Service Composition for Collective Adaptive Systems

Service composition for collective adaptive systems

Stephen Gilmore[1], Jane Hillston[1], and Mirco Tribastone[2]

[1] Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh, Scotland
[2] Electronics and Computer Science, University of Southampton, Southampton, England

Abstract. Collective adaptive systems are large-scale resource-sharing systems which adapt to the demands of their users by redistributing resources to balance load or provide alternative services where the current provision is perceived to be insufficient. Smart transport systems are a primary example where real-time location tracking systems record the location availability of assets such as cycles for hire, or fleet vehicles such as buses, trains and trams. We consider the problem of an informed user optimising his journey using a composition of services offered by different service providers.

1 Introduction

Flexible composition of services lies at the heart of collective adaptive systems (CAS) where the collective interaction of users of the system shapes future system behaviour because the system adapts to patterns of use. Adaptive systems such as these are subject to a continuous process of tuning based on measurement data collected by the system itself through integrated instrumentation. Use of the services provided by the system achieves goals which are important to the user (perhaps a goal as simple as travelling across the city to enjoy a social occasion with friends and colleagues) but it also alters the system so that user experience in the future will be affected by this use of this service, even if only very subtly. Service provision in the future depends on decisions made by transport system operators, based on perceived demand for services as determined by collective journey statistics.

CAS depend on real-time measurement and monitoring of their services coupled with the dissemination of service availability information, allowing users to make informed choices. The provision of real-time information makes it possible for users to interact intelligently with adaptive systems and to make informed decisions which are supported by vital, current information.

Investigation of such systems by the construction of formal models of their behaviour is a hugely productive activity. A formal model provides a compact representation of an important aspect of a complex system, throwing light on the most significant issues and giving us the intellectual tools to study them closely. In this paper we consider a formal model of a collective adaptive system which is composed of distributed services. In particular we study an integrated smart transport system which blends public transport and self-powered transport in an effort to solve the so-called *last mile* transport problem experienced in modern cities. This problem arises because although public transport can be used to transport a passenger close to their intended destination, a final stage of the journey (the “last mile”) remains to be travelled in another way. The consequence of not addressing the last mile transport problem is that users become disenchanted with the service and resort to private transport, putting more cars on the road with negative consequences for road congestion and the environment.

Specifically, we consider the interaction between a real-time public-transport tracking service, a location-identification service, a transport-planning service, and a cycle-hire service, from the point-of-view of public transport passengers. These passengers also subscribe to a cycle-hire scheme and wish to optimise their journey to their destination. Subscribers in a cycle-hire scheme can borrow cycles from a cycle station when they need one, use the cycle for their allotted time, and then return the cycle to the cycle station nearest to their destination.

Authors' note. Our interest in service composition, and the development of stochastic modelling techniques suitable for studying the performance of composed services, can be directly attributed to Martin. We had the great privilege of working with him on the SENSORIA project and we look forward to future opportunities to travel through Munich to enjoy a Maß with him, availing ourselves of smart transport services (such as those described in our scenario below).

2 Scenario: Travel in Munich

As our running example, we consider how the situation of users of an integrated smart transport system can be assisted by automated tools which allow them to make an optimal choice between alternative routes to reach a desired destination. If journey times were deterministic it would be possible to easily compute the shortest path. However, in reality all transport systems exhibit a great deal of uncertainty in journey times due their inherent stochasticity: a tram can break down, a bicycle tyre can go flat, traffic congestion may affect a bus journey, and so on. Our idea is to be able to compute an optimal path on-line using the current state of the system.

To be more concrete and to provide a familiar scenario, we set our example in Munich during Oktoberfest, and assume that the hard-working staff of the *Ludwig-Maximilians-Universität* at the LMU building in Oettingenstrasse wish to plan their journey to Theresienwiese for their well-earned *Maß*[3] after a long day in the office. Naturally they prefer to minimise their journey time. We assume that a user, hereby denoted by M, has a choice between the following three routes:

[3] Maß is the Bavarian for a mug of beer, equivalent to 1 litre.

1. Take the 54 bus from Hirschauer Strasse to Giselastrasse. We assume that a bike-sharing station is available at Giselastrasse. Thus M has three options: directly walk or cycle to Theresienwise, or change with the underground line U6 to Odeonsplatz. At Odeonsplatz is another bike-sharing station; now M has the choice to either walk or cycle to Theresienwise.
2. M may prefer to start the journey with a relaxing stroll through the English Gardens, and take the occasion to drop off a document at the LMU building in Leopoldstrasse. The journey can then continue by walking to nearby Universität U-Bahn station. There M will decide between continuing by bike to Theresienwise, or taking the U3 to Odeonsplatz, where he will choose between cycling or walking, as in Route 1.
3. Take the tram 18 to Lehel from Tivolistrasse, and change with the U4 to Hauptbahnhof. We assume the existence of a bike-sharing station at Lehel and Hauptbahnhof, thus M has always the choice to directly walk or cycle to Theresienwise.

We will refer to these as routes #1, #2 and #3.

