Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems

Massimiliano Colombo[1], Elisabetta Di Nitto[1], Massimiliano Di Penta[2], Damiano Distante[2], and Maurilio Zuccalà[1]

[1] CEFRIEL – Politecnico di Milano
Via Fucini 2, 20133 Milano, Italy
dinitto@elet.polimi.it, {mcolombo, zuccala}@cefriel.it
http://www.cefriel.it
[2] RCOST – Research Centre on Software Technology
University of Sannio, Department of Engineering
Palazzo ex Poste, Viale Traiano, 82100 Benevento, Italy
{dipenta, distante}@unisannio.it
http://www.rcost.unisannio.it

Abstract. The diffusion of service-oriented computing is today heavily influencing many software development and research activities. Despite this, service-oriented computing is a relatively new field, where many aspects still suffer from a lack of standardization. Also, the service-oriented approach is bringing together researchers from different communities or from organizations having developed their own solutions. This introduces the need for letting all these people communicate with each other using a common language and a common understanding of the technologies they are using or building.
This paper proposes a conceptual model that describes actors, activities and entities involved in a service-oriented scenario and the relationships between them. While being created for a European project, the model is easily adaptable to address the needs of any other service-oriented initiative.

1 Introduction

Service-oriented computing represents a conceptual approach and a set of technologies that are greatly contributing to radically change the perspective of today's software development. Services are an effective solution to let software systems, developed by different organizations and spread across the world, interoperate. One typical example is, for sure, the one of bioinformatics [1], where services allow an easier integration of solutions developed by different research groups, each one having different skills and using various development technologies. Also, services permit to parallelize computational-intensive tasks: Grid Computing is probably the most relevant example in which parallel computing can benefit from services.

Lately, interesting challenges such as automatic service discovery, composition, or verification, have pushed several researchers, coming from different fields

and communities, to put together their efforts. However, service-oriented computing is still a relatively new field. There are too many different issues that are not yet mature, lacking standardization or even full comprehension by researchers. A significant example of these problems is ensuring trustworthiness [2] between interacting parties. There are attempts to identify approaches solving the issue under specific constraints, which usually imply the preliminary establishment of a service level agreement (SLA). However, the issue of offering mechanisms to enable trust in a dynamically changing set of services is still open.

More in general, there is no common terminology nor common understanding on the basic concepts of the service domain. For example, in some cases services are assimilated to components, while in some others they appear to be a distinct even if related concept. This introduces the need for a rationalization of activities, entities, and stakeholders involved in the service-oriented scenario, clearly indicating their meaning and their relationships.

Working within the SeCSE European project [3] – which aims at developing processes, methods and tools to develop service-oriented systems – we faced the urgent need to provide a clear definition of the concept of service and of the related concepts concerning service publication, discovery, composition, execution, and monitoring, as a common reference for partners involved in the project. As we discuss in Sect. 5, although other conceptualization attempts have been proposed in the literature, they focus on aspects that are different or complementary to our goals.

This paper presents our conceptual model for service-oriented systems. Even if it has been originally created to deal with the needs of the SeCSE project, its main principles should fit any service-oriented scenario. The model describes actors, entities, and activities relevant to the service domain, and the relationships existing between them. The model is specified using UML class diagrams complemented with a data dictionary. To properly ensure the model understanding, we have instantiated it on a simple scenario. The remainder of the paper is organized as follows. Sect. 2 presents the requirements for our conceptual model. Sect. 3 presents the model itself describing the diagrams it is composed of. Sect. 4 introduces the scenario we use to exemplify the conceptual model. Sect. 5 summarizes other attempts to conceptualize the world of service-oriented systems and relates them to our approach. Sect. 6 concludes the paper.

2 Requirements for the Conceptual Model

The definition of our conceptual model has been driven by the need to provide a common conceptual framework within the SeCSE project. The first two (meta) requirements we faced were compactness – thus avoiding the redundancies and inconsistencies we found in other models (see Sect. 5) – and extensibility – thus enabling the possibility to add new concepts, relationships, and activities to the model itself. Moreover, for the sake of generality, we decided to keep the model independent of any technological choice or standard, even if the SeCSE project is currently focusing on Web services as its main technology.

