursive behaviour). Process terms are built up using prefix (`.`) and choice (`+`). The following process definition describes a lossy buffer which loses, on average, one datum in every ten. As the example shows, activity rates can be conditioned by probabilities (0.1 and 0.9 here).

```
LossyBuffer::{
   Empty = (put, 0.1 * r).Empty + (put, 0.9 * r).Full;
   Full = (get, s).Empty;
}
```

Processes of the SRMC language give rise to labelled transition systems which are converted to Continuous-Time Markov Chain (CTMC) representations in the way which is familiar from PEPA [1].

Process expressions can be defined conditionally in SRMC depending on the values obtained in the resolution of dynamic binding. For example, a server might allow additional sessions to be opened if availability is above 70% and forbid the creation of new sessions otherwise.

```
if availability > 0.7 then (openSession, r).ServeClient
```

An equivalent effect can be obtained using *functional rates* [7] which can allow the use of space-efficient state-space representation using Kronecker methods. The equivalent process expression using functional rates is below.

```
(openSession, if availability > 0.7 then r else 0.0).ServeClient
```

In stochastic Petri nets functional rates are termed "marking dependent rates".

Dynamic service binding is described by associating a name with a set of processes. The example below records that the server is either the Edinburgh server (`UEDIN`) or the Bologna server (`UNIBO`).

```
Server = { UEDIN::Server, UNIBO::Server };
```

2.1 Discussion

It might seem that it is not necessary to have the ability to describe sets of processes, binding to one of these later because it would be possible to implement the idea of dynamic binding instead using well-known process calculus primitives. For example, one could use a silent, internal τ transition at the start of the lifetime of one of the components to choose to behave as one of the binding sites, thereafter ignoring all of the possible behaviour described by the other components from the other sites. While this is possible, we do not favour this approach because it leads to the consideration of the full state space for every evaluation of parameters of the system. In contrast, the method of first projecting down to a particular binding and then evaluating this leads to the smallest possible state-space for each evaluation run, with attendant benefits for run-times and stability of the results. Further, the binding projection method allows the problem to be decomposed in a larger number of smaller problems, each of which can be solved independently and the results combined. We wish to perform scalable analysis of scalable systems and so this approach suits us well.

2.2 Numerical evaluation

We have been keen to decompose the analysis problem so that we can ensure that the analysis can be performed as a large number of numerical evaluations of small size. Our preference for problems of this form stems from the fact that they are easy to distribute across a network of workstations. Thus, we use a distributed computing platform (Condor [8]) to accelerate the numerical evaluation work by distributing the computation across a cluster of workstations (a Condor "pool"). In this way we can greatly increase the speed of generation of results. In practice we have found that our Condor pool of 70 machines gives a speedup over sequential evaluation close to 70-fold. Because we are aware that others may wish to use our software but may not have a local Condor pool we also provide a purely sequential evaluation framework which does not depend on Condor.

We know that numerical linear algebra is not to everyone's taste so we will just give an outline of what we do here and refer the curious to [9]. Investigation of SLAs requires the transient analysis of a CTMC, represented as an $n \times n$ state transition matrix Q (the "generator matrix"). We are concerned with finding the transient state probability row vector $\pi(t) = [\pi_1(t), \ldots, \pi_n(t)]$ where $\pi_i(t)$ denotes the probability that the CTMC is in state i at time t. Transient

and passage-time analysis of CTMCs proceeds by a procedure called *uniformisation* [10, 11]. The generator matrix, Q, is "uniformized" with:

$$P = Q/q + I$$

where $q > \max_i |Q_{ii}|$ and I is the identity matrix. This process transforms a CTMC into one in which all states have the same mean holding time $1/q$.

Passage-time computation is concerned with knowing the probability of reaching a designated target state from a designated source state. It rests on two key sub-computations. First, the time to complete n hops ($n = 1, 2, 3, \ldots$), which is an Erlang distribution with parameters n and q. Second, the probability that the transition between source and target states occurs in exactly n hops.