3 Modelling

We may now represent this problem as a formal model with a network structure where the intermediate stops on the journey are represented as nodes in the network. We number the stops on each route to remind us of the journey. Thus route #1 has stops $\boldsymbol{S_{10}}$, $\boldsymbol{S_{11}}$ and $\boldsymbol{S_{12}}$. We can identify 12 nodes in the network. These are not all distinct and thus correspond to our 11 destinations of interest. The nodes and corresponding locations are given in Table 1.

Node	Location
$\boldsymbol{S_0}$	Origin: The LMU building in Oettingenstrasse, at the end of the day
$\boldsymbol{S_{10}}$	Hirschauer Strasse
$\boldsymbol{S_{11}}$	Giselastrasse
$\boldsymbol{S_{12}}$	Odeonsplatz
$\boldsymbol{S_{20}}$	The English Gardens
$\boldsymbol{S_{21}}$	The LMU building in Leopoldstrasse
$\boldsymbol{S_{22}}$	Universität U-Bahn station
$\boldsymbol{S_{23}}$	Odeonsplatz
$\boldsymbol{S_{30}}$	Tivolistrasse
$\boldsymbol{S_{31}}$	Lehel
$\boldsymbol{S_{32}}$	Hauptbahnhof
$\boldsymbol{D}$	Destination: Theresienwise, Oktoberfest, and a well-earned *Maß*

Table 1. Nodes and locations in the network.

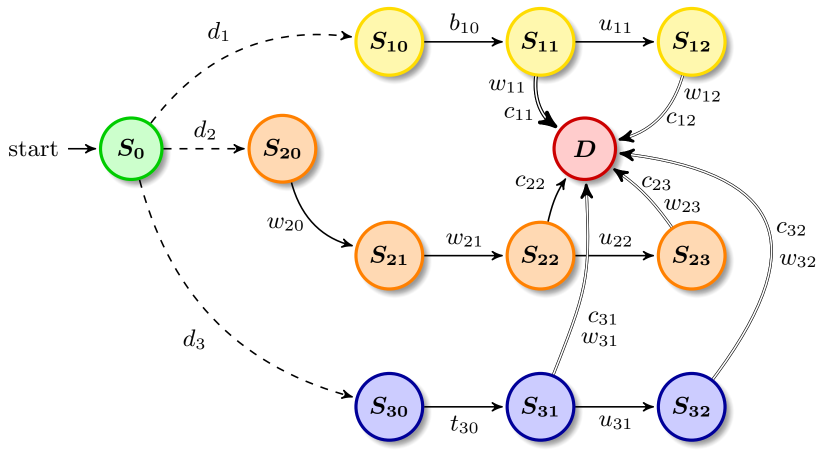


Fig. 1. Possible routes to the destination

Having named the nodes of interest we can now express the journey as a network such as the one illustrated in Figure 1 with our traveller M starting at the origin of the journey ($\boldsymbol{S_0}$) with options to travel via routes #1, #2 or #3, each of which has intermediate stops along the way.

Every route eventually passes relatively close to their desired destination ($\boldsymbol{D}$), but not so close that walking is the preferred option. Fortunately, cycle-hire stations are located at intermediate stops $\boldsymbol{S_{11}}$ and $\boldsymbol{S_{12}}$ on route #1, intermediate stops $\boldsymbol{S_{22}}$ and $\boldsymbol{S_{23}}$ on route #2, and intermediate stops $\boldsymbol{S_{31}}$ and $\boldsymbol{S_{32}}$ on route #3. The real-time public-transport tracking service predicts delays d_1, d_2 and d_3 for the public-transport services needed.

We write b_n and similarly for the average bus journey times from stop n. We write t_n for a tram journey from stop n and we use u_n to denote a journey by underground train. We write c_n for the average cycle journey time from stop n to the destination, and w_n for the average walking time. Journeys are always completed either by cycling or by walking, ending at the destination of Oktoberfest in our example ($\boldsymbol{D}$).

In this scenario there are fourteen possible journeys, depending on the route chosen, and where passenger M decides to alight in order to collect a cycle for the last stage of the journey.

$\bullet \xrightarrow{d_1} \bullet \xrightarrow{b_{10}} \bullet \xrightarrow{c_{11}} \bullet$

$\bullet \xrightarrow{d_1} \bullet \xrightarrow{b_{10}} \bullet \xrightarrow{w_{11}} \bullet$

$\bullet \xrightarrow{d_1} \bullet \xrightarrow{b_{10}} \bullet \xrightarrow{u_{11}} \bullet \xrightarrow{c_{12}} \bullet$

$\bullet \xrightarrow{d_1} \bullet \xrightarrow{b_{10}} \bullet \xrightarrow{u_{11}} \bullet \xrightarrow{w_{12}} \bullet$

$\bullet \xrightarrow{d_2} \bullet \xrightarrow{w_{20}} \bullet \xrightarrow{w_{21}} \bullet \xrightarrow{c_{22}} \bullet$

