SENSORIA Process Calculi for Service-Oriented Computing*

Martin Wirsing[1], Rocco De Nicola[2], Stephen Gilmore[3], Matthias Hölzl[1], Roberto Lucchi[4,**], Mirco Tribastone[3], and Gianlugi Zavattaro[4]

[1] Ludwig-Maximilians-Universität München, Germany
[2] University of Florence, Italy
[3] University of Edinburgh, United Kingdom
[4] University of Bologna, Italy

Abstract. The IST-FET Integrated Project SENSORIA aims at developing a novel comprehensive approach to the engineering of service-oriented software systems where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach. Process calculi and logical methods serve as the main mathematical basis of the SENSORIA approach.

In this paper we give first a short overview of SENSORIA and then focus on process calculi for service-oriented computing. The Service Centered Calculus SCC is a general purpose calculus which enriches traditional process calculi with an explicit notion of session; the Service Oriented Computing Kernel SOCK is inspired by the Web services protocol stack and consists of three layers for service description, service engines, and the service network; Performance Evaluation Process Algebra (PEPA) is an expressive formal language for modelling distributed systems which we use for quantitative analysis of services. The calculi and the analysis techniques are illustrated by a case study in the area of distributed e-learning systems.

1 Introduction

Service-oriented computing is an emerging paradigm where services are understood as autonomous, platform-independent computational entities that can be described, published, categorised, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems and applications. These characteristics push service-oriented computing towards nowadays widespread success, demonstrated by the fact that many large companies invest efforts and resources in promoting service delivery on a variety of computing platforms, mostly through the Internet in the form of Web services. Soon there will be a plethora of new services as required for e-government, e-business, and e-science, and other areas within the rapidly evolving Information Society.

* This work has been partially sponsored by the project SENSORIA, IST-2 005-016004 and by the DFG project MAEWA.

** Currently at European Commission, DG Joint Research Centre, Institute for Environment and Sustainability, Spatial Data Infrastructures Unit.

U. Montanari, D. Sannella, and R. Bruni (Eds.): TGC 2006, LNCS 4661, pp. 30–50, 2007.

However, service-oriented computing and development today is done in a pragmatic, mostly ad-hoc way. Theoretical foundations are missing, but they are badly needed for trusted interoperability, predictable compositionality, and for guaranteeing security, correctness, and appropriate resources usage. Service-oriented software development is not integrated in a controllable software development process based on powerful analysis and verification tools and it is not clear whether service-oriented software development scales up to the development of complete systems.

The IST-FET Integrated Project SENSORIA addresses the problems of service-oriented computing by building, from first-principles, novel theories, methods and tools supporting the *engineering of software systems for service-oriented overlay computers*. Its aim is to develop a novel comprehensive approach to the engineering of service-oriented software systems where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach.

The results of SENSORIA will include a new generalised concept of service for global overlay computers, new semantically well-defined modelling and programming primitives for services, new powerful mathematical analysis and verification techniques and tools for system behaviour and quality of service properties, and novel model-based transformation and development techniques. The innovative methods of SENSORIA will be demonstrated by applying them in the service-intensive areas of e-business, automotive systems, and telecommunications.

A main research strand of SENSORIA is the development of adequate linguistic primitives for *modelling and programming global service-oriented systems*. This includes an ontology for service-oriented architectures, UML extensions (see e.g. [35]) and the declarative language SRML [16] for system modelling over service-oriented architectures. Process calculi serve as the mathematical basis for programming and modelling dynamic aspects of services and service-oriented systems and for analysing qualitative and quantitative properties of services such as quality of service, security, performance, resource usage, scalability. During the first year of SENSORIA a *family of process calculi for services* has been developed which supports complementary aspects of service-oriented computing and qualitative and quantitative analysis of service-oriented systems. The family comprises four core calculi for service description, invocation and orchestration [6,19,25,9], stochastic and probabilistic extensions of calculi for global computing [14,15,8], and process calculi and coordination languages with cost and priority [10,5,3].

In this paper we give first a short overview of SENSORIA and then focus on process calculi for service-oriented computing. For reasons of space we present only two of the SENSORIA core calculi and one stochastic process algebra for analysing quantitative properties of service-oriented systems. For the other SENSORIA calculi the reader is referred to the SENSORIA reports and publications available on the SENSORIA web site [32].

The *Service Centered Calculus* SCC is a general purpose calculus based on an abstract notion of service (independent of any specific technology) and aiming at a model suitable for different technologies and scenarios. SCC enriches the name passing communication mechanism of traditional process calculi, such as the π-calculus [27], with explicit notions of service definition, service invocation and bi-directional sessioning.

SCC's notion of service has been influenced by Cook and Misra's calculus Orc [28] for structured orchestration of services, where a service is a function returning a stream of values.

The *Service Oriented Computing Kernel* SOCK is a three-layered calculus inspired by the Web services protocol stack. The *service description* layer consists of a simple calculus for service interface description which takes inspiration from WSDL [34] and the abstract fragment of WS-BPEL [4]. At the *service engine* layer additional information can be added which indicates how a service is actually run; this layer is inspired by the full (executable) WS-BPEL language. The third *service network* layer, permits modeling an overall network of interacting services; the source of inspiration for this level has been the SOAP [7] protocol for message exchange among Web services.

Finally, we present Jane Hillston's stochastic *Performance Evaluation Process Algebra* (PEPA) [22,21]. This is an expressive formal language for modelling distributed systems which is supported by a flexible workbench [17]. We use PEPA for quantitative analysis of services. PEPA models are obtained by composing elements which perform individual activities or cooperate on shared ones. To each activity is attached an estimate of the rate at which it may be performed. The rates associated with activities are exponentially distributed random variables thus PEPA is a *stochastic process algebra* which describes the evolution of a process in continuous time. As an example for scalability analysis, we investigate with PEPA how models of Web service execution scale with increasing client population sizes.

All three presented calculi and analysis techniques are illustrated by a case study in the area of distributed e-learning systems.

The paper is organised as follows: In Sect. 2 we present shortly the SENSORIA project, its approach to service-oriented system development, and the running example, i.e., the case study of a distributed e-learning and course management system. We present the Service Centered Calculus SCC in Sect. 3.1 and the Service-Oriented Computing Kernel SOCK in Sect. 3.2. In Sect. 4 we use PEPA for scalability analysis: we investigate how models of Web service execution scale with increasing client population sizes. We conclude the paper in Sect. 5 with some remarks on further SENSORIA results.

2 SENSORIA

SENSORIA is one of the three Integrated Projects of the Global Computing Initiative of FET-IST, the Future and Emerging Technologies action of the European Commission. The SENSORIA Consortium consists of 12 universities, two research institutes and four companies (two SMEs) from seven countries[1].