3 Example: Distributed e-Learning Case Study

Our general concern is with evaluating quality of service in the presence of uncertainty such as that caused by dynamic binding but as a lighthearted example to illustrate the approach we consider a (fictional) Web Service-based distributed e-Learning and course management system run by the Sensoria Virtual University (SVU).

The SVU is a virtual organisation formed by bringing together the resources of the universities at Edinburgh (UEDIN), Munich (LMU), Bologna (UNIBO), Pisa (UNIPI) and others not listed in this example. The SVU federates the teaching and assessment capabilities of the universities allowing students to enrol in courses irrespective of where they are delivered geographically. 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.

Learning objects may contain digital audio or video presentation of lecture courses and students may be required to upload archives of full-year project work. Both of these may be large files so the *scalability* of such a system to support large numbers of students is a matter of concern. We have addressed this issue previously [12, 13].

3.1 The servers

We start by describing the servers which are available for use. Dedicated upload and download portals are available at each site. At Edinburgh the portals sometimes fail and need to be repaired before they are available to serve content again. They are usually relatively lightly loaded and availability is between 70% and 100%. The portals at Edinburgh are described in SRMC thus.

```
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;
   }
}
```

The portals at Munich are so reliable that it is not worth modelling the very unlikely event of their failure. However, they are slower than the equivalent portals at Edinburgh and availability is more variable and usually lower, because the portals are serving a larger pool of local students.

```
LMU::{
   lambda = 0.965;   delta = 2.576;
   avail = { 0.5, 0.6, 0.7, 0.8, 0.9 };
   UploadPortal::{
      Idle = (upload, avail * lambda).Idle;
   }
   DownloadPortal::{
      Idle = (download, avail * delta).Idle;
   }
}
```

Because it is running a more recent release of the portal software the Bologna site offers secure upload and download also. Availability is usually very good. To maintain good availability the more expensive operations of secure upload and secure download are not offered if the system seems to be becoming heavily loaded.

```
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;
   }
}
```

The Pisa site is just like the Bologna site, but uses a higher grade of encryption, meaning that secure upload and download are slower (`slambda` = 0.975, `sdelta`

= 1.765). We can list the possible bindings for upload and download portals in the following way.

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

DownloadPortal =
   { UEDIN::DownloadPortal::Idle, LMU::DownloadPortal::Idle,
     UNIBO::DownloadPortal::Idle, UNIPI::DownloadPortal::Idle };
```

3.2 The clients

We now describe two typical clients of the system, Harry and Sally. Both Harry and Sally wish to accomplish the same task, which is to download three sets of learning materials and to upload two coursework submissions. They perform this behaviour cyclically. Harry is unconcerned about security and never uses secure upload or download even if it is available. Sally uses secure upload and secure download sometimes when it is available, and uses non-secure upload and download when it is not. We are interested in the passage of time from start to finish for both Harry and Sally. Clients do not determine the rates of activities: others do (we write "_" for the rate here).

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

Sally::{
   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;
}
```

The client is either Harry or Sally, both initially idle.

```
Client = { Harry::Idle, Sally::Idle };
```

Finally, the complete system is formed by composing the client with the two portals, cooperating over upload and download. The upload and download portals do not communicate with each other (<>).

```
System = Client <upload, download, supload, sdownload>
             (UploadPortal <> DownloadPortal);
```

4 Analysis

The analysis applied to SRMC models is a staged computation:

Resolving service bindings: Each possible service binding is chosen in turn. This involves selecting one element of each set of possibilities for service providers.

Model minimisation: The model is reduced to remove unused definitions of processes and rate constants. This is a necessary economy applied to make the next stage more productive.

Parameter sweep: Parameter sweep is performed over the remaining rate values, executing processes in a distributed fashion on a Condor pool, or sequentially on a single machine.

