Early Model-Analysis of Logistics Systems*

Freeha Azmat[1], Laura Bocchi[2], and José Luiz Fiadeiro[2]

[1] Department of Engineering, Bahria University, Islamabad, PK
freehaazmat@bahria.edu.pk

[2] Department of Computer Science, University of Leicester, UK
{bocchi,jose}@mcs.le.ac.uk

Abstract. In logistics, as in many other business sectors, service-oriented architectures (SOAs) are offering the possibility for applications to interact with each other across languages and platforms, and for external services to be procured automatically through dedicated middleware components. In this setting, one of the critical business aspects that need to be supported is the negotiation of non-functional quantitative aspects, such as costs, leading to service-level agreements (SLAs) between business parties. In this paper, we present a formal, probabilistic approach through which services can be analysed in relation to the probability that a quality of service property is satisfied.

1 Introduction

Business integration through Web-service infrastructures is becoming prevalent in business sectors that, like Logistics, depend on the procurement of services from external providers to deliver their own services. Service-oriented architectures (SOAs) support a new generation of business processes that can involve outsourced functionalities discovered at run time, automating the procurement of services so as to optimize quantitative quality-of-service (QoS) properties [15] that are essential for their business operation. For this purpose, the dynamic discovery and selection of services needs to rely on the negotiation of service-level agreements (SLAs) between providers and requestors. In this scenario, it is critical that providers be able to guarantee the non-functional properties of the services that they advertise. It is also critical that service requesters can decide which requirements to specify on the services that they need to procure in order to meet given business goals.

To this aim, we propose a formal approach for the quantitative analysis, at the early stages of systems design, of service-oriented applications that allows developers to tune QoS properties offered or required. A number of approaches feature quantitative analysis in the early stages of system design in order to prevent that design choices affect the deployment of the system in a negative way (see [3] for an overview); however they are not specifically targeted to SOAs.

In this paper, we extend the approach that we developed in [6] for the analysis of time-related properties of service models, in order to address a wider range of QoS properties involved in the provision of logistic services. The quality of a logistic

* This work has been partially sponsored by the Leverhulme Trust Award "Tracing Networks".

system is typically described in terms of a number of different QoS indicators [10,17]. For example, when specifying the model for a logistics cycle (e.g., involving the integration of different functionalities such as information, transportation, inventory, warehousing, material handling, and packaging), one may need to make sure that the cost of the overall business process does not exceed a given threshold, or that the cost of the service being modelled does not exceed the advertised price.

The proposed approach is presented in four steps:

1. We use a simple product-supply service to present a language for modelling services as compositions of internally-provided and outsourced functionalities.
2. We illustrate how the models produced in this language can be annotated with rates that capture cost indicators. In Section 6 we discuss how our approach can be used for other QoS indicators (e.g., *throughput*, *scalability* or *reliability*) that satisfy certain requirements.
3. We show how quantitative analysis of the annotated model can be performed.
4. We show how the feedback obtained from the analysis can be used for improving the original model so that overall non-functional constraints can be guaranteed.

The modelling language that we use in step (1) is SRML – the Sensoria Reference Modelling Language [9] – which was developed in order to capture and formalize foundational aspects of SOAs, including service composition, dynamic binding and reconfiguration, and service level agreements. SRML follows the basic principles and structures put forward by SCA [20], a middleware-independent framework proposed by an industrial consortium for building business applications over SOAs. Whereas SCA focuses on the assembly and deployment of service-oriented software artefacts, SRML offers high-level primitives for business modelling, from component interoperability to business integration.

As a case study, we consider a logistics service for supplying products of a certain kind that finds the warehouse closest to the customer if the chosen product is not available in a local stock [11]. An example of an SLA constraint that we would like to certify is whether in 85% of the cases the modelled business process is able to maintain the costs of the transaction lower than, say, \$6. As could be expected, the overall cost depends on the costs of the services that may be discovered at run-time.