$\bullet \xrightarrow{d_2} \bullet \xrightarrow{w_{20}} \bullet \xrightarrow{w_{21}} \bullet \xrightarrow{u_{22}} \bullet \xrightarrow{c_{23}} \bullet$

$\bullet \xrightarrow{d_2} \bullet \xrightarrow{w_{20}} \bullet \xrightarrow{w_{21}} \bullet \xrightarrow{u_{22}} \bullet \xrightarrow{w_{23}} \bullet$

$\bullet \xrightarrow{d_3} \bullet \xrightarrow{t_{30}} \bullet \xrightarrow{c_{31}} \bullet$

$\bullet \xrightarrow{d_3} \bullet \xrightarrow{t_{30}} \bullet \xrightarrow{w_{31}} \bullet$

$\bullet \xrightarrow{d_3} \bullet \xrightarrow{t_{30}} \bullet \xrightarrow{u_{31}} \bullet \xrightarrow{c_{32}} \bullet$

$\bullet \xrightarrow{d_3} \bullet \xrightarrow{t_{30}} \bullet \xrightarrow{u_{31}} \bullet \xrightarrow{w_{32}} \bullet$

Cycle stations can store only a limited number of cycles meaning that on a given day, some of these potential journeys might not be viable. If there are no cycles available for hire at the cycle stations near intermediate stops $\boldsymbol{S_{11}}$, $\boldsymbol{S_{12}}$ and $\boldsymbol{S_{22}}$ (say) then cycling from these intermediate stops on these routes is not an option.

A 'smart' solution to this problem would integrate the real-time information services offered by the different public-transport service providers involved, informing us about arrival times of buses, trams, underground trains, and the real-time cycle tracking service, keeping subscribers to the cycle-hire scheme informed about the number of cycles available at each station.

Location-tracking services play a role in this scenario because it is not sufficient to compute a best route at the start of the journey and not revisit this decision *en route*. If a downstream cycle station becomes depleted while the journey is underway then it is important to be aware of this. We would like the systems which we use to be *locally adaptive* as well as *collectively adaptive*. Knowing that downstream options are no longer viable may promote a possible choice to being the only choice.

In order for it to be possible to compute results from our model, we must determine model parameters by estimating concrete values for journey times whether journeys are made by bus, tram, underground train, cycling or walking. Fortunately, in our data-rich times this information is readily available from a variety of web-based sources and we have been able to find all of the model parameters which we need for our example.

A more comprehensive treatment of all aspects of this scenario should also consider additional compilations which we do not address here. As with all service-oriented computing, we should consider the possibility of lack-of-service for all of the services which the system depends upon. The actor in our story, M, may be unable to connect to the real-time bus information service because no 3G connection is available. Location-tracking services may be unavailable because of an occluded GPS signal. The cycle-hire tracking service may be unable to respond to our request for information because of excessive load on the server, software failures, network failures, a period of maintenance activity, or a host of other reasons. Failures are ubiquitous in distributed and service-oriented systems, so it is necessary to represent them in our models. We are aware that the model which we present in this paper misses many other sources of complexity in real-time-informed travel such as these.

We would also like our algorithm to prefer cycle stations where more cycles are available. It might at first seem that the number of cycles which are available should not play a role in the decision of which route to take: it is enough to know whether some are available, or none. However, there are at least two complicating factors. The first complication is that some of the cycles, although present, might not be usable because of flat tyres, missing saddles, damaged wheels, or other reasons. Cycle stands at cycle stations report whether a cycle is attached to the stand, but have no way of knowing whether or not the cycle is usable. The second complication is that CAS are *resource-sharing* systems. Other passengers, and other pedestrians, are also borrowing cycles concurrently, so a small supply of cycles might be depleted by the time that the bus, tram, or underground train has made its journey to the cycle station. For these, and other reasons, the *number* of cycles available is significant, not just the presence or absence of cycles.

Similarly, we should prefer those routes which offer more cycle stations, because this maximises the number of options which remain open to us once we have committed to a particular route, but we do not address this here.

4 Model

Our high-level representation of the system in Figure 1 is not yet in a form which is suitable for analysis. The reason for this is that although we have detailed durations and dependencies, we have not yet clarified the decision which the user has to make when they have to choose between cycling, or walking, or continuing to travel to the next cycle station (if there is a feasible cycle station further along the route).

If at the end of the route (say at stop $\boldsymbol{S_{12}}$) then there is a two-way choice between cycling and walking. We can represent this choice by saying that the previous part-journey had two possible outcomes, leading to committing to cycling ($\boldsymbol{S^c_{12}}$ is reached with probability p^c_{12}) or committing to walking ($\boldsymbol{S^w_{12}}$, reached with probability $p^w_{12} = (1-p^c_{12})$). Committing to cycling may incur a delay while waiting for a cycle to be returned by another user.