Analysis and visualisation: The results are collected and summarised using statistical procedures. We visualise the results to aid in model interpretation and analysis.

4.1 Qualitative analysis

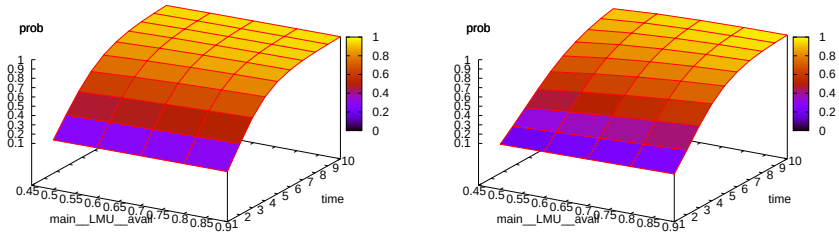
On the way towards the quantitative results which we seek our state-space analysis delivers qualitative insights about the function of the system being modelled. We list three of the things which we learn here:

1. The system is *deadlock-free* for all configurations. No binding of service instances to service parameters gave rise to a model with a deadlock.
2. The system is *livelock-free* for all configurations. No binding of service instances to service parameters gave rise to a model where states could be visited only a finite number of times (a *transient state*, in Markov chain terminology).
3. All activities in the model are *weakly live*. That is, for each activity (such as `supload`) there is some configuration which allows that activity to occur, although it may be blocked in other configurations. Put more plainly, the SRMC model has no "dead code" (activities which can never occur).

4.2 Sensitivity analysis

We are here concerned generally with lack of certainty about parameters such as rates but even in the case where rate information can be known with high confidence the framework which we have available for performing a parameter sweep across the rate constants can be used to perform *sensitivity analysis*. One way in which the results obtained by sensitivity analysis can be used is to determine which activities of the system are bottlenecks. That is, to discover which rate or rates should we alter to ensure that the user sees the greatest improvement in performance. We have an evaluation function which assigns a score to each solution of the underlying Markov chain. In this case, the less is the response time then the higher is the score.

It might seem that the results obtained from sensitivity analysis are likely to be pretty unsurprising and that it will turn out to be the case that increasing the rate of any activity brings about a proportional decrease in response time. To see that this is not the case, we will compare two sets of results. Recall that SRMC generates many PEPA models; we number these. The first set of results shown in Fig. 1 comes from PEPA model 1, where Edinburgh is the upload portal, Munich the download portal, and Harry is the client. In model 3 they swap around so that Munich is the upload portal, Edinburgh the download, and Harry is again the client. In the latter case low availability of the Munich server makes a noticeable impact on response time (the curve takes longer to get up to 1) but in the former case the low availability of the Munich server has negligible impact. This is made clear in the results but it is unlikely that a modeller would see this trend just by inspecting the model; we needed to see the results to get this insight. We have generated many results so we have been able to get many such insights.



(Results of PEPA model 1) (Results of PEPA model 3)

Fig. 1. Graphs showing sensitivity analysis over the rates in the produced models. The basic plot is a cumulative distribution function showing how the probability of completion of the uploads and downloads increases as a function of time. The surface plot is obtained from this because we vary one of the parameters. Here in both cases we vary the availability of the Munich server from 50% availability to 90% availability. Expressed as a scaling factor this becomes 0.5 to 0.9.

4.3 Computing response-time percentiles

The results shown in Fig. 1 show ten of the nearly 250 cumulative distribution functions which we computed for the possible configurations of the example. We wanted to produce a simple statistical summary which brought together all of the results obtained. We computed *percentiles* of the results which declare that in (say) 90% of the possible configurations of the system the response-time will be in this region. This tells us about the experience which most users will have (where here, "most" means "90% of"). Some will see better response times, and

some with see worse, but it is usually interesting to consider the common case response times.