In step (2), we annotate SRML models with rates representing a non-functional property. The resulting models are then encoded into the Performance Evaluation Process Algebra (PEPA) [12] so that they can be analysed. PEPA has a formal semantics and is supported by a range of powerful analysis tools [22]. Consistency properties of the encoding have been proved in [6] that show that the encodings preserve the structure of the original SRML models which, together with the fact that the encoding is defined in a modular way, makes it straightforward to establish a correspondence between interactions in PEPA models and events in SRML models. This is particularly useful for obtaining immediate feedback on the original model from the quantitative analysis of the PEPA encoding.

1.1 Related Work

Quantitative model analysis. In [1] the authors propose an approach for stochastic analysis of logistic models based on Petri-nets. SRML supports a higher level of abstraction and offers domain-specific primitives for modelling business processes. In [13,14] Iacon and Jonkers propose a layered analysis approach covering technical infrastructures, software applications, business processes and products; they focus on performance analysis taking into account inter-relationships between the different layers. Our approach considers two layers – architectural and behavioural – and focuses on properties that arise from the dynamic aspects of service-oriented systems, namely discovery and loose coupling, which are key for flexible architectures.

Analysis of logistic systems. [7] compares agent-based and equation-based approaches to the modelling of logistic services, and [24] uses an equational model to analyse the adjustment of costs in logistic systems. Our approach proceeds top-down by first creating a high-level model and then deriving a corresponding lower-level PEPA (i.e., equation-based) system. Due to the modularity of the encoding (see Section 4) the feedback obtained from the PEPA system can be straightforwardly applied to the re-engineering of the high-level model.

QoS description and run-monitoring of conformance. [8,23] present methods for the evaluation of the best match among the available logistic services considering multiple properties that are possibly related with each other. [25] presents a semantic description model for QoS and proposes a solution for dynamic assessment, management and ranking of QoS values based on user feedback. [16] proposes a method for service evaluation where reputation is a function of user ratings, quality compliance and verity (i.e., the changes of service quality conformance over time). In [18] services are rated in terms of their quality, with QoS information provided by monitoring services and users. [19,21] use mainly third-party service brokers or specialized monitoring agents to collect performance of all available services in registries, which would be expensive in reality. Instead, our approach focuses in analysing that such properties are consistent with a model. In addition, we address quantitative analysis at the early stages of system design in order to prevent that design choices affect the deployment of the system in a negative way.

2 Modelling a Supply Service with SRML

The unit of design in SRML is the *module*, through which business processes are defined as orchestrations of tightly-coupled components that are locally implemented and loosely-coupled dynamically-discovered services. Figure 1 illustrates the structure of the module *Product Supply* that, according to customer preferences, checks for the availability of a product in a local stock or outsources it from an external warehouse that is discovered and selected according to given SLAs. The module includes: (1) a *provides-interface CR* describing the interface that the service offers to customers; (2) a *uses-interface ST* describing the interface of a local database that keeps record of the products in stock; (3) two *requires-interfaces*, *DL* and *WR*, that specify the interfaces to services that may need to be procured on the fly from external parties – a delivery service and a warehouse, respectively; (4) a *component-interface OS* describing the interface of the component that orchestrates the parties that, together, deliver the service; and (5) a number of *wires*, *CO*, *OT OD* and *OW*, that specify interaction protocols between those parties. Whereas *ST* is part of the resources that are

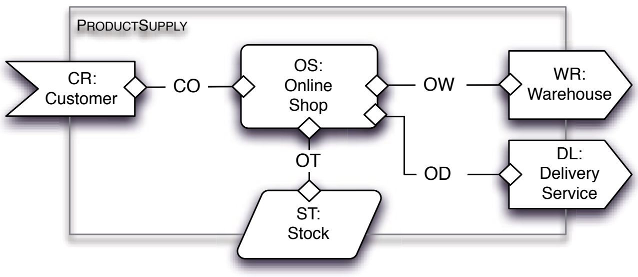

Fig. 1. SRML Module structure of service *Product Supply*

local to *Product Supply*, *WR* and *DL* are discovered at run time only if and when they are necessary (when the product is not in stock) depending on information that is not known a-priori such as the specific product that the customer needs, the quantity, the location for delivery, and so on.