If instead there is another cycle station further along the route, then the user has a three-way choice, which we represent as being between states (for example $\boldsymbol{S^c_{11}}$, $\boldsymbol{S^u_{11}}$ and $\boldsymbol{S^w_{11}}$ representing being at location $\boldsymbol{S_{11}}$, having committed to travel the next part of the route by cycling, underground train, or walking respectively). These are reached with probabilities p^c_{11}, p^u_{11} and p^w_{11}, summing to 1. Figure 2 shows how these probabilities play a role in determining the journey to the destination.

Our next challenge in modelling arises from the fact that we have populations of users of the cycle-sharing scheme, and populations of cycles which can be borrowed. Cycle stations may vary from having the capacity to store as few as 10 cycles, or as many as 100. Without cycle stations having a finite capacity, and without a population of users concurrently borrowing cycles, the aspect of competition for resources which is a defining aspect of resource-sharing collective systems would not be captured in the model.

Cycle stations consist of an array of cycle stands, each of which is a simple process recording the presence or absence of a cycle at this stand, together with the activities which cause a change of state. Figure 3 illustrates the idea.

This modelling decision incorporates the simple but powerful abstraction that individuals in a population are identityless. One cycle in a cycle-sharing scheme is treated as being just like any other: each only represents the capacity to allow M to complete his journey more quickly than he would if he was walking. Similarly, the individual identities of the users of the cycle-hire scheme is not important in this modelling context. Each user just represents the potential to remove a cycle which was previously available for hire and hence possibly force

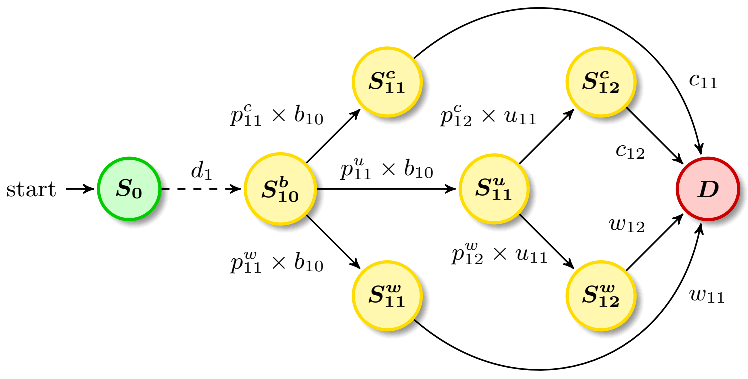


Fig. 2. Exploring route number #1 in greater detail

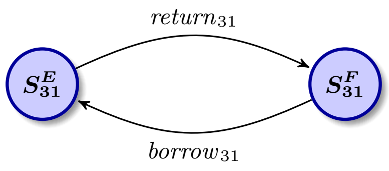


Fig. 3. A cycle station has several stands, each of which may be empty or full

the outcome that M must complete his journey by walking (if there are no cycles left to borrow at the chosen cycle station).

Appendix A presents the complete PEPA model for this scenario.

5 Analysis

We encoded our model in the stochastic process algebra PEPA [1] and analysed it with the PEPA Eclipse Plug-in [2], a modelling tool developed in the European project SENSORIA (Software Engineering for Service-Oriented Overlay Computers) and subsequently used in teaching and research internationally.

PEPA is a compact formal modelling language which provides the appropriate abstract language constructs to represent the model in our example. It has stochastically-timed activities which can be used to encode activities which take time to complete, such as travelling between intermediate stops in a journey and a probabilistic choice operator to express the likelihood of taking different routes. Different patterns of behaviour are encoded in recursive process definitions. Features such as these are found in many modelling formalisms [3] but a distinctive strength of the PEPA language is that populations of components, encoded as arrays of process instances, are both convenient to express in the language and efficiently supported by the dynamic analysis which reveals the collective behaviour which emerges from the interactions of the populations of components. The PEPA language has found application in many modelling problems such as scalable and quantitative analysis of web-services [4–6], comparing communications protocols [7], response-time analysis of safety-critical systems [8], software performance engineering with UML-based models [9, 10], software patterns [11], software architecture [12], signalling pathways [13], model-driven development [17], and robot movement [18].

Many of these models would have been impossible to construct without an efficient method of analysing large-scale population-based models. A mapping from the PEPA language to systems of ordinary differential equations is presented in [19], making these analyses possible. A formal semantic account of the transformation is available [20], together with supporting theory enabling the definition of reward structures on top of the underlying fluid model [21].

These efficient analysis methods are implemented in the PEPA Eclipse Plug-in which provides an integrated modelling environment for PEPA. It incorporates a custom editor for PEPA models, model visualisation and static analysis tools, a model debugger, Markov chain analysis tools, stochastic simulation and discrete analysis tools, a model compiler which delivers a continuous representation of the system, efficient ODE-based solvers, and plotting functions for analysis results.

6 The Optimisation Problem

There are several possible optimisation problems which could be of interest to the traveller in our story, many of which depend heavily on the choice of which route to take at the outset of the journey because this makes a commitment to certain cycle stations.