[1] LMU München (coordinator), Germany; TU Denmark at Lyngby, Denmark; FAST GmbH München, S and N AG, Paderborn (both Germany); Budapest University of Technology and Economics, Hungary; Università di Bologna, Università di Firenze, Università di Pisa, Università di Trento, ISTI Pisa, Telecom Italia Lab Turino (all Italy); Warsaw University, Poland; ATX Software SA, Lisboa, Universidade de Lisboa (both Portugal); Imperial College London, University College London, University of Edinburgh, University of Leicester (all United Kingdom).

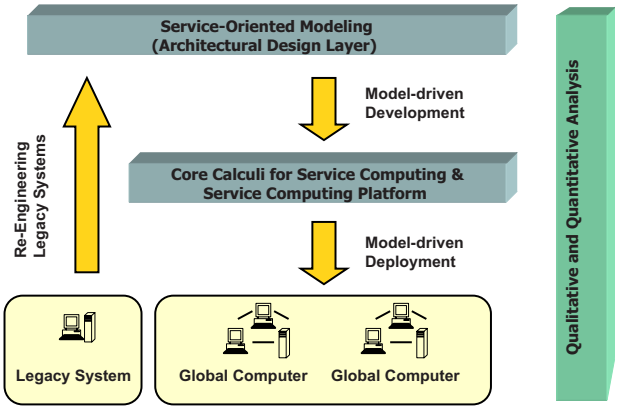


Fig. 1. The SENSORIA approach to service-oriented systems development

2.1 Aim and Approach of SENSORIA

The aim of the IST-FET Integrated Project SENSORIA is to develop a novel comprehensive approach to the engineering of service-oriented software systems where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach. This includes a new generalised concept of service, new semantically well-defined modelling and programming primitives for services, new powerful mathematical analysis and verification techniques and tools for system behaviour and quality of service properties, and novel model-based transformation and development techniques.

The three main research themes of Sensoria concern

- **The definition of adequate linguistic primitives for modelling and programming global service-oriented systems**, by building on a category theoretic and process algebraic framework supporting architectural modelling and programming for mobile global computing and by formally connecting these linguistic primitives with UML in order to make the formal approaches available for practitioners;
- **The development of qualitative and quantitative analysis methods for global services**, by using powerful mathematical analysis techniques including program analysis techniques, type systems, logics, and process calculi for investigating the behaviour and the quality of service of properties of global services;
- **The development of sound engineering and deployment techniques for global services**, by providing new mathematical well-founded techniques for model-based transformation, deployment, and re-engineering, and integrating them into a novel engineering approach to service-oriented development.

In the envisaged software development process (cf. Fig. 1), services are modelled in a platform-independent architectural design layer. By using model transformations, these models are then transformed to the SENSORIA core calculi for qualitative and quantitative analysis; moreover, for constructing operational realisations they are transformed

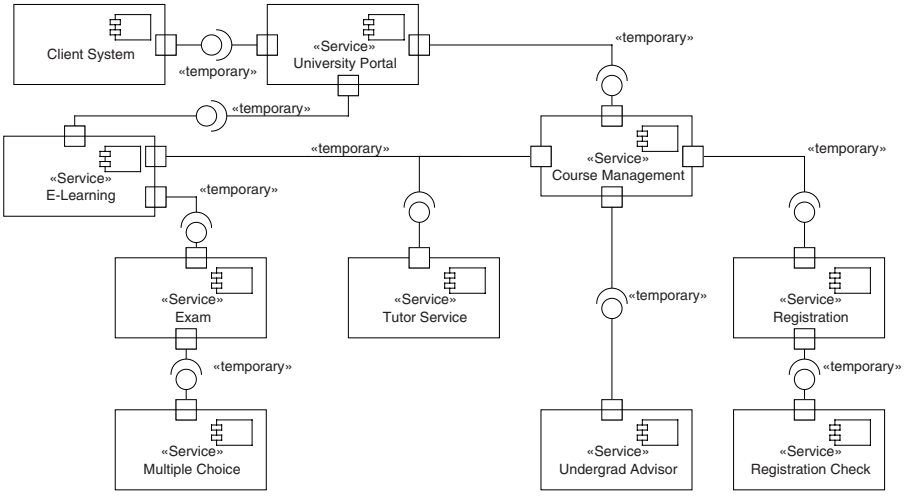


Fig. 2. Architecture of the SENSORIA distributed e-learning and course management system

and/or refined to the service computing platform of SENSORIA which, in turn, can be used for generating specific implementations over different global computing platforms in a (semi-)automated way. On the other hand, legacy code is transformed systematically into service oriented software models (see Fig. 1). In all phases of service-oriented development formal analysis techniques are used for validating and verifying qualitative and quantitative properties of services and their interactions.

2.2 The E-Learning and Course Management Case Study

Today's academic environment poses several challenges for administration, faculty, and students. Administration has to provide services for more and more students while the numbers of specialised subjects (e.g., bio-informatics and media informatics in addition to traditional computer science) increases steadily and hence more courses need to be scheduled. Faculty members are facing a higher workload and increasing demands from students, e.g., with regards to the availability of homework sheets, course slides and additional teaching aids. Furthermore students are expected to spend parts of their studies in foreign countries without delaying their exams. To manage these problems efficiently and cheaply, universities are beginning to use computerised course-management and e-learning systems. Some of the functionalities performed by typical university software systems are: management of curricula and students by a university, management of single courses by teaching personnel, management of degrees by students, and e-learning.

In SENSORIA we build a prototypical service-oriented distributed e-learning and course management system (DECMS) that satisfies these requirements and is used to guide SENSORIA research efforts. In order to support distribution, extensibility and dynamic composition, DECMS has a service-oriented architecture (cf. Fig. 2) consisting of services which are interconnected through ports that are typed by provided and

required interfaces (written in a "ball-and-socket" notation). Fig. 2 shows a UML service diagram of the DECMS as a first simplified snapshot of an architecture model.

A client service connects to DECMS through a University Portal service which in turn holds connections to the e-learning and the course management services. A tutor service interacts with both, the e-learning and course management service. The e-learning service offers an examination facility with multiple choice questions. This is provided to the e-learning service by a dedicated examination service which in turn requires an auxiliary multiple choice service. Similarly, the course management service offers services for enrolling students in courses and querying and updating the current undergrad advisor. These services are provided by the corresponding auxiliary services.

In the following we use examples concerning the management of single courses and of degrees for illustrating the different process calculi presented in this article.

3 Core Calculi for Service-Oriented Computing

The core calculi for service specification permit focusing on the foundational aspects and properties of services, and drive (via operational semantics) the implementation of prototypes: Moreover they provide a common ground for experiments and for studying relative expressiveness and lay the basis for extensions to deal with issues such as linguistic primitives (e.g. compensation), quantitative properties (e.g. Quality of Service) and qualitative properties (e.g. logics and type systems).