Formally, a SRML module *M* consists of

- A set *nodes(M),* which is the union of the following four disjoint sets:
  - *provides*(M) consisting of a single provides-interface,
  - *uses*(M) consisting of uses-interfaces,
  - *requires*(M) consisting of requires-interfaces,
  - *components*(M) consisting of component-interfaces,
- A se*t edges(M)* consisting of (*wire-interfaces*) where each edg*e w* is a set of two node*s w: m*↔*n.*
- *A* labelling function $label_M$ that associates a behavioural specification with every node, as discussed in Section 2.1.
- An internal configuration policy that specifies initialisation conditions for component-interfaces and triggers for the discovery of external services.
- An external configuration policy *cs(M)* that describes the SLA properties of *M*, as discussed in Section 2.2.

2.1 Modelling the Behaviour of Services

The behavioural properties of the service modelled through a module results from the specifications of all its nodes and wires. Each specification consists of a signature (the set of interactions that the entity can engage in) and a behavioural protocol. In *Product Supply*, *Customer* denotes the specification of the interface *CR*, *Warehouse* that of *WR*, and so on.

Different formalisms are used in SRML for defining the behavioural protocols of each node depending on the nature of that node. It is out of the scope of this paper to present these specification formalisms, which can be found in [9]. For our purposes, it is sufficient to consider that each of the languages relies on the same computational model, which is based on the notion of *interaction event* discussed below.

SRML supports a number of interaction types to reflect both the interactions occurring within an enterprise and the typical business conversations that arise in SOC [4]. More specifically, interactions can be synchronous (i.e., the party waits for the co-party to reply), or asynchronous (i.e., the party does not block). Typically, synchronous interactions occur in communications with persistent components, namely through uses-interfaces. We distinguish the following types of interactions, each described from the point of view of the party in which it is declared:

- ***snd*** and ***rcv*** are one-way asynchronous (send or receive) interactions.
- ***s&r*** and ***r&s*** are conversational asynchronous interactions (***s&r*** after "send-and-receive" is initiated by the party, and ***r&s*** is initiated by the co-party).
- ***ask*** and ***tll*** (resp. ***rpl*** and ***prf***) are synchronous interactions to obtain data (resp. to ask to perform an operation).

One-way interactions (***snd*** and ***rcv***) have an associated initiation event denoted by *interaction*🔔. Conversational interactions (***s&r*** and ***r&s***) involve a number of events exchanged between the two parties: once the initiation event *interaction*🔔 is issued by a party, a reply-event *interaction*✉ is sent by the co-party offering a deal; the party may then either commit to the deal (issuing *interaction*✓) or terminate the conversation (issuing *interaction*×); after committing, the party can still revoke the deal (issuing *interaction*✞), triggering a compensation. In this paper, we abstract from the parameters exchanged in the interactions, as they are not relevant for the analysis.