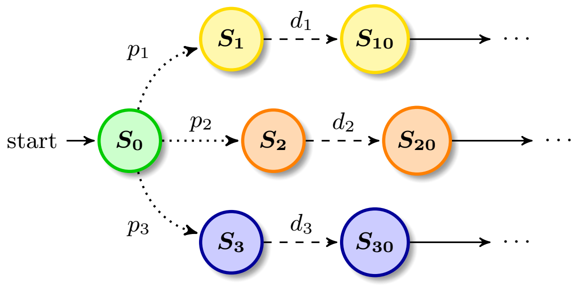


Fig. 4. The optimisation problem: choose p_1, p_2 and p_3 to minimise the time to travel to the destination.

Figure 4 depicts one aspect of the optimisation problem which we intend to solve. We assume that all model parameters except p_1, p_2, and p_3, are known. In practice, we may assume that these other parameters are inferred from measurements on the real system. Indeed, journey times were set using data collected from the Google Maps and the MVV (Munich's public transportation provider) websites. The probabilities related to traveller commitment were arbitrarily fixed. Here we present results with varying configurations of the cycle stations. Our problem is to find the optimal values of p_1, p_2, and p_3 to minimise the average journey time of a traveller wishing to start a journey at location $\boldsymbol{S_0}$. We envisage this optimisation problem to be solved by a service provider which computes the optimal route, given the current conditions of the system. The solution can be interpreted as a *randomised algorithm*: for instance, p_1 represents the probability with which the service provider suggests to go to Hirschauer Strasse. This implicitly guarantees some balancing in the system — if all requests returned the same route, this would introduce contention for shared cycles along the route to which the traveller commits.

We solved the optimisation problem by means of genetic algorithms; in particular we used the implementation available in Matlab R2013b, with its default settings. Figure 5 shows the results of the optimisation problem for three different load conditions on the cycle sharing system, characterised by the ratio between the number of users and the number of cycles available. For simplicity, we fixed the same capacity for all cycle stations.

Figure 5(a) plots the best and mean fitness values in the situation where the system has 10 users per available cycle. The optimal configuration suggests a preferential choice for route through Tivolistrasse, with probability 0.57. The average journey time is ca. 1 hour in this case.

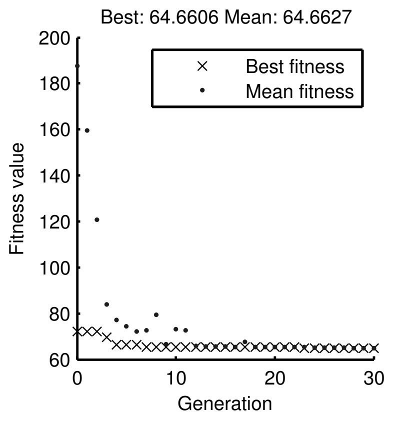
Figure 5(b) shows the results for a less loaded cycle sharing system, where there are 5 users per available cycle. Although the optimal configuration is the same as in Figure 5(a), we observe that the average journey time is substantially reduced; this is explained by a lower contention for bikes at the cycle stations, leading to a higher probability that a traveller will find a cycle as soon as they arrive.

Finally, Figure 5(c) shows an ideal situation with no contention for bikes in the network. This leads to a slight improvement in the average journey time experienced by the user; more interestingly, the algorithm suggests a substantial preference for the route through $\boldsymbol{S_{10}}$, unlike the previous cases.

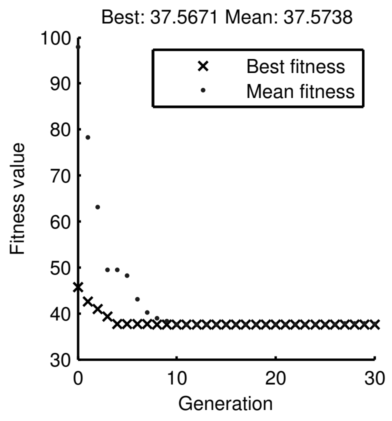
7 Conclusions

Systems which are built as compositions of services are ubiquitous. The ability to consume services provided by others and to compose services from different sources plays a crucial role in the design and evolution of the systems of today. Services make systems work.

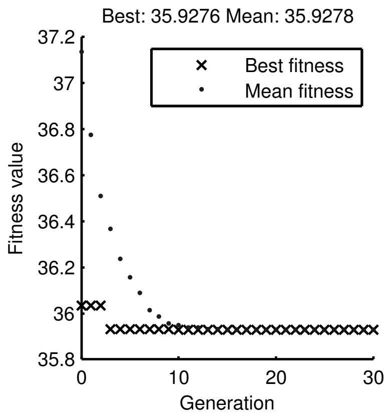
In part the impetus towards these kinds of service composition architectures has been fuelled by a change in attitude towards open systems and open data.



(a) $0.20, 0.23, 0.57$



(b) $0.20, 0.23, 0.57$



(c) $0.97, 0.01, 0.02$

Fig. 5. Optimisation results. Best and mean fitness values (i.e., average journey time, measured in minutes) against generation for three different conditions of the bike sharing system. (a) 10 users per bike; (b) 5 users per bike; (c) 1 user per bike. Each caption shows the optimal configuration of p_1, p_2, and p_3, respectively.