To illustrate how percentiles can be used to summarise the results we show in Fig. 2(a) forty sets of results in the form of the cumulative distribution functions which we computed. These give a sense of the "envelope" in which the results are contained. Most configurations of the system produced by resolving the service instance bindings are very likely to have completed the work to be done by $t = 10$. The majority of configurations give response-time distributions which put them towards the top of the "envelope" but there are a few configurations which perform quite a bit worse (and our analysis has identified which configurations these are).

The graph in Fig. 2(b) is known as a "candlestick" graph and is a summary of all of the solutions produced. It shows that 90% of the time the response time distribution will lie within the area described by the thick bar of the candlestick, but it has been seen to be as high as the top of the candlestick, and it has been seen to be as low as the bottom of the candlestick.

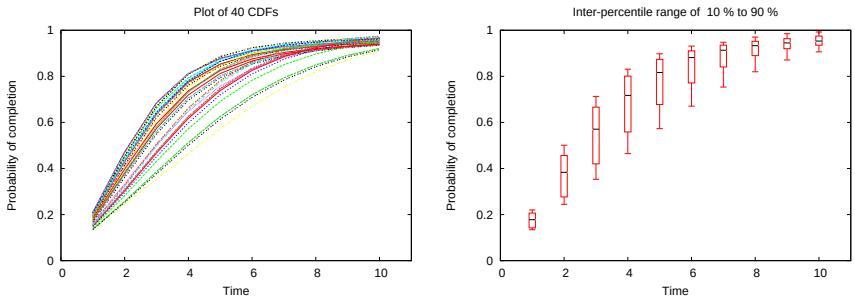


Fig. 2. Sub-figure (a) shows 40 of the response-time distributions computed for the Sensoria Virtual University example. Sub-figure (b) shows the 10% to 90% percentile of the results over all of the runs. The median value is also marked as a horizontal line cutting across the thick bar in the candlestick. From sub-figure (b) we can report results of the form "All uploads and downloads will have completed by time $t = 10$ with probability between 0.90 and 0.97, in 90% of configurations".

4.4 Comparisons across all runs

Even for traditional computer systems without dynamic binding, service-level agreements are already quite complex because they relate a path through the system behaviour, a time bound, and a probability bound. (A typical example of an SLA is "We guarantee that 97.5% of requests will receive a response within three seconds". Here "from request to response" is the path through the system,

three seconds is the time bound, and 97.5% gives the probability bound.) In the service-oriented computing setting we have yet another dimension of complication because we must add a qualifier speaking about the quantile of system configurations being considered ("...in 90% of the possible configurations"). Complicated service-level agreements of this form are unattractive.

We have found that an alternative presentation of the results can be easier to interpret in some cases and so the SRMC software supports a presentation mode where we show the probability of completion by a particular point in time, across all possible configurations. Having all of the results to hand, we are able to reduce the dimension of the problem and make statements about the probability of completion of the work at a particular point in time, irrespective of the configuration of the system.

In reference to Fig. 3 we can see not a statistical summary (as we saw in Fig. 2(b) before) but the actual results of all runs at a particular point in time. This makes clear the difference between the best-performing configurations at time t and the worst-performing configurations at time t. For low values of t such as 1.0 there is little chance that any user has completed all uploads and downloads. For high values of t such as 10.0 there is little chance that they have not.