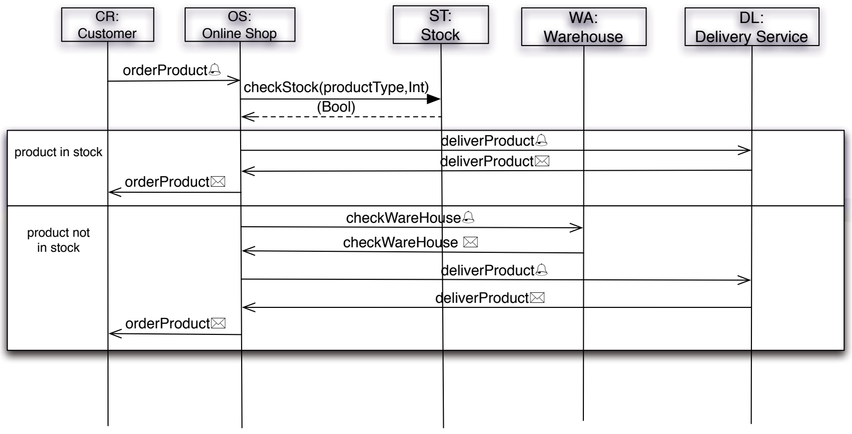

Fig. 2. Overview of the behaviour of module *Product Supply*

The behaviour of *Product Supply* is illustrated in Figure 2. The interaction *orderProduct* allows *CR* to ask the service to supply a product and receive a confirmation; the orchestrator *OS* checks for the product in the local stock with a synchronous interaction *checkStock*; if the product is not available in the local stock, *OS* requests the product to the warehouse service using the conversational interaction *checkWarehouse*. The interaction *deliverproduct* arranges for the product to be dispatched. In this scenario, *OS* may find the product in the stock (case *product in stock*); otherwise, it will contact the warehouse service *WR*. In any case the service will arrange the delivery of the product.

2.2 Modelling SLA Properties

SRML adopts a model for constraints on non-functional requirements in dynamically changing configurations based on c-semirings [5]. The constraints are used for modelling the SLAs involved in service discovery and selection. Our example uses a fuzzy c-semiring where the degree of satisfaction lies in the interval [0,1]. An external configuration policy *cs(M)* consists of: (1) a set of SLA variables $var_{SLA}(M)$, and (2) a set of constraints *SLA(M).* The variables and constraints determine the quality profile to which the provided service and each discovered service need to adhere; they can represent QoS parameters such as cost, response time, delays, *inter alia*. Below we show *cs(Product Supply)*:

EXTERNAL POLICY

SLA VARIABLES

CR.COST,WR.COST,DL.COST,CR.LOCATION, WR.LOCATION

CONSTRAINTS

C_1: *{CR.COST, WR.COST, DL.COST}*

$Def_1(d,e,f)$ = $\begin{cases} 1 \text{ if } d = e + f + 1.6 \text{ and } d \le 6 \\ 0 \text{ otherwise} \end{cases}$

C_2: *{*CR.LOCATION, WR.LOCATION*}*

$Def_3(d,e)$= 1 if **distance**(d,e)<20, 20/**distance**(d,e) otherwise

The set var_{SLA}*(Product Supply)* includes three SLA variables – *CR.Cost*, *WR.Cost*, and *DL.Cost* – representing the transaction costs associated with the customer, the warehouse and the delivery service, respectively. Variables *CR.Location* and *WR.Location* represent the location of the customer and the warehouse. respectively.

The set *SLA(Product Supply)* includes two constraints – C_1 and C_2. C_1 expresses that the cost of the transaction for the customer should not be more than $6 and that it consists of the sum of the cost of the warehouse transaction, the cost of the delivery service and a local cost of $1.6. C_2 minimizes the distance between the customer and the warehouse – *distance(d,e)* is a function returning the distance between two locations. If the distance is less than 20 miles, the satisfaction is 1, otherwise the satisfaction is inversely proportional to the distance.

The aim of the quantitative analysis illustrated in Section 4 is to guarantee that C_1 holds up to a certain probability, for example, the cost of the overall service, defined as the business process occurring between the events *orderProduct*🔔 and *orderProduct*✉, is less than or equal to 6$ in 85% of the cases.

3 Modelling Cost in SRML

In this section, we extend SRML by annotating the different elements of a module with *cost rates.* A cost is considered to be the amount of money that a user will incur in when using a certain node or edge. Cost is modelled as a rate with exponential distribution – other distributions would require knowledge of higher moments and other parameters, which would require careful measurement or estimation; the exponential distribution requires only a single parameter – the average response time – which is more likely to be known.

The rate represents the coefficient *r* of the exponential distribution, which defines the probability for the cost to be lower than a certain value: $F_{Delay(a,b)}(t)=1-e^{-rt}$. Figure 3 shows the exponential distributions for the rates 3.0, 4.0, and 5.0. This diagram was obtained using the PEPA toolset. If, for example, the rate is 3, then the cost will be lower than \$2 in 50% of the cases and will be lower than \$5 in 100% of the cases. Naturally, a higher rate characterises a lower cost.