Transport operators in particular have embraced the challenges of providing open access to their data about the state of their service, allowing others to build apps which give users access to information about journey times, disruptions, availability of cycles, and other key system descriptors. More recently apps and applications have been developed which aggregate disparate sources of data to give richer insights and open up new possibilities, as in the scenario considered here where bus and cycle services are integrated.

Possessing the ability to efficiently analyse such service-oriented systems is equally important. Advances in analysis tools and frameworks are needed to keep pace with the ever-increasing challenges which stem from the complex systems which surround us in our technology-dense lives.

Acknowledgements This work is supported by the EU project QUANTICOL, 600708.

References

1. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
2. M. Tribastone, A. Duguid, and S. Gilmore. The PEPA Eclipse Plug-in. *Performance Evaluation Review*, 36(4):28–33, March 2009.
3. Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribastone. *Formal Methods for Performance Evaluation: the 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007*, volume 4486, chapter Stochastic Process Algebras, pages 132–179. Springer-Verlag, Bertinoro, Italy, May–June 2007.
4. Stephen Gilmore and Mirco Tribastone. Evaluating the scalability of a web service-based distributed e-learning and course management system. In Mario Bravetti, Manuel T. Núñez, and Gianluigi Zavattaro, editors, *Third International Workshop on Web Services and Formal Methods (WS-FM'06)*, volume 4184 of *Lecture Notes in Computer Science*, pages 156–170, Vienna, Austria, 2006. Springer.
5. Mario Bravetti, Stephen Gilmore, Claudio Guidi, and Mirco Tribastone. Replicating web services for scalability. In G. Barthe and C. Fournet, editors, *Proceedings of the Third International Conference on Trustworthy Global Computing (TGC'07)*, volume 4912 of *LNCS*, pages 204–221. Springer-Verlag, 2008.
6. Igor Cappello, Allan Clark, Stephen Gilmore, Diego Latella, Michele Loreti, Paola Quaglia, and Stefano Schivo. Quantitative analysis of services. In Wirsing and Hölzl [22], pages 522–540.
7. Hao Wang, Dave Laurenson, and Jane Hillston. Evaluation of RSVP and Mobility-aware RSVP Using Performance Evaluation Process Algebra. In *Proceedings of the IEEE International Conference on Communications (ICC 2008)*, Beijing, China, May 2008.
8. Ashok Argent-Katwala, Allan Clark, Howard Foster, Stephen Gilmore, Philip Mayer, and Mirco Tribastone. Safety and response-time analysis of an automotive accident assistance service. In *Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008)*, number 17 in Communications in Computer and Information Science (CCIS), pages 191–205, Porto Sani, Greece, October 2008. Springer-Verlag.
9. Mirco Tribastone and Stephen Gilmore. Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In *Proceedings of the 7th International Workshop on Software and Performance (WOSP2008)*, pages 67–78, Princeton NJ, USA, 2008. ACM Press.
10. Mirco Tribastone and Stephen Gilmore. Automatic translation of UML sequence diagrams into PEPA models. In *5th International Conference on the Quantitative Evaluation of SysTems (QEST 2008)*, pages 205–214, St Malo, France, 2008. IEEE Computer Society Press.

11. Martin Wirsing, Matthias M. Hölzl, Lucia Acciai, Federico Banti, Allan Clark, Alessandro Fantechi, Stephen Gilmore, Stefania Gnesi, László Gönczy, Nora Koch, Alessandro Lapadula, Philip Mayer, Franco Mazzanti, Rosario Pugliese, Andreas Schroeder, Francesco Tiezzi, Mirco Tribastone, and Dániel Varró. SENSORIA patterns: Augmenting service engineering with formal analysis, transformation and dynamicity. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings*, volume 17 of *Communications in Computer and Information Science*, pages 170–190. Springer, 2008.

12. Alexander Knapp, Stephan Janisch, Rolf Hennicker, Allan Clark, Stephen Gilmore, Florian Hacklinger, Hubert Baumeister, and Martin Wirsing. Modelling the CoCoME with the JAVA/A component model. In Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and Frantisek Plasil, editors, *The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]*, volume 5153 of *Lecture Notes in Computer Science*, pages 207–237. Springer, 2007.

13. Nil Geisweiller, Jane Hillston, and Marco Stenico. Relating continuous and discrete PEPA models of signalling pathways. *Theor. Comput. Sci.*, 404(1-2):97–111, 2008.

14. Yishi Zhao and Nigel Thomas. Approximate solution of a PEPA model of a key distribution centre. In Samuel Kounev, Ian Gorton, and Kai Sachs, editors, *Performance Evaluation: Metrics, Models and Benchmarks, SPEC International Performance Evaluation Workshop, SIPEW 2008, Darmstadt, Germany, June 27-28, 2008. Proceedings*, volume 5119 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2008.

15. Jeremy T. Bradley, Stephen Gilmore, and Jane Hillston. Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models. *J. Comput. Syst. Sci.*, 74(6):1013–1032, 2008.