Such overall needs led to the following more specific requirements:

- *To clarify the meaning of 'service'.* We have noticed that this term is being used in quite different ways in various domains. For example, in the technical domain it is usually considered as a particular software system that can be published, located, and invoked across the Internet. In the business domain it has a much broader and abstract meaning, and it is defined as the non-material equivalent of a good, while service provision is defined as an economic activity that does not result in ownership; this is what differentiates it from providing physical goods.
- *To clarify the difference between a service and its public description.* A service is really available if information on how to access it is made public. In some cases services are confused with their public descriptions. While this is understandable from the service consumer viewpoint, we think that for service developers and integrators it is beneficial to highlight differences and relationships between those concepts.
- *To clarify the distinction between 'simple' vs. 'stateless', and 'composite' vs. 'stateful' services.* While in [4] there is no clear distinction between the elements of these two pairs, we think that they are distinct if not orthogonal. In particular, we argue that the term 'stateful' refers to the possibility for a service to maintain a state between two consecutive operation requests, while the term 'stateless' has the opposite meaning. Moreover, we feel that the terms 'simple' and 'composite' are better used to mean, respectively, services that do not rely on the execution of others, and services that do so.
- *To identify the various stakeholders that exploit, offer, and manage services.* Various actors are involved in a service-oriented system. Besides the usual roles of service consumers and providers, we have noticed that other important roles are the different mediators who compose or certify services, support service discovery, etc. Indeed, we have also noticed that such roles are increasingly played by automated agents, not only by human beings.
- *To capture the relevant aspects concerning service discovery, composition, publication, execution, and monitoring.* These, in fact, are the main research areas of the SeCSE project. More in detail, the project is structured in the following research activities:
  - *Service engineering:* extending the existing approaches to service and system specification in order to include requirements capable of modeling, from a service perspective, quality of service (QoS) specifications, and to provide support for using these specifications within service discovery and binding mechanisms. The project is also developing approaches and tools for service testing.
  - *Service discovery:* providing means to discover services in different phases of the service life-cycle, from requirement analysis to run-time execution.
  - *Service delivery:* focusing, at deployment and operation level, on tools and techniques for the validation, testing and run-time monitoring of services and service-centric systems.

- *System engineering:* focusing on the analysis and development of architectural models for service-centric systems that accommodate services, components, and their dynamic composition.

3 Conceptual Model Overview

The SeCSE conceptual model aims at providing a common terminology across the project. It has been designed as a compact and extensible model that takes into account all the service-related concepts that have been identified inside the project. The model is specified using UML and is described by means of different diagrams, each offering a view on a specific aspect of the service-oriented system engineering process. Namely these diagrams are: *Agent-Actors*, *Core*, *Service Description*, *Service Discovery*, *Service Composition*, *Service Monitoring* and *Service Publication*. In the remainder of this section we briefly describe each diagram (conceptual model items are capitalized and formatted in *italic* the first time they appear). We only show the most interesting ones due to the limited space available. The interested reader can find all the diagrams in the extended technical report [5]. It is worth pointing out that any ontology-based language could be used instead of UML for describing the model.

3.1 Agents and Actors

A very important aspect concerning the development and operation of any application is to identify the stakeholders and the roles they play. This is particularly important for service-oriented systems since in this case, as we have highlighted in Sect. 2, the number of different stakeholders and roles can be quite high. We use the term *Agents* to mean entities of the real-world, and *Actors* to indicate roles that Agents may play. Agents include *Person* or *Organization*, *Software System*, *Service* and *Legacy System*. Actors include *Service Provider*, *Service Developer*, *Service Integrator*, *Service Broker*, *Service Consumer*, *Service Monitor*, *Service Certifier* and *System Engineer*. In principle, an Agent can take any of the roles identified by Actors (e.g., a Person can act as an Operation Provider), and vice versa a role can be taken by any Agent (e.g., a Service Consumer can either be a Person or a Service). Some of the identified Actors are represented in the Core model in Fig. 1, while a complete diagram showing the hierarchy of Agents and Actors can be found in [5].

3.2 Core Model

Fig. 1 depicts the main concepts that are part of the conceptual model, and highlights the main Actors that interact with them. A Service is a particular concrete *Resource* which is offered by a Software System.