The Cost Policy for a module *M* includes the following *cost rates*:

- For every provides-interface $n \in provides(M)$: *ProvidesRate(n).*
- For every requires-interface $n \in requires(M)$: *RequiresRate(n).*
- For every $w \in edges(M)$: *WireRate(w).*
- For every $n \in components(M)$: *ComponentRate(n).*
- For every $n \in uses(M)$: *UsesRate(n).*

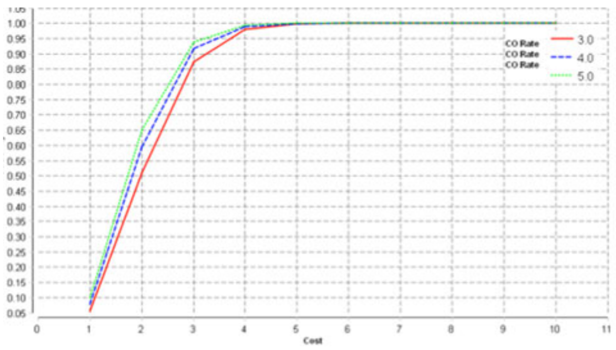

Fig. 3. Relationship between Cost and Rate

WireRate represents the cost a requester would incur to when using the wire. Similarly the cost for using a component-interface, requires-interface, Uses-interface is regarded as *ComponentRate*, *RequiresRate*, and *UsesRate*, respectively.

In order to make analysis more accurate, one may annotate each alternative branch of the sequence diagram representing the overall behaviour of *Product Supply* with the probability of engaging in that branch (where the sum of the probabilities for all branches of the same branching point is equal to 1). For example, we could annotate the sequence diagram of *Product Supply* in Figure 2 to model that the probability that the product is in stock is 0.6.

Once the nodes and wires of a module have been annotated, the aim is to derive, say, the cost of the business process between two events occurring in the provides-interface *CR* (e.g., *orderProduct*🔔 and *orderProduct*🖂) by considering the cascade of costs incurred between those two events. Figure 5 gives the overall cost of the activity between *orderProduct*🔔 and *orderProduct*🖂 (assuming that the product is not in stock), which is denoted with (1) and it is built upon the costs of wire *CO* – (2), component *OS* – (3), wire *OW* – (4), requires interface *WR* – (5), wire *OD* – (6) and requires interface *DL* – (7).

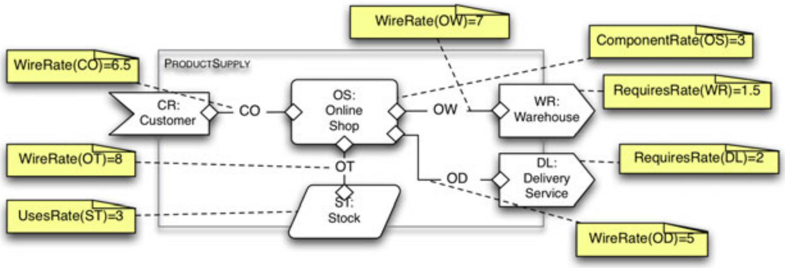

Fig. 4. SRML module annotated with *cost rates*

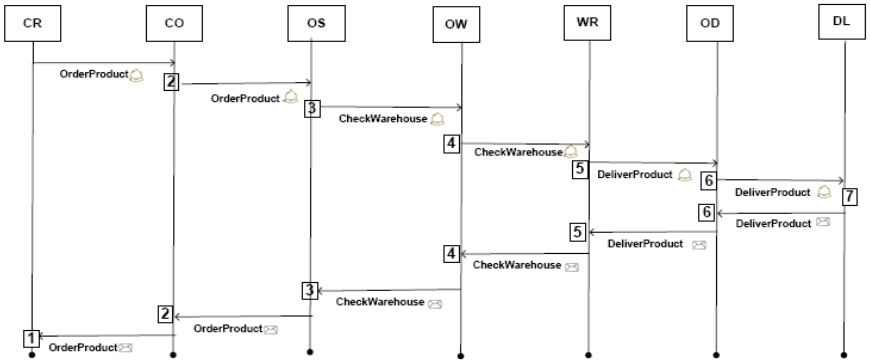

Fig. 5. Cascade of Costs in service *Product Supply*

4 Encoding SRML Modules into PEPA Configurations

SRML modules annotated with cost rates can be encoded in PEPA using a simplification of the encoding presented in [6]. A system is described in PEPA as a configuration involving a number of processes collaborating along a number of channels. Processes perform activities of the form *$(\alpha, r).P$* where α is an action associated with rate *r* and *P* is the continuation. The encoding of *Product Supply* is

```
(CR||OS||WR||ST||DL)<L>(CO||OT||OW||OD)
```