During the first year of SENSORIA four core calculi have been developed which put different stress on and support complementary aspects of service-oriented computing:

SCC. The *Service Centered Calculus* [6] is a general purpose calculus for services which focuses on sessions, i.e. client-service bidirectional interactions,

SOCK. The *Service-Oriented Computing Kernel* [19] proposes a three-layered calculus inspired by the Web services protocol stack (WSDL, WS-BPEL, SOAP),

COWS. The *Calculus for Orchestration of Web services* [25] mirrors WS-BPEL constructs such as message correlations, fault and compensation handlers, and flow graphs,

SC. The *Signals Calculus* [9] considers publish/subscribe service interaction instead of message-based client-service communication and supports the prototype implementation of a middleware for programming Long Running Transactions as described by the SAGA-calculus.

Two different approaches have been followed in the design of core calculi for SOC: *technology-driven* and *theory-driven*. The technology-driven approach of SOCK and COWS consists of electing one specific service-oriented technology as the starting point, and extracting from it a corresponding core model. This permits to crosscheck whether the proposed general calculi adhere to the specificities of the currently available service oriented technologies. The opposite theory-driven approach of SCC and SC consists of designing an abstract model of services that is not bound to a specific technology; the resulting calculi are general enough to be applied to different global computers on which services run.

In the following we present informally two of the four calculi, namely SCC and SOCK.

$$
\begin{array}{rll}
P, Q & ::= & \mathbf{0} \qquad\qquad\quad \text{Nil} \\
& | & a.P \qquad\qquad\quad \text{Concretion} \\
& | & (x)P \qquad\qquad\quad \text{Abstraction} \\
& | & \mathsf{return}\, a.P \qquad \text{Return Value} \\
& | & a \Rightarrow (x)P \qquad\quad \text{Service Definition} \\
& | & a\{(x)P\} \Leftarrow Q \quad \text{Service Invocation} \\
& | & r \rhd P \qquad\qquad\quad \text{Session} \\
& | & P|Q \qquad\qquad\quad \text{Parallel Composition} \\
& | & (\nu a)P \qquad\qquad\quad \text{New Name}
\end{array}
$$

Fig. 3. SCC: syntax of processes

3.1 A Session-Oriented Process Calculus for Service-Oriented Systems

SCC is a calculus developed around the notions of *service definition*, *service invocation* and *bi-directional sessioning*; it has been influenced by Cook and Misra's Orc [28], a basic programming model for structured orchestration of services. Orc is particularly appealing due to its simplicity and yet great generality; its three basic composition operators are sufficient to model the most common workflow patterns, identified by van der Aalst et al. in [33].

SCC has novel features for programming and composing services, while taking into account their dynamic behaviour. In particular, SCC supports explicit modeling of sessions both on the client and on the service side, and provides mechanisms for session naming and scoping, by relying on the constructs of π-calculus [27]. Sessions permit describing and reasoning about interaction modalities more structured than the simple *one-way* and *request-response* modalities provided by Orc and typical of a producer / consumer pattern. Moreover, in SCC, sessions can be closed thus providing a mechanism for process interruption and service cancellation and update which has no counterpart in most process calculi.

Summarising, SCC combines the service oriented flavour of Orc with the name passing communication mechanism of the π-calculus.

Calculus description. Within SCC, services are seen as interacting functions (and even stream processing functions) that can be invoked by clients. The syntax of (the kill-free fragment of) SCC is reported in Figure 3. The operational semantics is not reported for space constraints, it can be found in [6].

Service definitions take the form $a \Rightarrow (x)P$, where a is the service name, x is a formal parameter, and P is the actual implementation of the service. As an example, consider the service `double` defined as follows:

$$\texttt{double} \Rightarrow (x)x + x$$

(Here and in the following we omit the trailing $\mathbf{0}$.) This service receives the value x and computes its double $x + x$. Service invocations are written as $a\{(x)R\} \Leftarrow Q$: each new value v produced by the client Q will trigger a new invocation of service a; for each invocation, an instance of the process R, with x bound to the actual invocation value

v, implements the client-side protocol for interacting with the new instance of a. As an example, a client for the simple service described above will be written in SCC as

$$\texttt{double}\{(x)(y)\mathsf{return}\, y\} \Leftarrow 5$$

After the invocation x is bound to the argument 5, the client waits for a value from the server and the received value 10 is substituted for y and hence returned as the result of the service invocation.

This is only the simplest pattern of interaction in the context of service oriented computing, the so-called *request-response* pattern. Differently from object oriented computing, in service oriented computing clients and services can interact via more complex patterns activating *sessions* after the first invocation. Within a session several values can be exchanged from the service to the client and vice versa. Moreover, also other services can be involved giving rise to a multi-party *conversation*.

A service invocation causes activation of a new session. A pair of dual fresh names, r and $\overline{r}$, identifies the two sides of the session. Client and service protocols are instantiated each at the proper side of the session. For instance, interaction of the client and of the service described above triggers the session

$$(\nu r)\big(r \rhd 5 + 5 \,|\, \overline{r} \rhd (y)\mathsf{return}\, y\big)$$

(in this case, the client side makes no use of the formal parameter). The value 10 is computed on the service-side and then received at the client side, that reduces first to $\overline{r} \rhd \mathsf{return}\, 10$ and then to $10 \,|\, \overline{r} \rhd \mathbf{0}$ (where $\mathbf{0}$ denotes the nil process).

More generally, communication within sessions is bi-directional, in the sense that the interacting partners can exchange data in both directions. Values returned outside the session to the enclosing environment can be used to invoke other services. For instance, a client may invoke the service `double` and then print the obtained result as follows:

$$\texttt{print}\{(z)\mathbf{0}\} \Leftarrow (\ \texttt{double}\{(x)(y)\mathsf{return}\, y\} \Leftarrow 5\ )$$

(in this case, the service `print` is invoked with vacuous protocol $(z)\mathbf{0}$).

As a more significant example than those reported above, we present a simple orchestrator used in the course management system whose aim is to invoke a service which collects the results of two other services.

Example 1 (Service orchestration: email service for course events). A student wants to be notified via email of all important events for two courses in which he is enrolled. Assume that the course management system of the university provides the following services: services $Course1Events$ and $Course2Events$ provide announcements for the respective courses; the service $email$ expects a value and then sends it to a student's address. Then the following process

$$\begin{array}{rl} email\{(-)\mathbf{0}\} \Leftarrow \big(& Course1Events\{(x)(y)\mathsf{return}\, y\} \Leftarrow \bullet \\ \big| & Course2Events\{(x)(y)\mathsf{return}\, y\} \Leftarrow \bullet \\ \big) & \end{array}$$

will send an email for each announcement from either $Course1$ or $Course2$ to the student. Note that we use the names $\bullet$ and $-$ to denote unused names and binders for unused names, respectively.