A Service has a *Service Description*. In order to discover a Service, a Service Consumer can express a *Service Request* that may match with zero or more

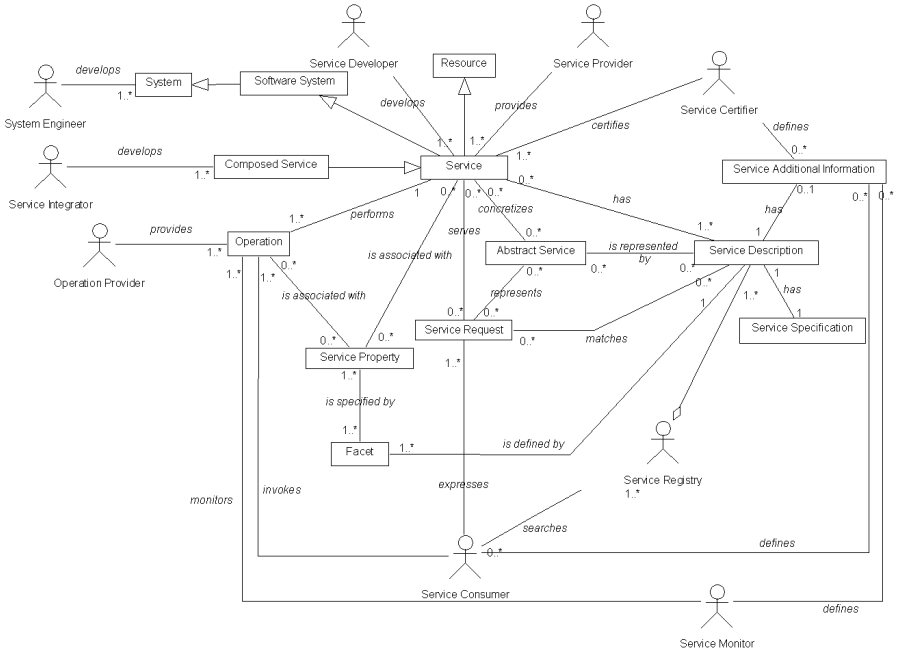

Fig. 1. Core model

Service Descriptions. After being discovered, a Service will serve the Service Request, i.e., it will be used by the Service Consumer that invokes its *Operations*. Either a Service Request or a Service Description by itself can represent an *Abstract Service*, i.e., the 'idea' of a service, a service that does not have a concrete implementation (yet). An Abstract Service can be published, discovered, and then concretized when needed in a (concrete) Service. A *Composed Service* is a particular kind of Service, developed by a Service Integrator, which makes use of other Services.

3.3 Service Description

An important aspect of services is their Service Description. In fact, it is through such a description that they are known by the potential consumers. In our model both aspects of a Service Description, i.e., Service Specification and Service Additional Information, are expressed by means of *Facets*. Each Facet is the expression of one or more *Service Properties* in some specification language. A Service Specification is usually provided by the Service Provider and may include both functional and non-functional information such as the service interface, service behavior, information on service exceptions, test suites, or service QoS attributes. The Service Additional Information is usually provided by actors

different from the Service Provider (e.g., by Service Consumers, or by Service Certifiers) and may include information such as user Ratings, Measured QoS, Usage History, some measure of trustworthiness, etc.

3.4 Service Discovery

Service discovery can be performed in various phases of a service–oriented system life cycle. It can be done when the requirements for a new system are gathered (within the SeCSE project this is called 'early discovery'), when the system is being designed and new specific needs for services are identified (in SeCSE this is called 'Architecture-time discovery'), or when the system is running. Run-time discovery has the goal of finding Services that can replace the ones that the system is currently using or, also, part of the internal logic of the system itself. Consistently with this classification, the conceptual model includes three types of queries that are executed in the three cases described above.