where each process on the left-hand side (e.g., *CR*) corresponds to a node of *Product Supply,* and each process on the right-hand side (e.g., *CO*) corresponds to an edge of *Product Supply*. *L* is called a *coordination se*t and includes the actions corresponding to each event triggered by any node. The encoding is modular in the sense that the process corresponding to each node/edge can then be obtained independently of the encoding of the other elements of the module (for example, the SRML component *CR* is encoded as a PEPA process *CR*, etc.).

The simplified encoding has been developed in [2]. In the case of components/requires-interfaces, where delays affect each interaction with the component/requires-interface, it considers a forfeit cost for using a component or a service and is more concise. We are currently working on the automation of the approach by extending the SRML editor to allow the annotations with rates, and by automating the encoding of annotated SRML models into PEPA models, creating a bridge between the SRML editor and the PEPA toolset.

5 Quantitative Analysis of Cost Properties

We perform quantitative analysis in three steps: (1) We derive the state space of the PEPA encoding to find any deadlock or unreachable state; (2) We perform passage-cost analysis to derive the probability of the cost of the business process inter-occurring between two activities; (3) We perform sensitivity analysis, which allows us to observe how changing the value of rates changes the cost.

As to (1), state-space derivation is necessary to find deadlock states in the system because the deadlocks prevent the analysis to proceed further. Passage-cost analysis cannot be performed until all the deadlock states are removed from the system. State-space analysis of *Product Supply* service modelled 11 elements (i.e., wires and nodes) and 20 deadlock free states resulted from the modelled interaction pattern. The approach scales to larger systems as the state space, in most cases, grows linearly with respect to the number of architectural elements. In fact, the flow of events of a SRML business process typically involves one or two architectural elements a time (i.e., the cases of parallellism are fairly limited in relation to the global number of architectural elements). This relieves us from considering all possible interleavings of events. As to (2), we want to analyse the cost of the business process between the occurrence of the event *orderProduct*🔔 (i.e., the order from the customer) and *orderProduct*✉ (i.e., the system's reply). The aim is to determine whether the system is able to guarantee that the cost, in 85% of the service invocations, is not greater than 6$. As to (3), we use feedback from step (2) to determine that where it best to invest effort in making one of the activities more economic. By changing the values of the rates, we can identify those nodes and wires that cause maximum change in cost.

Passage-Cost Analysis — The feature passage-time analysis provided by the PEPA Eclipse Plug-in is typically used to analyse the performance of a model when the rates represent the delay of each single activity. We use such functionality here to perform "passage cost analysis" by interpreting each rate in the model as a *cost rate* instead of a time rate. Passage-cost analysis produces a graph representing the cumulative distribution function (CDF) that plots the probability of having completed the passage of interest (e.g., *orderProduct*🔔, *orderProduct*✉) by a given cost bound. The CDF graph is illustrated in Figure 6. The black dot in the graph underlines the probability of a maximum cost of \$6. The probability is less than 85%; in other words, in 85% of the cases we can only guarantee a maximum cost of \$6.