As already anticipated above, another interesting aspect of SCC is that other services can be invoked during the execution of a session thus giving rise to a multi-party conversation. As an example, let us consider the following Course-Registration Check.

Example 2 (Multi-party conversation: registration service). Using a syntax enriched with the boolean *and* operator and an **if-else** construct, we can specify that a course registration might require the student to satisfy certain requirements, e.g., having completed a lab in the previous term and passing a selection test.

$$regCheck \Rightarrow (x)\,\mathbf{if}((completedLab \Leftarrow x)\ and\ (passedTest \Leftarrow x))$$
$$allow$$
$$\mathbf{else}\, deny$$

This example demonstrates the invocation of other services (`completedLab` and `passedTest`) during the execution of one service.

The full SCC comprises also other more specific operators that permit, for instance, to interrupt the execution of a session or to cancel/update service definitions. The full syntax is not reported here for space constraints, but can be found in [6]. Nevertheless we describe informally how the interruption mechanism can be used.

A protocol, both on client-side and on service-side, can be interrupted (e.g. due to the occurrence of an unexpected event), and interruption can be notified to the environment. More generally, the keyword close can be used to terminate a protocol on one side and to notify the termination to a suitable handler at the partner site. For example, the above client is extended below for exploiting a suitable service `fault` that can handle printer failures:

$$\texttt{print}\{(z)\mathbf{0}\} \Leftarrow_{\texttt{fault}} (\ \texttt{double}\{(x)(y)\mathsf{return}\, y\} \Leftarrow 5\ ) \mid \texttt{fault} \Rightarrow (code)Handler$$

where $Handler$ is a protocol able to manage printer errors according to their identifier $code$.

Suppose that P is the printer protocol and that the keyword close occurs in P. When invoked by the above client, a service-side session of the form $r \rhd_{\texttt{fault}} P[\texttt{fault}/\mathsf{close}]$ is created, where `fault` is substituted for close. In case of printer failure the protocol P should invoke the service close (instantiated to `fault`), with an error code err as a parameter. As effect of this invocation, the whole service-side session r is destroyed. The invocation will instantiate an error recovery session that executes $Handler[\texttt{err}/code]$.

Example 3 (Undergrad advisor service update). Session closing can be used also for service update. Consider, for instance, the following service from a university management system

$$undergradAdvisor \Rightarrow (-)\texttt{Prof. A}$$

that returns the name of current advisor for undergraduates. The service must be updated as soon as the occupancy of this position changes. In the kill-free fragment of SCC reported in Figure 3 there is no way to cancel a definition and replace it with a new one. By contrast, in the full calculus, we can exploit session closing in order to remove

services and the interruption handler service can be used to instantiate a new version of the same service. Consider, for instance,

$$r \rhd_{new} \left(\begin{array}{l} undergradAdvisor \Rightarrow (-)\texttt{Prof. A} \mid \\ new\{(-)\mathbf{0}\} \Leftarrow_{new} (update \Rightarrow (y)\mathsf{return}\, y) \end{array} \right) \mid$$
$$new \Rightarrow (z) \left(\begin{array}{l} undergradAdvisor \Rightarrow (-)z \mid \\ new\{(-)\mathbf{0}\} \Leftarrow_{new} (update \Rightarrow (y)\mathsf{return}\, y) \end{array} \right)$$

The service $update$, when invoked with a new name z, permits to cancel the currently active instance of service $undergradAdvisor$ and replace it with a new one that returns the name z. Notice that the service $update$ is located within the same session r of the service $undergradAdvisor$; this ensures that when it invokes the interruption handler service new the initial instance of the service $undergradAdvisor$ is removed.

Other examples that assess the expressive power of SCC can be found in [6] and include also a mapping of Orc into SCC and applications to hotel booking and blog management.

3.2 SOCK: Service Oriented Computing Kernel

SOCK is a three-layered calculus which addresses all the basic mechanisms for service interaction and composition. In particular, SOCK permits to separately deal with all the service design issues, that are decomposed into three fundamental parts: the *behaviour*, the *declaration* and the *composition*. In a few words, the behaviour represents the workflow of a service instance (session), the service declaration introduces the aspects pertaining to execution modalities and, finally, composition allows us to reason about the behaviour of the whole system composed by all the involved services.

One of the main aims of SOCK is to deal with current standards and in particular with the ones related to Web services technologies (WS-BPEL, WSDL and SOAP). Indeed SOCK extends more simple calculi [11,12,18] whose aim was to capture and model the peculiarities of orchestration languages for Web services and, in particular, of WS-BPEL. Consequently, according to WSDL specification, the basic building blocks for service interaction are the two interaction forms supported by Web services: the one-way and the request-response ones. On top of these two simple interaction modalities we can build more complex interactions among participants by means of correlation sets. Such a mechanism follows a data-driven approach to correlate many interactions, thus making it possible to program communication sessions among participants. It is worth noting that communication sessions may involve more than two peers; by communicating correlation sets new participants can enter the communication session. Activities in SOCK can be composed by means of well known WS-BPEL workflow operators like parallel, sequence and two forms of choice, the external one depending on the behaviour of the other services and the internal one where the selected activity depends only on the internal state of the service instance. Finally, as in WS-BPEL, variables are used to track the internal state of SOCK processes.

Example 4 (Course registration service). The following service allows students to register for courses. To this end it accepts `register` messages that contain identifiers for the student and the course. The service replies to the client with either a `cancel`

message (if the course is already fully booked or if the student is not eligible to take this course), or with a confirmation. If the course was confirmed the student chooses an exercise group; finally, the registration service notifies the student that he is enrolled for the course and the exercise group. The interface of the `Registration` service is specified in SOCK as follows:

$$\begin{aligned}\textsc{Registration} = \; & \mathit{register}(\langle \mathit{student}, \mathit{courseNr}\rangle);\\ & (\ \overline{\mathit{cancel}}@\mathit{student}(\langle \mathit{student}, \mathit{courseNr}\rangle) + \\ & \ (\overline{\mathit{confirm}}@\mathit{student}(\langle \mathit{student}, \mathit{courseNr}\rangle);\\ & \ \ \mathit{exerciseGroup}(\langle \mathit{student}, \mathit{courseNr}, \mathit{groupNr}\rangle);\\ & \ \ \overline{\mathit{enrolled}}@\mathit{student}(\langle \mathit{student}, \mathit{courseNr}\rangle))\ )\end{aligned}$$

The above specification is intended to be non-executable. This because further information must be added in order to indicate how the service interface is actually run by an actual service executor. The information that must be added specifies if different instances of the same service are run either in parallel or sequentially, and how the variables are managed (they can be either persistent, that is they are kept also after the execution of a service instance, or they are volatile).