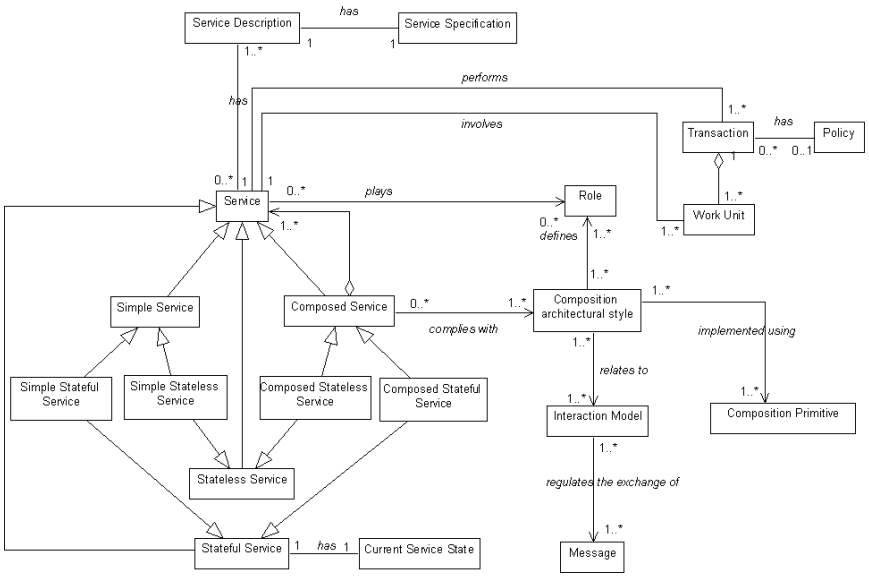

Fig. 2. Service composition

3.5 Service Composition

Fig. 2 presents the classification of a Service with respect to its state (i.e., stateless vs. stateful) and its compositeness (i.e., simple vs. composed), already discussed in Sect. 2. The diagram also shows the relationships existing between a

Composed Service and the adopted *Composition Architectural Styles*, the Roles a Service can accordingly play, etc. In addition, the diagram links the concept of *Transaction* and *Work Unit* with a Service: a Service performs Transactions, which are composed of Work Units (these are atomic steps for which some property applies, e.g., ACID). A *Policy* associated with a Transaction is a collection of assertions that declare the semantics of the Transaction itself (e.g., ACID or long-running, participants, coordination protocol, transaction faults and corresponding actions to be performed, etc.).

3.6 Service Monitoring

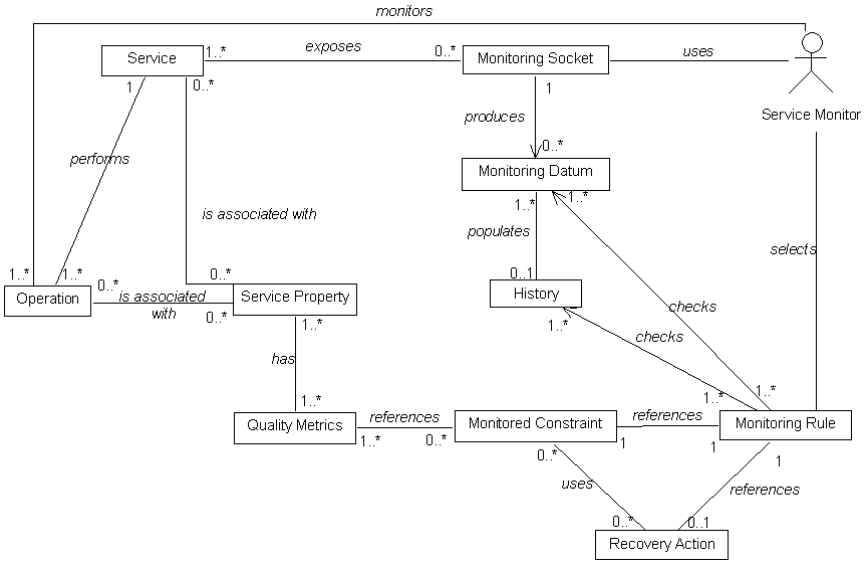

Fig. 3. Service monitoring

Monitoring is very important in a service-oriented scenario where the services being used within a system are not under the control of the system itself. Fig. 3 depicts the main concepts and the relationships involved in monitoring. A Service can be monitored if the software system that features it offers some appropriate monitoring mechanisms, i.e. the *Monitoring Sockets*. A Monitoring Socket is able to produce some *Monitoring Data* that are then checked by some *Monitoring Rule*, to verify some *Monitored Constraints* expressed over one or more *Quality Metrics*. These, in turn, express some measure of some Service Property (see Sect. 3.3). Service Properties can refer either to an entire Service (e.g., Mean Time Between Failure (MTBF) of 1 hour per week) or to one or

more Operations offered by the Service itself (e.g., operation 'X' has to feature some transactional property). Monitoring Data can be collected in a *History*. In some cases the Monitored Constraints check an entire History rather than a single datum. Service monitoring is performed by a Service Monitor. The kind of properties involved in a Monitored Constraint usually depends on the actual agent that performs the monitoring and on its visibility on the service execution.