16. Adam Duguid. Coping with the parallelism of BitTorrent: Conversion of PEPA to ODEs in dealing with state space explosion. In Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings*, volume 4202 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2006.

17. Stephen Gilmore, László Gönczy, Nora Koch, Philip Mayer, Mirco Tribastone, and Dániel Varró. Non-functional properties in the model-driven development of service-oriented systems. *Software and System Modeling*, 10(3):287–311, 2011.

18. Allan Clark, Adam Duguid, and Stephen Gilmore. Passage-end analysis for analysing robot movement. In Wirsing and Hölzl [22], pages 506–521.

19. J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, pages 33–43, Torino, Italy, September 2005. IEEE Computer Society Press.

20. Mirco Tribastone, Stephen Gilmore, and Jane Hillston. Scalable differential analysis of process algebra models. *IEEE Trans. Software Eng.*, 38(1):205–219, 2012.

21. Mirco Tribastone, Jie Ding, Stephen Gilmore, and Jane Hillston. Fluid rewards for a stochastic process algebra. *IEEE Trans. Software Eng.*, 38(4):861–874, 2012.

22. Martin Wirsing and Matthias M. Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, volume 6582 of *Lecture Notes in Computer Science.* Springer, 2011.

A PEPA Model

This section contains the complete PEPA model for the scenario described in Section 4. It is presented with the syntax accepted by the PEPA Eclipse Plug-in.

```
// Probabilities of choosing routes.
p_1 = 0.20;
p_2 = 0.23;
p_3 = 0.57;

// Probabilities of cycling, walking or bussing from S11
p_11_c = 0.45;
p_11_w = 0.15;
// Ensure p_11_c+p_11_w+p_11_b = 1
p_11_u = 1 - (p_11_c + p_11_w);

// Probabilities for S12
p_12_c = 0.385;
p_12_w = 1 - p_12_c;

// Probabilities for S22
p_22_c = 0.35;
p_22_u = 1 - p_22_c;

// Probabilities for S23
p_23_c = 0.3853232;
p_23_w = 1 - p_23_c;

// Probabilities for S31
p_31_c = 0.55;
p_31_w = 0.275;
p_31_b = 1 - (p_31_c + p_31_w);

// Probabilities for S32
p_32_c = 0.525;
p_32_w = 1 - p_32_c;

// Rate parameters: unit time is minutes
t = 1; // Think time

// Delays waiting for the buses
d_1 = 5;
d_2 = 8;
d_3 = 12;


b_10 = 17; // Bus time
t_30 = 10; // Tram time

// Cycle times
c_11 = 10;
c_12 = 7;
c_22 = 7;
c_23 = 9;
c_31 = 12;
c_32 = 15;

// Walking times are roughly twice the cycle times
w_11 = 20;
w_12 = 14;
w_20 = 6;
w_21 = 10;
w_23 = 18;
w_31 = 24;
w_32 = 30;

// Underground times
```

```
u_11 = 11;
u_22 = 14;
u_32 = 17;

// Return and borrow from S11
r_r_11 = 0.21431;
r_b_11 = 0.8239457;

// Return and borrow from S12
r_r_12 = 0.21431;
r_b_12 = 0.8239457;

// Return and borrow from S22
r_r_22 = 0.21242321;
r_b_22 = 0.2822223294527;

// Return and borrow from S23
r_r_23 = 0.212431;
r_b_23 = 0.8222329457;

// Return and borrow from S31
r_r_31 = 0.73453;
r_b_31 = 0.21348;

// Return and borrow from S32
r_r_32 = 0.37845;
r_b_32 = 0.13534;

// Return and borrow from D
r_r_D = 0.37845;
r_b_D = 0.13534;

// Work
w = 1;

User_0 = (choose, p_1/t).User_1
       + (choose, p_2/t).User_2
       + (choose, p_3/t).User_3;

// User taking the #1 route
User_1 = (delay_1,d_1).User_10;

User_10 = (bus_to_11, p_11_c*b_10).User_c_11
        + (bus_to_11, p_11_w*b_10).User_w_11
        + (bus_to_11, p_11_u*b_10).User_u_11;
User_c_11 = (borrow_11, r_b_11).(cycle_from_11, c_11).(return_D, r_r_D).User_W;
User_w_11 = (walk_from_11, w_11).User_W;
User_u_11 = (underground_to_12,p_12_c * u_11).User_c_12
          + (underground_to_12,p_12_w * u_11).User_w_12;
User_c_12 = (borrow_12, r_b_12).(cycle_from_12, c_12).(return_D, r_r_D).User_W;
User_w_12 = (walk_from_12, w_12).User_W;

// User taking the #2 route
User_2 = (delay_2,d_2).User_20;

User_20 =  (walk_to_21, w_20).User_w_21;
User_w_21 = (walk_from_21, p_22_c * w_21).User_c_22
          + (walk_from_21, p_22_u * w_21).User_u_22;
User_c_22 = (borrow_22, r_b_22).(cycle_from_22, c_22).(return_D, r_r_D).User_W;
User_u_22 = (underground_from_22, p_23_c * u_22).User_c_23