Feedback and Reengineering — Sensitivity analysis was performed by changing each cost rate to a fixed number of possible values to see if any improvement could be

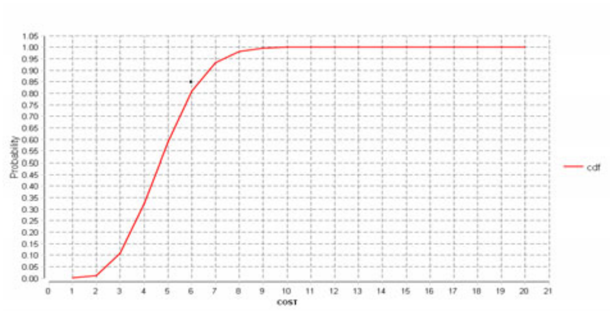

Fig. 6. CDF with initial Rates

found and the advertised SLA be satisfied. The cost rates affiliated with each node and wires are changed. There are four wires involved in the scenario *CO*, *OT*, *OW* and *OD*. The original cost rate of all wires is 3. We have changed the cost rates of every wire to two values below the true value (2,2.5) and two values above the true value (3.5,4). We noticed that all the wires brought the same change and were able to satisfy SLA at cost rate of 4. All the wires reduced the cost of transaction from 6.4$ to 6$. The nodes of the scenario are component-interfaces *OS*, ST, *WR* and *DL*. We have changed the cost rates of each of these nodes and tried to find the critical node. The original cost rate of all nodes is 4. We have changed the cost rates of all nodes two values below the true value (3,3.5) and two values above the true value (4.5,5). Only in the analysis of *OS* it was possible to satisfy the advertised SLA. Changes in the rates of *OS* brought the maximum performance change and reduced the cost from 6.4$ to 6$, as shown in Figure 7.

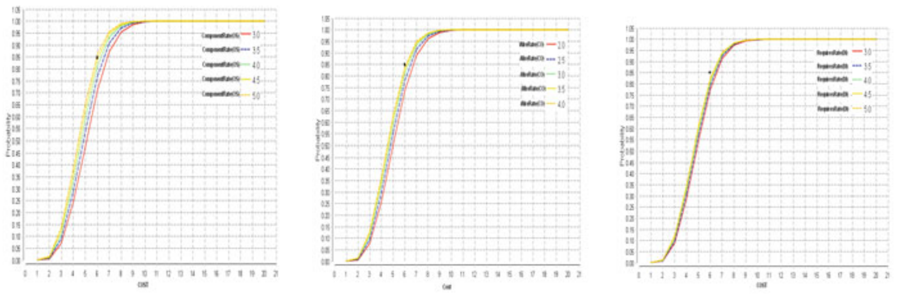

Fig. 7. Sensitivity Analysis of resp. *ComponentRate(OS), WireRate(CO)*, *RequiresRate(DL)*

6 Concluding Remarks

We have proposed an approach for analysing non-functional properties of service-oriented models as it arises in a number of business areas that, like logistics, are adopting service-oriented architectures. For this purpose, we have used the service modelling language SRML and the stochastic process algebra PEPA. We have presented an encoding of a SRML model – *Product Supply* – of a typical logistics service into PEPA, and used the PEPA Eclipse Plug-in to perform quantitative analysis of cost-related properties, guaranteeing that SLAs can be met with a certain probability. We have also performed sensitivity analysis to find those critical components of the service that directly affect advertised SLAs. By finding these critical components, we can know where we should invest more to improve the performance of the system.

The proposed approach can be used for other non-functional properties, more specifically (following from our definition of rate) all non-functional properties that (1) can be represented by an exponential distribution and (2) affect the model as described by the annotation we have provided in Section 3. For example, we can use the proposed method to analyse *throughput.* In this case, *WireRate* would characterise the throughput of a wire (instead of the cost) where the rate represents the coefficient r of the exponential distribution of the form $F_{Delay(a,b)}(t)=1-e^{-rt}$ defining the probability that the throughput is higher than a certain value. In this case, a lower value of rate characterises a preferable setting, therefore a higher throughput. At the moment, the analysis of different properties should be performed independently. On the other hand, one could expect that the cost of a service could be directly proportional to other properties such as throughput. A topic for future development is supporting quantitative analysis of different inter-related properties.

Acknowledgements

We would like to thank Stephen Gilmore, Reiko Heckel for many useful discussions.