3.7 Service Publication

The service publication model addresses the fact that a Service Provider can publish one or more Service Descriptions on a *Service Registry*. Service Registries can be organized in *Federations* resulting from an agreement made by organizations running Service Registries to achieve a joint aim (e.g., being focused on a similar topic, having some trust relationship, etc.). Federations can be used to propagate information (e.g., Service Requests or Service Descriptions) among different Service Registries.

4 An Example Scenario: The Pizza Delivery System

In this section we exemplify the concepts composing the conceptual model presented in Sect. 3 by describing an example scenario. We firstly describe the scenario from a service consumer viewpoint. Then, we provide a description of the scenario from a behind-the-scenes perspective. Finally, we explicitly map the conceptual model component elements over the scenario. Services are formatted in `typewriter` style, while in Sect. 4.3 conceptual model items are again capitalized and formatted in *italic* the first time they appear.

4.1 Pizza Delivery System: The Service Consumer Viewpoint

James and his wife Sarah want to have pizza for dinner at home. Through his PDA, James connects to the service directory of his Internet Service Provider (ISP) and searches for 'pizza'. He gets assorted results (pizza restaurants, pizza parlors offering delivery or takeaway, supermarkets selling frozen pizzas, recipes to prepare and bake pizza at home, etc.).

James refines his request searching for 'pizza parlor and delivery'. Then he selects one of the available parlors, taking into account criteria such as oven type, price range, maximum delivery time and rating expressed by previous clients of that service. The selected service is `PizzaOverall`.

James accesses the `PizzaOverall` service interface and orders two pizzas providing his and Sarah's preferences as for topping and baking options (with pepperoni and crusty for James, with mushrooms and soft for Sarah) together with other required information such as delivery address and time. He invokes the proper service operations providing necessary input data.

James is also requested to select a payment method to complete the order. He chooses to pay by credit card, so he has to provide further details such as card

company, card number, expiration date, etc. After this, the service invocation is finished and James receives a receipt via e-mail with a detailed summary.

At 8 p.m., perfectly on time, a delivery boy knocks at James and Sarah's door and delivers their pizzas. James signs a delivery receipt on the delivery boy's PDA which records the delivery time and completes the payment transaction.

James and Sarah have a very nice dinner and at the end, since they are very satisfied of the service received, they decide to recommend the service to other Internet users.

4.2 Pizza Delivery System: Behind the Scenes

`PizzaOverall`, the service chosen by James, is a 'virtual' pizza parlor: it represents a service capable of dynamically discovering and combining actual services in order to accomplish its task. In particular, `PizzaOverall` has to discover and compose services such as an actual pizza parlor, a delivery service, and a payment gateway service.

In order to discover other services, `PizzaOverall` relies on the 'local' registry made available by its service provider. Through the discovery phase, `PizzaOverall` can find services that, once properly composed, can satisfy James's request and meet the related criteria (price range, delivery time, etc.).

`PizzaOverall` finds `PizzaExpress`, a pizza parlor which also offers delivery. The credit card transaction will be handled through `PayBridge`, as most of `PizzaOverall` payment transactions. In this case there was no need to dynamically discover a payment gateway since `PayBridge` is a well-known service, which, in addition, offers to its clients a price per transaction decreasing with the number of processed transactions.

`PizzaOverall` forwards James's order to `PizzaExpress`. `PizzaExpress` starts to bake two pizzas as requested, but then it encounters a problem: its drivers unexpectedly go on strike. `PizzaOverall` recognizes that `PizzaExpress` will not be able to perform the delivery task, so a substitutive service has to be found not to loose James' order.

`PizzaOverall` searches the registry again, this time broadening the search scope: this is possible because the local registry links to other external registries, so service requests can be properly propagated to other registries (e.g., following a topic-based approach). `PizzaOverall` discovers a delivery service, named `PizzaWherever`, which is likely to solve the delivery issue. A delivery is booked in order to pick up the pizzas baked by `PizzaExpress` and bring them to James's place all the same.

`PizzaWherever` has tens of delivery boys spread all over the city, driving bikes or mopeds equipped with wireless devices that they use to receive delivery orders, directions, and to communicate delivery status information to `PizzaWherever`'s central logistic system.

One delivery boy is thus notified to pick up two pizzas at `PizzaExpress`'s parlor at 7.45 p.m., and to promptly deliver them to James's place. He reaches James's apartment at 8 p.m. sharp. `PizzaWherever`, and then `PizzaOverall` in

turn, are notified of the final delivery, as James signs the delivery receipt. All the pending payment transactions are finalized as well.