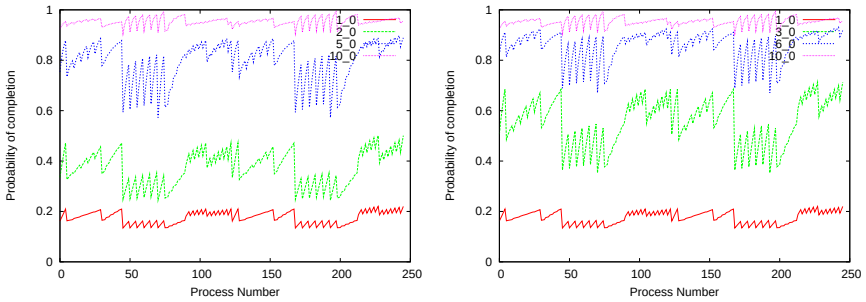


Fig. 3. Probability of completion of all uploads and downloads against time across all (nearly 250) possible configurations of the example. In sub-figure (a) the times considered are $t = 1.0, 2.0, 5.0$, and 10.0. In sub-figure (b) $t = 1.0$ and 10.0 are repeated for reference and $t = 3.0$ and 6.0 are also presented. By time $t = 10.0$ we are able to make meaningful comments about all configurations. For example, we can say that there is at least a 90% chance of having completed the uploads and downloads by time $t = 10.0$, irrespective of the system configuration. The greatest variability is seen at times around $t = 3.0$. Here for the best configurations the system has a 70% chance of having completed the work for the worst configurations there is less than a 40% chance of having completed.

5 Software Tool Support

SRMC is supported by a tool chain whose main design goal has been to provide a friendly and rich graphical user interface as well as a set of efficient model solvers. The software comprises a graphical front-end written in Java for the Eclipse framework and a back-end implemented in Haskell and C++. The latter exposes its functionality via a command-line interface, and thus can be used as a stand-alone application in headless environments such as Condor or to reduce the tool's overall memory footprint. This section provides an overview of both modules; further information is available at the SRMC Web site [14], which also provides a download link to the tool.

5.1 Analysis tools in the back-end

The analysis back-end is implemented as a series of three applications: the Sensoria Markovian Compiler (`smc`), the Imperial PEPA Compiler (`ipc`) and the Hypergraph-based Distributed Response-Time Analyser (`hydra`). `smc` accepts SRMC models as input and generates the intermediate PEPA descriptions that represent all the possible configurations of the system. The main tasks performed by `smc` are resolving binding instantiations, name-resolution and flattening of the SRMC model's namespaces, and generation of PEPA models for analysis. A database file produced by `srmc` maintains associations between the original SRMC model and the underlying PEPA models.

Such models are the basic units on which analysis is to be carried out. As PEPA benefits from extensive software support, a number of analysis tools are readily available for re-use in this context. Here, each PEPA model is run through `ipc` [15]. It translates the description into a format suitable for `hydra` [16], which performs passage-time analysis and stores the results to disk. Such results can be related back to the SRMC description via the database file from `smc`.

5.2 Presentation layer at the front-end

The graphical user interface is implemented as a contribution (*plug-in*) to Eclipse, a popular extensible cross-platform development framework. The plug-in provides an editor and a standard Eclipse contribution to the *Outline* view to concisely display information about the model. The plug-in also adds a top-level menu item through which SRMC features are accessible. In particular, a wizard dialogue guides the user through the set-up of passage-time analysis. Upon completion, the wizard schedules an array of background processes that run the back-end tool chain as described above. All the intermediate resources such as the PEPA model instances and the `hydra` description files are available in the user's workspace for further inspection via the Eclipse *Navigator* view. When the analysis is complete, the results are collected and presented to the user as a plot in the *Graph* view. Figure 4 shows a screenshot of an Eclipse session running the SRMC plug-in.