An actual executor of the `Registration` service can be specified as follows

$$\texttt{RegistrationExec} = !(\{\mathit{student}, \mathit{courseNr}\} \triangleright \texttt{Registration}_{\times})$$

where ! (the equivalent of the bang operator of the π-calculus) denotes that different service instances can be run in parallel and the subscript $\times$ indicates that variables are not persistent.

Another relevant information in the specification of the `RegistrationExec` is the correlation set, given by $\{\mathit{student}, \mathit{courseNr}\}$. Correlation sets are a fundamental information in case of parallel execution of different instances of the same service. In fact, when messages are received by the executor of the service, they must be delivered to the corresponding instance. The correlation set indicates which part of the message is used to identify the correct instance.

Calculus description. The idea in SOCK is that the service design is divided into three steps; the service behaviour defines a process describing the behaviour of a service instance, while the service declaration enriches such a process with some additional information about the execution modality of the service. Such parameters, that are exploited by the service engines, describe whether to allow concurrent execution of service instances or to support persistent state of service instances. Finally, the composition is used to observe the behaviour of services when interacting each other. The SOCK calculus is equipped with an operational semantics expressed by means of a labelled transition system. For space constraints we do not report here the semantics rules that are described in [19]. In the following we report the syntax and an informal description of how SOCK works.

The layer devoted to describe the service behaviour is programmed by using the syntax reported in Fig. 4. **0** is the nil process. Outputs can be a signal $\bar{s}$, the invocation of an operation that can be one-way $\bar{o}@k(\vec{x})$ or request-response $\overline{o_r}@k(\vec{x},\vec{y})$, where s

$$
\begin{array}{lll}
P, Q ::= & \mathbf{0} & \text{(Nil)} \\
& \mid \overline{\epsilon} & \text{(Output)} \\
& \mid \epsilon & \text{(Input)} \\
& \mid x := e & \text{(Assign)} \\
& \mid \chi ? P : Q & \text{(If-then-else)} \\
& \mid P; P & \text{(Sequence)} \\
& \mid P | P & \text{(Parallel)} \\
& \mid \sum^{+}_{i \in W} \epsilon_i; P_i & \text{(Choice)} \\
& \mid \chi \rightleftharpoons P & \text{(Loop)} \\
\epsilon ::= s \mid o(\vec{x}) \mid o_r(\vec{x}, \vec{y}, P) & & \\
\overline{\epsilon} ::= \bar{s} \mid \bar{o}@k(\vec{x}) \mid \overline{o_r}@k(\vec{x}, \vec{y}) & &
\end{array}
$$

Fig. 4. SOCK: syntax of processes

is a signal name, o and o_r are operation names, k represents the receiver location and, finally, $\vec{x}$ and $\vec{y}$ are vectors of variables used to store the information passed during the request and the response phase, respectively. Dually, inputs can be an input signal s, a one-way $o(\vec{x})$ or a request-response $o_r(\vec{x}, \vec{y}, P)$ invocation where s is a signal name, o and o_r are operation names, $\vec{x}$ and $\vec{y}$ are, respectively, the vectors of variables used to store the received information and the response and, finally, P is the process that has to be executed between the request and the response. The process $x := e$ assigns the result of the expression e to the variable x. Also, $\chi ? P : Q$ is the *if-then-else* process, where χ is a logic condition on variables; if it holds then the process P is executed, otherwise, the process behaves as Q. The processes $P; Q$ and $P \mid Q$ are the standard sequential and concurrent composition of processes P and Q, respectively. $\sum^{+}_{i \in W} \epsilon_i; P_i$ represents the choice operator among input guarded processes and, finally, $\chi \rightleftharpoons P$ is the conditional loop that stops looping on P when the guard χ does not hold. In order to illustrate how SOCK works we use some examples (in the following we complete the services with their corresponding service declaration).

Example 5 (Service behaviour of multiple choice test evaluator). Let us consider the case of a service which keeps track of the score in a multiple choice test. The service suppplies a one-way operation $update$ which is invoked every time a question is answered and a request-response operation $cres$ that returns the current number of correctly and incorrectly answered questions. The operation $update$ expects a parameter indicating whether the question was answered correctly or incorrectly, while $cres$ has no parameter. We also introduce a one-way operation $reset$ that resets the test results. The service behaviour is defined by the $MultipleChoice$ process:

$$
\begin{array}{l}
MultipleChoice ::= \\
\quad (update(answer); \\
\quad\ (answer = correct)\ ?\ nrCorrect := nrCorrect + 1 \\
\qquad\qquad\qquad\qquad :\ nrFalse := nrFalse + 1\ ) \\
\quad + \\
\quad cres(\langle\,\rangle, \langle nrCorrect, nrFalse\rangle, \mathbf{0}) \\
\quad + \\
\quad reset(\langle\,\rangle); nrCorrect := 0; nrFalse := 0
\end{array}
$$

$$U ::= P_{\times} \mid P_{\bullet} \qquad W ::= c \triangleright U \qquad D ::= !W \mid W^{*} \qquad \text{(Service declaration)}$$

$$Y ::= D[H] \qquad H ::= c \triangleright P_S \qquad P_S ::= (P, \mathcal{S}) \mid P_S | P_S \qquad \text{(Service engine)}$$

Fig. 5. SOCK: syntax of service declaration and service engine

As previously mentioned the service behaviour programs session instances, in this case $MultipleChoice$ supports three possible behaviours depending on the operation that is invoked: i) $update$: one of the variables used to count the number of correct or false answers is updated, the parameter $answer$ determines which one, ii) $cres$: the variables $nrCorrect$ and $nrFalse$, that contain the numbers of correct or false answers, are returned to the invoker, and iii) $reset$: the variables $nrCorrect$ and $nrFalse$ are set to 0.

Example 6 (Service behaviour of orchestration: tutor service). We consider the case of a matchmaking service for private tuition: This service can be used in a course management system to match tutors willing to offer private tuition with students requesting extra tuition. Each offer is identified by an offer id (oId). The process $TutorService$, whose definition follows, defines the skeleton of the service behaviour that orchestrates tutors and students:

$$\begin{aligned} &TutorService ::= \\ &\quad requestTuition(oId, accept, offerTuition(oId, accept, \mathbf{0})) \\ &\quad + \\ &\quad offerTuition(oId, accept, requestTuition(oId, accept, \mathbf{0})) \end{aligned}$$

In this process two request-response operations are supported, namely $requestTuition$ and $offerTuition$. If the $requestTuition$ (resp. the $offerTuition$) operation is selected, the process responds to the invoker when the $offerTuition$ (resp. the $requestTuition$) operation is performed and completed. As we will see in the following we exploit oId as a correlation set, in the service declaration, to drive the sessions and join the student and the tutor.