Another service, named `DeliveryMonitor`, transparently to James and Sarah, has followed the pizza order and delivery process, and is also notified of the time of delivery, then stored in the service history.

4.3 Explaining the Mapping between the Example and the Conceptual Model

James is a *Person* acting as a *Service Consumer*. He uses his PDA to query his ISP *Service Registry* in order to discover *Services* and then invoke the *Operations* they expose. James's ISP is an *Organization* acting as a *Service Provider*.

James's *Service Requests* (e.g., 'have pizza for dinner') can be considered as *Abstract Services*, i.e., they represent the description (more or less detailed) of Services.

Through the discovery phase, James finds one or more *Service Descriptions* of one or more abstract or concrete Services that match or are relevant to his Service Requests. In particular, Service Requests are matched up with the *Service Properties*. Service Properties may belong to the *Service Specification* (i.e., type of oven, price range) or to the *Service Additional Information* (e.g., ratings expressed by previous customers) stored in the Service Registry by means of *Facet* structures. The Service Specification and Service Additional Information form the *Service Description*.

`PizzaOverall`, the Service James has chosen, is actually an *Abstract Service*, i.e., it describes a Service which does not correspond to any fixed concrete implementation. Such an Abstract Service has been published by the ISP itself on its own Service Registry, in order to globally represent a possible way to compose some of the available concrete Services.

James's choice to invoke `PizzaOverall` leads to the concretization of the Service which will actually satisfy his request. `PizzaOverall` is concretized by means of a *Service Integrator*, that is able to perform dynamic Composition of Services, based on the goals to be achieved (i.e., pizza baking, delivery, and payment), and according to one or more specific *Composition Architectural Styles* (e.g., peer to peer). The resulting process is annotated with assertions which enable run-time monitoring. For example, the fact that `PizzaExpress` could not perform the delivery corresponds to a violation of the postcondition of its Operation 'bake and delivery', thus triggering a proper *Recovery Action* (i.e., federated discovery of a delivery service) leading to run-time discovery of a substitute Service. This time the discovery phase involves external Service Registries which are linked by the local registry and form with this a *Federation* of Registries.

`PizzaWherever`'s central logistic system can be seen as a *Legacy System*, which has been enabled to communicate with delivery boy's wireless devices. `PayBridge` is a *Stateful Service*, i.e., the results of its invocation depend also on its inner *Current Service State* (e.g., number of previous invocations by the same consumer).

The feedback provided by James and Sarah enriches the Service Additional Information available for `PizzaOverall`. `DeliveryMonitor`, finally, is a *Service Monitor* that tracks James's order till the pizzas are delivered, and uses proper *Metrics* to measure `PizzaOverall` service properties, such as delivery time, then storing the *Monitoring Data* in the service *History*.

5 Related Work

Several attempts to conceptualize the world of services can be found in the literature, and our work was initially inspired by some of them. In particular, our core model is rooted in the Web Service Architecture (WSA) [6] drafted by the W3C. The WSA conceptual model is structured in four parts each focusing on a specific aspect, namely the service (*Service Model*), messages (*Message Oriented Model*), resources (*Resource Oriented Model*), and policies that can constrain resources and behaviors (*Policy Model*).

In general, the WSA model and our model can be seen as complementary since we do not fully address the message oriented, resource oriented, and policy models of the WSA, but we try to clarify and detail more than the WSA does the concept of service, as well as all the concepts relevant to the service-related activities (i.e., publication, discovery, composition, and monitoring). Also, we try to clarify the relationships between the concepts of service description, semantics, and service interface, while the distinction among these concepts is not evident in the WSA. Moreover, we have choosen a different approach to characterize agents and actors which allows us to express the fact that roles can be covered, in principle, by any agent and vice versa.

Our model also has similarities with the Service-Oriented Solutions Approach (SOSA) [7] proposed by Computer Associates International, Inc. technology services department. The SOSA conceptual model is part of a method that aims to maximize the potential of Web services and SOA within medium and large enterprises. Such method is based on best practices (e.g., tracks, techniques, work packages, and deliverables) for service-oriented development [7]. The SOSA model has a complementary relationship to our model since it focuses more on service interfaces and business oriented issues, while it is less detailed with respect to other aspects related, e.g., to the publication, discovery, and execution of services.

