

Service-Level Agreements for Service-Oriented Computing

(Extended abstract)

Allan Clark, Stephen Gilmore and Mirco Tribastone*

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 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]). In the present paper we discuss only performance analysis.

2 Example: Distributed e-Learning Case Study

Our general concern is with evaluating quality of service in the presence of 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 (UNIFI) 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*

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

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 [7].

2.1 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. Rather than repeat the definitions, we use the Bologna namespace as a *prototype* from which the Pisa definitions are obtained by overriding the necessary constants.

```

UNUPI = UNIBO{ slambda = 0.975; sdelta = 1.765; }

```

This is an abbreviation for

```

UNUPI::{
  lambda = 1.65; mu = 0.0275; gamma = 0.125; delta = 3.215;
  slambda = 0.975; sdelta = 1.765; 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;
  }
}

```

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

```

SVU::{
  UploadPortal = *::UploadPortal::Idle
  DownloadPortal = *::DownloadPortal::Down;
}

```

This is a shorthand abbreviation for an explicit listing of the sets of possible bindings, thus.

```

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

```

```
SVU::DownloadPortal =
  { UEDIN::DownloadPortal::Down, UNIBO::DownloadPortal::Down,
    UNIPI::DownloadPortal::Down };
```

2.2 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.

```
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.

```
Client = { Harry, Sally };

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

3 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 process definitions and rate expressions.

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

The PEPA language is used as an intermediate language in this process. The SRMC model is compiled down to the PEPA language using `srmc` (the Sensoria Reference Markovian Compiler), to produce a PEPA model similar to the following.

```
lambda__UEDIN = 1.65; mu__UEDIN = 0.0275; gamma__UEDIN = 0.125;
delta__UEDIN = 3.215;
```

```
UEDIN__UploadPortal__Idle =
    (upload, avail * lambda__UEDIN).UEDIN__UploadPortal__Idle
    + (fail, mu__UEDIN).UEDIN__UploadPortal__Down;
```

```
UEDIN__UploadPortal__Down =
    (repair, gamma__UEDIN).UEDIN__UploadPortal__Idle;
```

```
UEDIN__DownloadPortal__Idle =
    (download, avail * delta__UEDIN).UEDIN__DownloadPortal__Idle
    + (fail, mu__UEDIN).UEDIN__DownloadPortal__Down;
```

```
UEDIN__DownloadPortal__Down =
    (repair, gamma__UEDIN).UEDIN__DownloadPortal__Idle;
```

This model is then analysed using `ipc` and `Hydra`.

References

- [1] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
- [2] R. De Nicola, J. P. Katoen, D. Latella, and M. Massink, “STOKLAIM: A stochastic extension of KLAIM,” Tech. Rep. ISTI-2006-TR-01, Consiglio Nazionale delle Ricerche, 2006.
- [3] T. P. Ayles, A. J. Field, J. Magee, and A. Bennett, “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] A. Clark, “The ipclib PEPA Library,” in *Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST)* (M. Harchol-Balter, M. Kwiatkowska, and M. Telek, eds.), IEEE, Sept. 2007. To appear.
- [5] M. Tribastone, “The PEPA Plug-in Project,” in *Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST)* (M. Harchol-Balter, M. Kwiatkowska, and M. Telek, eds.), IEEE, Sept. 2007. To appear.
- [6] J. Magee and J. Kramer, *Concurrency: State Models and Java Programming*. Wiley, second ed., 2006.
- [7] S. Gilmore and M. Tribastone, “Evaluating the scalability of a web service-based distributed e-learning and course management system,” in *Third International Workshop on Web Services and Formal Methods (WS-FM’06)* (M. Bravetti, M. T. Núñez, and G. Zavattaro, eds.), vol. 4184 of *Lecture Notes in Computer Science*, (Vienna, Austria), pp. 156–170, Springer, 2006.