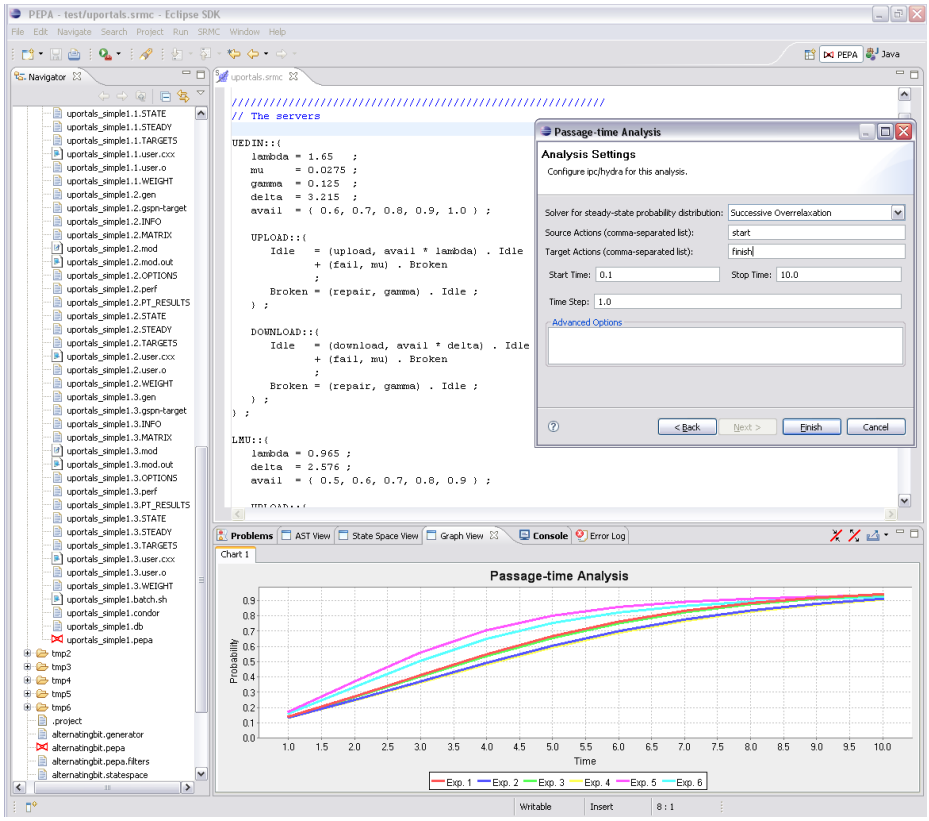


Fig. 4. Screenshot showing the SRMC Eclipse plug-in processing the SVU example. Displayed in the screenshot are (i) the workspace navigator showing compiled representations of the SRMC model as PEPA models, Hydra models and compiled Hydra C++ files; (ii) the SRMC model editor; (iii) the user-interface dialogue used for setting parameters on the analyser and running the transient analysis repeatedly; and (iv) a graphical display showing the results of all passage-time analysis runs expressed in the form of the cumulative distribution functions computed numerically by the Markov chain solver. In addition to providing user-interface widgets, the plug-in exposes SRMC tools to the framework through an application programming interface for third-party Eclipse plug-ins.

6 Conclusions

For software engineering to improve as a well-managed discipline we believe that it is critical to have access to a modelling process which can make sound quantitative predictions about the performance of complex systems. We have addressed the problem of how virtual organisations can defend any quantitative statements about their quality of service as expressed in service-level agreements given that

their operation is founded on service-oriented computing. The essential function of dynamic binding brings uncertainty to the model concerning both functional and non-functional aspects. We have been able to control this uncertainty by considering all possible bindings, undertaking separate numerical evaluations of these, and combining the results to correctly quantify the uncertainty induced by dynamic binding and degree of availability.

We decomposed the computations needed into a large number of independent numerical evaluations each of which has modest memory requirements. We distributed the independent runs across a network of workstations. The distributed computing platform which we chose, Condor, makes use of the idle cycles on networked workstations meaning that we could perform all of the computations which were needed on typical desktop PCs when they were unused in our student computing laboratories. Widely-used in computational science, this approach uses stock hardware and scales well to apply to more complex problem cases with a greater range of possible configurations and parameter values. More computing power can be deployed on larger problems simply by adding more machines to the Condor pool. We hope that this is a "real-world" approach to a "real-world" problem.