Our model is quite different in objectives and scopes to other works such as OWL-S [4] and the Web Services Modeling Ontology (WSMO) [8]. A first difference between our model and OWL-S stands in their different objectives. Our model provides a common understanding for human readers about the main actors, entities and artifacts that are somehow involved in the creation of a service-centric system. On the contrary, the OWL-S ontology was created to provide a computer-interpretable description of a service (particularly, web-based services), to allow software agents to discover, invoke, compose, and monitor Web resources offering services having particular properties. As a consequence, the OWL-S ontology pursues a very detailed service description suitable for the

needs of software agents. On the other hand, our model tries to embrace a larger application domain than OWL-S, i.e., the overall set of main actors and concepts involved in the various steps of the service-centric system creation process.

The WSMO, in line with the Web Services Modeling Framework (WSMF) [9], aims at providing a conceptual model for developing and describing Web services and their composition by means of a language (Web Services Modeling Language, WSML) and an execution environment (Web Services Modeling eXecution Environment, WSMX). The WSMF consists of four different main elements: ontologies that provide the terminology used by other elements, goal repositories that define the problems that should be solved by Web services, Web services descriptions that define various aspects of a Web service, and mediators which bypass interoperability problems. The WSMO extends these main elements by defining a set of crosswise non-functional properties named core-properties. WSMO mainly focuses on service descriptions, i.e., pre and post-conditions, non-functional properties, etc. Differently from our model, it does not provide a conceptual model of some key activities of a service-centric scenario, such as discovery, delivery, and monitoring.

6 Conclusions

The aim of this work is to provide a conceptual model that is complementary to the ones already presented in the literature, and is focused on the main issues concerning the development and operation of service-oriented systems.

We are currently enacting the adoption of the model within the SeCSE project as a unique reference for definitions and main concepts for the whole consortium. We experimented the first release of the model by having the other project partners check if their main ideas, requirements, and technical solutions fit into it. All partners provided comments and inputs that will be included in the next releases.

The model now plays a key role in the project, since it is used as a means for exchanging ideas and results in a coherent framework, thus helping every partner to better achieve the project goals (e.g., the development of a platform supporting the life cycle of SOA-based solutions).

The interest of the project partners, their willingness to participate in our discussions, and the number of debates we are still triggering convince us that the model can evolve to become a good common language, not necessarily limited to the SeCSE project.

Our model is already being exploited by the European Commission (Directorate D – Network and Communication Technologies, Software Technologies) as a framework to classify and explain the European projects related to service development [10].

Acknowledgements

This work is framed within the IST European Integrated Project *SeCSE (Service Centric Systems Engineering)* [3], 6th Framework Programme, Contract No. 511680. We thank all our partners in the project for their valuable comments and proposals aiming at improving the conceptual model.

References

1. Hong Gao, T., Huffman Hayes, J., Cai, H.: Integrating Biological Research through Web Services. IEEE Computer **38** (2005) 26–31
2. de Mes, A., Rongen, E.: Technical note: Web service credentials. IBM Systems Journal **42** (2003) 532–537
3. SeCSE Website: *http://secse.eng.it/* (2005)
4. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. W3C Member Submission (2004)
5. Colombo, M., Di Nitto, E., Di Penta, M., Distante, D., Zuccalà, M.: Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. Technical report, RCOST (2005) *http://www.rcost.unisannio.it/mdipenta/cm.pdf*.
6. W3C: Web Services Architecture (WSA). W3C Working Group Note 11 February 2004. (2004)
7. Lefever, B.: Service-Oriented Solutions Approach (SOSA). Technical report, Computer Associates International, Inc. (2005) *http://www.ca.com/be/english/past-events/lunch-s3/041209-sosa-lb-final-lefever.pdf*.
8. de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Kifer, M., König-Ries, B., Kopecky, J., Rubén, L., Oren, E., Polleres, A., Scicluna, J., Stollberg, M.: Web Service Modeling Ontology WSMO (2005)
9. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce: Research and Applications (2002) 113–137
10. Sassen, A.M., Macmillan, C.: The service engineering area: An overview of its current state and a vision of its future. European Commission, Directorate D – Network and Communication Technologies, Software Technologies (2005) *ftp://ftp.cordis.lu/pub/ist/docs/directorate_d/st-ds/sota_v1-0.pdf*.