References

1. Alves, Maciel, Massa: Evaluating Supply Chains with Stochastic Models similar with Petri Nets. In: SOLI, pp. 1–6. IEEE, Los Alamitos (2007)
2. Azmat: Quantitative Analysis of Service Oriented Models. MSc Project Dissertation, Department of Computer Science, University of Leicester, UK (2010)
3. Balsamo, Di Marco, Inverardi, Simeoni: Model-based performance prediction in software development: a survey. IEEE Trans. on Software Engineering 30(5), 295–310 (2004)
4. Benatallah, Casati, Toumani: Web services conversation modeling: A cornerstone for e-business automation. IEEE Internet Computing 8(1), 46–54 (2004)
5. Bistarelli, Montanari, Rossi: Semiring-based constraint satisfaction and optimization. J. ACM 44(2), 201–236 (1997)
6. Bocchi, Fiadeiro, Gilmore, Abreu, Solanki, Vankayala: A Formal Approach to Modelling Time Properties of Service-Oriented Systems. In: Handbook of Research on Non-Functional Properties for Service-oriented Systems: Future Directions. IGI (to appear in Spring 2011), `http://www.cs.le.ac.uk/srml`
7. van Dam, Adhitya, Srinivasan, Lukszo: Critical evaluation of paradigms for modelling integrated supply chains. In: Proc. ESCAPE. Computers & Chemical Engineering, vol. 33(10), pp. 1711–1726. Elsevier, Amsterdam (2009)
8. Feng, Hong-yan: Evaluation of Logistics Service Provider Using Analytic Network Proces. ICNC 2, 227–231 (2007)

9. Fiadeiro, Lopes, Bocchi, Abreu: The Sensoria reference modelling language. In: Rigorous Software Engineering for Service-Oriented Systems. LNCS. Springer, Heidelberg (2010), `http://www.cs.le.ac.uk/srml`
10. Franceschini, Rafele: Quality evaluation in logistic services. International Journal of Agile Management Systems 2(1), 49–54 (2000)
11. Ghiani, Laporte, Musmanno: Introducing Logistic Systems. In: Introduction to Logistics Systems Planning and Control, ch. 1. Wiley, Chichester (2004)
12. Hillston: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
13. Iacob, Jonkers: Quantitative Analysis of Enterprise Architectures. In: Interoperability of Enterprise Software and Applications, pp. 239–252. Springer, Heidelberg (2006)
14. Iacob, Jonkers: Quantitative Analysis of Service Oriented Architectures. International Journal of Enterprise Information Systems 3(1) (2007)
15. IBM Developer Works. Understanding quality of service for Web services, `http://www.ibm.com/developerworks/library/ws-quality.html`
16. Kalepu, Krishnaswamy, Loke: Reputation = f(user ranking, compliance, verity). In: Proc. ICWS, p. 200. IEEE, Los Alamitos (2004)
17. Legeza: Measurement of logistics-quality. Per. Pol. Trans. Eng. 31(1-2), 89–95 (2003)
18. Liu, Ngu, Zheng: QoS computation and policing in dynamic Web service selection. In: Proc. WWW Conference on Alternate Track Papers & Posters, ACM, New York (2004)
19. Ran: A model for Web services discovery with QoS. SIGecom Exch. 4(1), 1–10 (2003)
20. The Open Service Oriented Architecture collaboration. Whitepapers and specifications, `http://www.osoa.org` (see also `http://oasis-opencsa.org/sca`)
21. Tian, Gramm, Naumowicz, Ritter, Schiller: A concept for QoS integration in Web services. In: Proc. WISEW, pp. 149–155. IEEE Computer Society, Los Alamitos (2003)
22. Tribastone: The PEPA Plug-in Project. In: Quantitative Evaluation of SysTems, pp. 53–54. IEEE, Los Alamitos (2007)
23. Xu, Cao: Logistics Service Quality Analysis Based on Gray Correlation Method. International Journal of Business and Management 3(1) (2008)
24. Yang, Wu, Li: The Empirical Research on Cluster Supply Chain Flexibility and Logistics Capacity. In: ICAL, pp. 1066–1071. IEEE, Los Alamitos (2009)
25. Vu, Hauswirth, Porto, Aberer: A Search Engine for QoS-enabled Discovery of Semantic Web Services. International Journal of Business Process Integration and Management 4(4), 244–255 (2006)