In our numerical evaluation of the many possible system configurations which are described by an SRMC model we have essentially used the "brute force" solution of solving for all possible bindings. This has the advantage that it ensures that all of the bindings are considered, and is trivially parallelisable, but still costs a lot of computation time. It is possible that we could do fewer numerical evaluations and still explore the space of all possibilities well by applying methods which are well-known in the field of design of experiments. Similar strategic exploration of the solution space is found in state-of-the-art modelling platforms such as Möbius [17].

Acknowledgements: The authors are supported by the EU FET-IST Global Computing 2 project SENSORIA ("Software Engineering for Service-Oriented Overlay Computers" (IST-3-016004-IP-09)). The Imperial PEPA Compiler was developed by Jeremy Bradley of Imperial College, London. The Hydra response-time analyser was developed by Will Knottenbelt and Nick Dingle of Imperial College, London. We extended both of these software tools for the present work.

References

1. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
2. De Nicola, R., Katoen, J.P., Latella, D., Massink, M.: STOKLAIM: A stochastic extension of KLAIM. Technical Report ISTI-2006-TR-01, Consiglio Nazionale delle Ricerche (2006)
3. Ayles, T.P., Field, A.J., Magee, J., Bennett, A.: Adding Performance Evaluation to the LTSA Tool. In: Tool demonstration, 13th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, September 2003. (September 2003)

4. Clark, A.: The ipclib PEPA Library. In Harchol-Balter, M., Kwiatkowska, M., Telek, M., eds.: Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), IEEE (September 2007) 55–56
5. Tribastone, M.: The PEPA Plug-in Project. In Harchol-Balter, M., Kwiatkowska, M., Telek, M., eds.: Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), IEEE (September 2007) 53–54
6. Magee, J., Kramer, J.: Concurrency: State Models and Java Programming. Second edn. Wiley (2006)
7. Hillston, J., Kloul, L.: An efficient Kronecker representation for PEPA models. In de Alfaro, L., Gilmore, S., eds.: Proceedings of the first joint PAPM-PROBMIV Workshop. Volume 2165 of Lecture Notes in Computer Science., Aachen, Germany, Springer-Verlag (September 2001) 120–135
8. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency – Practice and Experience **17**(2–4) (2005) 323–356
9. Knottenbelt, W.: Performance Analysis of Large Markov Models. PhD thesis, Imperial College of Science, Technology and Medicine, London, UK (February 2000)
10. Grassmann, W.: Transient solutions in Markovian queueing systems. Computers and Operations Research **4** (1977) 47–53
11. Gross, D., Miller, D.: The randomization technique as a modelling tool and solution procedure for transient Markov processes. Operations Research **32** (1984) 343–361
12. Gilmore, S., Tribastone, M.: Evaluating the scalability of a web service-based distributed e-learning and course management system. In Bravetti, M., Núñez, M.T., Zavattaro, G., eds.: Third International Workshop on Web Services and Formal Methods (WS-FM'06). Volume 4184 of Lecture Notes in Computer Science., Vienna, Austria, Springer (2006) 156–170
13. Bravetti, M., Gilmore, S., Guidi, C., Tribastone, M.: Replicating Web Services for scalability. Submitted for publication (October 2007)
14. SRMC Team: Sensoria Reference Markovian Calculus Web Site and Software. Available on-line at http://groups.inf.ed.ac.uk/srmc (October 2007)
15. Bradley, J., Dingle, N., Gilmore, S., Knottenbelt, W.: Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In Kotsis, G., ed.: Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, University of Central Florida, IEEE Computer Society Press (October 2003) 344–351
16. Dingle, N., Harrison, P., Knottenbelt, W.: HYDRA: HYpergraph-based Distributed Response-time Analyser. In: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2003), Las Vegas, Nevada, USA (June 2003) 215–219
17. Courtney, T., Gaonkar, S., McQuinn, M., Rozier, E., Sanders, W., Webster, P.: Design of Experiments within the Möbius Modeling Environment. In Harchol-Balter, M., Kwiatkowska, M., Telek, M., eds.: Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), IEEE (September 2007) 161–162