The service declaration consists of the service behaviour and of some parameters describing how to execute the service instances. The syntax is reported in Fig. 5. The term D represents a service declaration while Y represents a service engine. Service declarations are composed by a service behaviour, a flag describing how to manage the internal states of service instances, the set of variables which play the role of correlation set and a flag used to allow concurrent or sequential execution of service instances. In particular, flag $\times$ denotes that P is equipped with a non-persistent state while $\bullet$ denotes the opposite. Also, c is the correlation set which guards the execution of the sessions and, finally, $!W$ denotes a concurrent execution of the sessions while W^* denotes that sessions are executed sequentially. Service engines are used to describe the behaviour of the service during the execution. In particular, they are characterized by a service declaration and by the process H which represents the execution state of the service instances that, if the persistent state is not supported, are equipped with their own state S while, in the opposite case, they refer to a unique state shared among the instances.

Example 7 (Service declaration of multiple choice service). Now we recall the $MultipleChoice$ service behaviour of Example 5 and we conclude its design by describing the service declaration which follows:

$$MultipleChoiceDec ::= \{\,\} \triangleright MultipleChoice^{*}_{\bullet}$$

In this case the service supports the sequential execution of service instances, the persistent state and does not exploit correlation sets. The persistent state makes it possible to use variables to keep track of the test results; this is because service instances inherit the variables state of the previous service instance execution. It is worth noting that the sequential execution guarantees that variables $nrCorrect$ and $nrFalse$ are managed in a consistent way. Indeed, in the case of concurrent $update$ invocations the variables updates are sequentially performed. When we intend to support the concurrent execution of service instances, the service behaviour must be refined by controlling the access to the critical section which updates the variables. This could be done by exploiting, for instance, the internal synchronization primitives.

Example 8 (Service declaration of tutor service orchestration). Now we recall the $TutorService$ service behaviour of Example 6 and we conclude its design by describing the service declaration which follows:

$$TutorServiceDec ::= \{oId\} \triangleright\, !TutorService_{\times}$$

In this case the service supports the concurrent execution of service instances and non-persistent state. The correlation set contains oId which is instantiated by the first operation invocation and is exploited in the second one to select the right invocation call (i.e. the one associated to the same offer id). As it emerges by this example, the correlation set mechanism allows to involve a number of peers (in this case the service itself, the student and the tutor clients) within a service instance.

Concluding, the third layer of the calculus allows us to reason about the behaviour of the whole system that, essentially, consists of the parallel composition of service engines. For example, this layer could be used to investigate the behaviour of the system composed by the tutor and student clients and by a *TradingService* orchestration service. Interested readers can find all the details in [19].

4 Stochastic Analysis of Nonfunctional Properties of Service-Oriented Systems

Well-engineered, safe systems need to deliver reliable services in a timely fashion with good availability. For this reason, we view quantitative analysis techniques as being as important as qualitative ones. The quantitative analysis of computer systems through construction and solution of descriptive models is a hugely profitable activity: brief analysis of a model can provide as much insight as hours of simulation and measurement. Jane Hillston's Performance Evaluation Process Algebra (PEPA) [22] is an expressive formal language for modelling distributed systems. PEPA models are constructed by the composition of components which perform individual activities or cooperate on shared ones. To each activity is attached an estimate of the rate at which it may

be performed. The rates associated with activities are exponentially distributed random variables thus PEPA is a *stochastic process algebra* which describes the evolution of a process in continuous time.

Using such a model, a system designer can determine whether a candidate design meets both the behavioural and the temporal requirements demanded of it. That is: the service may be secure, but can it be executed quickly enough to perform its function within a specified time bound, with a given probability of success?

4.1 An Application: Scalability Analysis

A growing concern of Web service providers is *scalability*. An implementation of a Web service may be able at present to support its user base, but how can a provider judge what will happen if that user base grows? We present a modelling approach supported by the PEPA process algebra which allows service providers to investigate how models of Web service execution scale with increasing client population sizes. The method has the benefit of allowing a simple model of the service to be scaled to realistic population sizes without the modeller needing to aggregate or re-model the system.

One of the most severe problems a Distributed E-learning and Course Management System (DECMS) has to deal with is the performance degradation occurring when many users are requesting the service simultaneously. Let us imagine a DECMS is available for collecting final course projects of a class. Teaching staff usually put a deadline on those activities, and students are likely to get their projects ready very close to the due date. The DECMS has to cope with a flash crowd-like effect, as server resources (i.e. memory, CPU and bandwidth) have to be shared among a large number of users, thus paving the way for performance penalties experienced by users.

4.2 Setup of the Model

We consider the model in the optimistic scenario where hardware and software failures are assumed to occur sufficiently infrequently that we will not represent them. Further, the server is sufficiently well-provisioned that we may also neglect the possibility failures caused by out-of-memory errors or overrunning the thread limit on the JVM hosting the Web container. We will return to review these optimistic assumptions after we compute performance results from our model.

We conducted experiments to estimate the appropriate numerical values for the parameters used in our model. We implemented a simple Web Service in which SwA was enabled to allow it to save a binary file attached by the client. The implementation of the server interface as well as the method for processing attachments are timed methods, in order to let us gather measurement data on their invocation.

The client makes a designer-tunable number of service calls, the attachment file size being passed as application argument. The designer may also set an inter-message idle period; however, our results were not affected by changes in this parameter.

We restrict our analysis to a case where one single course is being managed. We assume that no other services simultaneously run on the server; thus, the server download capacity c_s as well as server upload capacity μ_s are fully available for the Web Service. The clients' (i.e. students) arrival process is assumed to be well-described by a Poisson distribution with rate λ. The system allows a maximum number of students (course

size) N. We assume that all students have the same values for download capacity c_c and upload capacity μ_c. Like the server, we also suppose that no other process but the Web Service client-side application consumes network resources.

When multiple clients are involved, the server has to share its bandwidth among them. A model of the behaviour of the network is therefore necessary. We address this issue by developing a simple model for characterising service performance of the system. In this model we assume an ideal network in which no loss occurs and network nominal *capacity* means *available bandwidth*. We also suppose that transmissions are established on top of TCP connections where fairness against concurrent requests is perfect.