         + (underground_from_22, p_23_w * u_22).User_w_23;
User_c_23 = (borrow_23, r_b_23).(cycle_from_23, c_23).(return_D, r_r_D).User_W;
User_w_23 = (walk_from_23, w_23).User_W;

// User taking the #3 route
User_3 = (delay_3,d_3).User_30;

User_30 = (tram_to_31,p_31_c * t_30).User_c_31
```

```
      + (tram_to_31,p_31_w * t_30).User_w_31
      + (tram_to_31,p_31_b * t_30).User_u_31;

User_c_31 = (borrow_31, r_b_31).(cycle_from_31, c_31).(return_D, r_r_D).User_W;
User_w_31 = (walk_from_31, w_31).User_W;
User_u_31 = (underground_to_32,p_32_c * u_32).User_c_32
          + (underground_to_32,p_32_w * u_32).User_w_32;
User_c_32 = (borrow_32, r_b_32).(cycle_from_32, c_32).(return_D, r_r_D).User_W;
User_w_32 = (walk_from_32, w_32).User_W;

// User at work
User_W = (work, w).User_0;

// Definitions of stations
Station_11_Empty = (return_11, r_r_11).Station_11_Full;
Station_11_Full = (borrow_11, r_b_11).Station_11_Empty;

Station_12_Empty = (return_12, r_r_12).Station_12_Full;
Station_12_Full = (borrow_12, r_b_12).Station_12_Empty;

Station_22_Empty = (return_22, r_r_22).Station_22_Full;
Station_22_Full = (borrow_22, r_b_22).Station_22_Empty;

Station_23_Empty = (return_23, r_r_23).Station_23_Full;
Station_23_Full = (borrow_23, r_b_23).Station_23_Empty;

Station_31_Empty = (return_31, r_r_31).Station_31_Full;
Station_31_Full = (borrow_31, r_b_31).Station_31_Empty;

Station_32_Empty = (return_32, r_r_32).Station_32_Full;
Station_32_Full = (borrow_32, r_b_32).Station_32_Empty;

Station_D_Empty = (return_D, r_r_D).Station_D_Full;
Station_D_Full = (borrow_D, r_b_D).Station_D_Empty;

// Definitions for other users
User_11_Idle = (borrow_11, r_b_11).User_11_Busy;
User_11_Busy = (return_11, r_r_11).User_11_Idle;

User_12_Idle = (borrow_12, r_b_12).User_12_Busy;
User_12_Busy = (return_12, r_r_12).User_12_Idle;

User_21_Idle = (borrow_21, r_b_21).User_21_Busy;
User_21_Busy = (return_21, r_r_21).User_21_Idle;

User_22_Idle = (borrow_22, r_b_22).User_22_Busy;
User_22_Busy = (return_22, r_r_22).User_22_Idle;

User_23_Idle = (borrow_23, r_b_23).User_23_Busy;
User_23_Busy = (return_23, r_r_23).User_23_Idle;

User_31_Idle = (borrow_31, r_b_31).User_31_Busy;
User_31_Busy = (return_31, r_r_31).User_31_Idle;

User_32_Idle = (borrow_32, r_b_32).User_32_Busy;
User_32_Busy = (return_32, r_r_32).User_32_Idle;

User_D_Idle = (borrow_D, r_b_D).User_D_Busy;
User_D_Busy = (return_D, r_r_D).User_D_Idle;

(User_0[60] <> User_11_Idle[1] <> User_12_Idle[1] <>
 User_21_Idle[1] <> User_22_Idle[1] <> User_23_Idle[1] <>
  User_31_Idle[1] <> User_32_Idle[1] <> User_D_Idle[1]) <*> (
   Station_11_Full[2] <> Station_12_Full[2] <>
    (Station_22_Full[2] <> Station_23_Full[2]) <>
      (Station_31_Full[2] <> Station_32_Full[2] <> Station_D_Full[2]) )
```

Lines 2–4 define the probabilities of choosing routes. These are the variables in our optimisation problem; here they are set to the optimal values for the scenarios illustrated in Figures 5(a) and 5(b). Using the same notation as in the main text, lines 6–93 define all the other model parameters. Journey times were inferred from information available on the web, as discussed; the remaining parameters were arbitrarily fixed. Traveller behaviour is modelled in lines 98–142. The dynamics of a cycle station is characterised by a two-state automaton associated with each docking point, lines 145-164. We consider exogenous arrivals and departures to each cycle station by modelling further users, lines 1767–189. Finally, lines 191–296 defines the *system equation*, specifying the total population of users and the number of docking points for each cycle station. From [21], we compute the average journey time experienced by a user using Little's law as

$$\text{Average Journey Time} = \frac{60}{\mathsf{User\_0} \times w},$$

where the numerator gives the total number of users of interest and User_0 gives the total number of users in the steady state which are about to start their journey.