Given the above assumptions, if we denote i $(i > 0)$ as the number of uploading clients at any point in time, the uploading rate of each connection $request$ is:

$$request = \min\left\{\frac{c_s}{i}, \mu_c\right\} \tag{1}$$

Similarly, if j is the number of downloading clients (i.e., clients who are receiving the response message), the downloading rate of each connection $response$ is:

$$response = \min\left\{\frac{\mu_s}{j}, c_c\right\} \tag{2}$$

4.3 Model Analysis

Model analysis has been carried out by setting local activity rates as they were obtained in our experimental tests. Table 1 shows the complete parameter set. It is worthwhile to observe that network parameters represent bandwidths normalised by the message size being sent. For instance, $c_s = 0.001$ means that the server is able to get the entire message completed in 1000 s; this value resembles a realistic situation where a server equipped with a 10 Mbps connection has to download a file about 1 GB long. We also would like to point out that server upload capacity is much faster than its download

Table 1. Parameter set for model analysis

Parameter	Meaning	Rate (s^{-1})
α	$create$	1689.20
β	$attach$	25000.00
γ	$processResponse$	6493.50
θ	$save$	12.33
η	$processRequest$	1290.32
λ	$queue$	20.00
N	Population size	100
c_s	Server download bandwidth	0.001
μ_s	Server upload bandwidth	$c_s/3$
c_c	Client download bandwidth	$(c_s/10) \cdot 10^6$
μ_c	Client upload bandwidth	$c_c/30$

capacity because of the size of the message being transmitted: here we have assumed 1 KB long SOAP response messages in our parameter set. The value of λ is to consider flash crowd-like effect, such that triggered for instance by simultaneous service requests when a deadline is due.

As our model considers client components which perform only one request, transient analysis has to be carried out for evaluating the performance of the system. The traditional approach to attempt this numerical evaluation via transient analysis of a continuous-time Markov chain will not succeed here because the state space of the system is too large. However, as shown in [21], the ODE-based representation of the model offers excellent scalability because the size of the space vector does not change for N varying. The model is shown in Fig. 6.

$$
\begin{aligned}
ClientIdle &\stackrel{def}{=} (queue, \lambda).ClientUploading \\
ClientUploading &\stackrel{def}{=} (request, \top).Stop \\
Server_0 &\stackrel{def}{=} (queue, \top).Server_1 \\
Server_i &\stackrel{def}{=} (queue, \top).Server_{i+1} + (request, \min\{\tfrac{c_s}{i}, \mu_c\}).Server_{i-1} \\
&\quad (0 < i < N) \\
Server_N &\stackrel{def}{=} (request, \min\{\tfrac{c_s}{N}, \mu_c\}).Server_{N-1}
\end{aligned}
$$

$$
\left(\underbrace{ClientIdle \parallel ClientIdle \parallel \cdots \parallel ClientIdle}_{N} \right) \underset{\{queue, request, response\}}{\bowtie} Server_0
$$

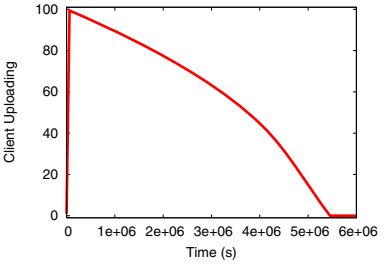
Fig. 6. Simplified PEPA model of the DECMS

4.4 Numerical Results

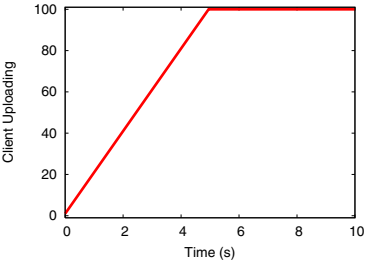
We used the PEPA Workbench [17] to compile the PEPA model to a differential equation form which we could solve using a fifth-order Runge Kutta numerical integrator. In the continuous-space representation performance results could be evaluated at low computational cost. In particular, we required only 0.03 seconds of compute time to obtain a 10^6 seconds time series analysis. We considered a maximum number of users $N = 100$, requesting service according to a flash crowd-like effect at rate $\lambda = 20$. Server download capacity c_s was set to 0.001, and client upload capacity $\mu_c = c_s/30$.

Figure 7 (a) shows a time series plot of the number of client uploading to the server and Figure 7 (b) the initial burstiness of requests. Figure 7 (c) plots service durations for different server bandwidths (i.e., $c_s = 0.01, 0.02$, and 0.1) and Figure 7 (d) plots service durations for different values of N, when $c_s = 0.1$ and $\mu_c = c_s/30$.

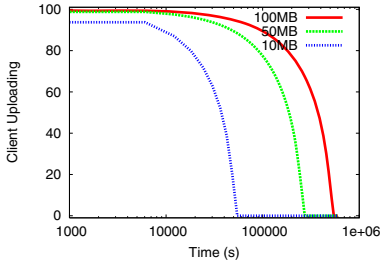
Commentary on the results: We note that the system requires a significant amount of time to get every client request completed. Earlier we outlined a series of assumptions about the model setup which included the optimistic assumptions of absence of failure of various kinds, and did not include the possibility of users aborting long-running file uploads only to restart them again later. Since unsuccessful file transfers (of whatever kind) will only tend to delay things more we can safely interpret the results presented



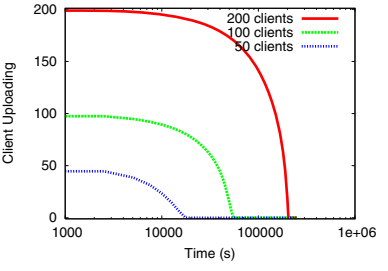
(a) Evolution of the number of clients uploading



(b) Flash crowd effect in DECMS



(c) Time series for different server bandwidths



(d) Time series for different number of users

Fig. 7. Scalability analysis of the e-learning case study

above as saying that even in this very optimistic setting the system is impractical for use and that an alternative design must be tried to support the expected number of student users.

5 Concluding Remarks

In this paper we have presented some of the first results of the SENSORIA project on software Engineering for Service-oriented Overlay Computers. We have focused on process calculi for service-oriented computing and informally explained the session-oriented general purpose calculus SCC for service description, the three layered calculus SOCK inspired by the Web Services protocol stack (WSDL, WS-BPEL, SOAP), and a technique for scalability analysis using the stochastic process calculus PEPA.

But these results represent only a small part of the SENSORIA project. In addition, the SENSORIA project is developing a comprehensive service ontology and a (SENSORIA) Reference Modelling Language (SRML) [16] for supporting service-oriented modelling at high levels of abstraction of "business" or "domain" architectures (similar to the aims of the service component architecture SCA [31]). Other research strands of SENSORIA comprise a probabilistic extension of a Linda-like language for service-oriented computing [8], stochastic extensions of KLAIM [30], and beta-binders [15].

SENSORIA addresses qualitative analysis techniques for security and control of resource usage. A first step towards a framework for modelling and analysing security and trust for services includes trust management and static analysis techniques for crypto-protocols [26,36], security issues on shared space coordination languages [20], secure service composition [1], techniques for ensuring constraints on interfaces between services [29], and autonomic security mechanisms [23]. The results for control resource usage by services range from a flow logic for resource access control [20] and model checking properties of workflow processes [24] to type systems for confining movements of data and processes [13] and for composing incomplete software components [2].

Moreover, SENSORIA is developing a model-driven approach for service-oriented software engineering (see also [35]) and a suite of tools and techniques for deploying service-oriented systems and for re-engineering of legacy software into services. By integrating and further developing these results SENSORIA will achieve its overall aim: a comprehensive and pragmatic but theoretically well founded approach to software engineering for service-oriented systems.

References

1. Bartoletti, M., Degano, P., Ferrari, G.: Security Issues in Service Composition. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 1–16. Springer, Heidelberg (2006)
2. Bettini, L., Bono, V., Likavec, S.: Safe and flexible objects with subtyping. SAC 2005 4(10), 5–29 (2005)
3. Bistarelli, S., Gadducci, F.: Enhancing constraints manipulation in semiring-based formalisms. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) Proceedings of ECAI 2006, 17th European Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 63–67. IOS Press, Amsterdam (2006)
4. Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C.K., Thatte, S., Yendluri, P., Yiu, A.: Web services business process execution language version 2.0. Technical report, WS-BPEL TC OASIS (2005), http://www.oasis-open.org/
5. Bonchi, F., Koenig, B., Montanari, U.: Saturated semantics for reactive systems. In: Proceedings of LICS 2006, 21st Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, Los Alamitos (to appear, 2006)
6. Boreale, M., Bruni, R., Caires, L., De Nicola, R., Lanese, I., Loreti, M., Martins, F., Montanari, U., Ravara, A., Sangiorgi, D., Vasconcelos, V., Zavattaro, G.: SCC: a service centered calculus. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 38–57. Springer, Heidelberg (2006)
7. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.2. W3C Recommendation (June 24, 2003), http://www.w3.org/TR/SOAP/
8. Bravetti, M., Zavattaro, G.: Service Oriented Computing from a Process Algebraic Perspective. Journal of Logic and Algebraic Programming 70(1), 3–14 (2006)
9. Bruni, R., Ferrari, G., Melgratti, H., Montanari, U., Strollo, D., Tuosto, E.: From theory to practice in transactional composition of web services. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.) Formal Techniques for Computer Systems and Business Processes. LNCS, vol. 3670, pp. 272–286. Springer, Heidelberg (2005)
10. Buscemi, M.G., Montanari, U.: Cc-pi: A constraint-based language for specifying service level agreements. In: Proc. ESOP'07, volume to appear of LNCS (2007)

11. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration: a synergic approach for system design. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 228–240. Springer, Heidelberg (2005)
12. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration conformance for system design. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 63–81. Springer, Heidelberg (2006)
13. De Nicola, R., Gorla, D., Pugliese, R.: Confining data and processes in global computing applications. Science of Computer Programming 63(1), 57–87 (2006)
14. De Nicola, R., Katoen, J.-P., Latella, D., Massink, M.: STOKLAIM: A Stochastic Extension of KLAIM. TR 2006-TR-01, ISTI (2006)
15. Degano, P., Prandi, D., Priami, C., Quaglia, P.: Beta-binders for biological quantitative experiments. In: ENTCS - Proceedings of QAPL, 4th Workshop on Quantitative Aspects of Programming Languages, 2006 (to appear, 2006)
16. Fiadeiro, J.L., Lopes, A., Bocchi, L.: A formal approach to service component architecture. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 193–213. Springer, Heidelberg (2006)
17. Gilmore, S., Hillston, J.: The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In: Haring, G., Kotsis, G. (eds.) Computer Performance Evaluation. LNCS, vol. 794, pp. 353–368. Springer, Heidelberg (1994)
18. Guidi, C., Lucchi, R.: Mobility mechanisms in service oriented computing. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 233–250. Springer, Heidelberg (2006)
19. Guidi, C., Lucchi, R., Busi, N., Gorrieri, R., Zavattaro, G.: SOCK: a calculus for service oriented computing. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 327–338. Springer, Heidelberg (2006)
20. Hansen, R.R., Probst, C.W., Nielson, F.: Sandboxing in myKlaim. In: The First Internat. Conference on Availability, Reliability and Security, ARES 2006 (2006)
21. Hillston, J.: Fluid flow approximation of PEPA models. In: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems, Torino, Italy, September 2005, pp. 33–43. IEEE Computer Society Press, Los Alamitos (2005)
22. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
23. Koshutanski, H., Martinelli, F., Mori, P., Vaccarelli, A.: Fine-grained and history-based access control with trust management for autonomic grid services. In: Proceedings of the 2nd International Conference on Automatic and Autonomous Systems (ICAS'06), Silicon Valley, California, July 2006, IEEE Press, Orlando (2006)
24. Kovács, M., Gönczy, L.: Simulation and formal analysis of workflow models. In: Bruni, R., Varro, D. (eds.) Proc. of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques. ENTCS, Elsevier, Amsterdam (2006)
25. Lapadula, A., Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. In: Proc. of ESOP'07, volume to appear of LNCS (2007)
26. Martinelli, F., Petrocchi, M.: A uniform framework for the modeling and analysis of security and trust. In: Proc. of 1st Workshop on Information and Computer Security- ICS 2006. ENTCS, Elsevier, North-Holland (to appear, 2006)
27. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. Inform. and Comput. 100(1), 1–40 (1992)
28. Misra, J., Cook, W.R.: Computation orchestration: A basis for wide-area computing. Journal of Software and Systems Modeling (to appear, 2006)
29. Nielson, H.R., Nielson, F.: Data flow analysis for CCS. Festschrift dedicated to Reinhard Wilhelm's 60. birthday (2006)

30. De Nicola, R., Katoen, J.P., Latella, D., Massink, M.: STOKLAIM: A Stochastic Extension of KLAIM. TR 2006-TR-01, ISTI (2006)
31. SCA Consortium. Service Component Architecture, version 0.9. Specification, 2005 (Last visited: June 2006), download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA_White_Paper1_09.pdf
32. SENSORIA. Software Engineering for Service-Oriented Overlay Computers. Web site at http://www.sensoria-ist.eu
33. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
34. W3C. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl
35. Wirsing, M., Clark, A., Gilmore, S., Hölzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-Based Development of Service-Oriented Systems. In: Najm, E., Pradat-Peyre, J.F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 24–45. Springer, Heidelberg (2006)
36. Zunino, R., Degano, P.: Handling exp, × (and timestamps) in protocol analysis. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006 and ETAPS 2006. LNCS, vol. 3921, pp. 413–427. Springer, Heidelberg (2006)