A Pragmatic Approach for Analysis and Design of Service Inventories

Patricia Lago and Maryam Razavian

Department of Computer Science, VU University Amsterdam, The Netherlands
{p.lago,m.razavian}@vu.nl

Abstract. Service Orientation is a paradigm for exposing business functions as reusable services and enabling business partners to discover those services on demand. Several service-oriented methodologies have been proposed in both academia and industry. Few methodologies, however, do provide concrete yet pragmatic support for developing software that is truly service-oriented, supporting service-specific aspects and challenges. As a consequence, the design of software services that are in-line with service aspects is left up to the prudence of the developers. To fill this gap, this paper proposes a methodology that incorporates essential ingredients of SOA during analysis and design. The methodology focuses service inventories while consciously addressing the perspectives of both service providers and service consumers. By supporting service-oriented reasoning, our methodology results in better reusable services, and the available inventories aid the development of service-oriented systems in a more cost-effective way. This is of the essence in cloud computing, which is a fast-spreading business model promising high ROI and cost savings.

1 Introduction

Nowadays, Service Orientation is having an impact on application development similar to that brought by Object Orientation in the nineties [1]. Both paradigms went through the *peak of inflated expectations* coined in the Gartner Technology Hype Cycle. While Object Orientation reached mainstream adoption long ago, Service Orientation is still, in our opinion, in the *slope of enlightenment* [2]: the *real* benefits of SOA are now becoming more widely understood, and more and more companies are migrating their software assets to SOA technologies.

In this maturation process, many so-called SOA methodologies have been defined in both industry and academia. Unfortunately, few (if any) do provide concrete support for the development of software that is *truly* service-oriented [3]. With term *service-oriented software* we mean both software services, and service-based applications (SBAs) reusing (i.e. composing) them. We argue that existing SOA methodologies have the same underlying assumptions as traditional software engineering methodologies, i.e. they assume that the developer both has ownership on the software (i.e. services), and has a system model in mind carrying holistic definition of all functionalities that should be produced. Both assumptions are of course not true in service-oriented (SO) development and cloud computing (CC) where services are neither owned nor always part of a "monolithic" system. According to [4], the introduction of the SO paradigm introduced a shift in the way we conceive a "software system": from a large system to a set of small pluggable elements. We therefore, argue that the subject of the architecting process should also shift from a system to the building blocks (i.e. the services); SO methodologies should be in-line with such shift. To support a software architecture following the service-oriented paradigm, a SO methodology should cover some essential ingredients:

Support Both Service- and Application Development. Developers of service-oriented software can play the role of either the service provider or the service consumer (or both). The *service provider* typically develops reusable software services. In doing that, it might reuse (or not) other available services, to build its own service compositions. In the latter case, the service provider switches to a service consumer role (described below) that develops more complex services. In any case, the output of the development is an *inventory of independent services* (or service pool) meant to be published for (internal or external) reuse. In this *service provider perspective* the development challenge is to identify *those* essential business functions that should be added to a service inventory, without knowing the business logic of the system/SBA that will reuse them.

The *service consumer*, instead, typically develops SBAs. In doing that, it always reuses services provided by third parties. The output of the development is in this case a software system meant to be purchased to customers and directly used by end-users. In this *service consumer perspective* the development challenge is to identify the characteristics of the services to be reused, and design and application that will dynamically discover and compose them, and still deliver a reliable overall system.

Focus on SOA. SOA is an architectural style that supports Service Orientation [4]. As such, it should (implicitly or explicitly) support (technical) mechanisms for service publication, dynamic discovery and composition.

Aim at Software Services. At a conceptual level, a *service* can be defined as a logical representation of a repeatable (business) activity that has a specified outcome (e.g. check customer data, provide weather report, deliver pizza, provide higher level education). A service is self-contained (state-less and adhering to a service contract); may be composed of other services (service composition); and is a black-box to its consumers.

Embrace the "Open World Assumption". Software services yield distributed ownership [5], i.e. they are often owned by (other) service providers. Developing (own) SBAs or service compositions hence implies that reuse is planned at design-time but can be actually tested only at run-time. Modern SO development must hence support a mix of design- and run-time development activities to make sure that dynamic aspects are engineered well and that the resulting software is reliable.

While all four ingredients described above are relevant, the major problem in current SOA methodologies is in the first one. The following describes our methodology for SO analysis and design of service inventories. The service inventories, as such, provide the building blocks for a successful service offer following a SO paradigm. While we explicitly address the perspective of a service- or cloud provider developing inventories of software services (i.e. service provider perspective), we also need to support a service consumer perspective whenever services are to be reused from other providers. Both services and SBAs make up the service offer in a CC paradigm.

2 Terminology Overview

Services are the building blocks in our methodology. During the development lifecycle they get different forms. For the sake of clarity, in this section we define the different types of services as well as other basic concepts used in our methodology.

Conceptual Service: a conceptual service that is implementation agnostic. Considering the perspective of our methodology all services are conceptual as we do not address implementation of the services.

Business Service: a self-contained, stateless business function that accepts one or more requests and returns one or more responses through a well-defined, standard interface. During service identification, some elicited business services become the *service candidates* for design. They also might become *service operation candidates* to be clustered in service candidates.

Service Candidate: conceptual service identified during analysis that is a candidate software service.

Service Operation Candidate: a service operation identified from functional requirements; it might become a service candidate itself, or be composed (i.e. aggregated) in a (more complex) service candidate.

Task Service: service that mainly executes a functionality.

Entity Service: service that mainly manages, and offers access to, a (complex) data resource.

Hybrid Service: a mix of *task service* and *entity service.*

Utility Service: Domain- or application independent service offering access to generic functions or generic data resources. Also called *Infrastructure service.*

3 Methodology and Its Supporting Models

Service orientation is shifting the focus from engineering systems to integrating existing software services. During software development enterprises play both service provider and service consumer roles. Nevertheless, depending on the product they aim at producing (being a service inventory or a SBA), the methodology to be followed changes considerably. In this work we aim at producing service inventories. To this end, we identify the essential steps to carry out two phases: *SO analysis* is here defined as the process of determining how business requirements can be represented through business service candidates, while *SO design* is the process of modeling a software service inventory and/or reusing it to compose a SBA. In these phases, for modeling purposes we reused state-of-the-practice notations like UML and SoaML, and extended them only where they revealed to be

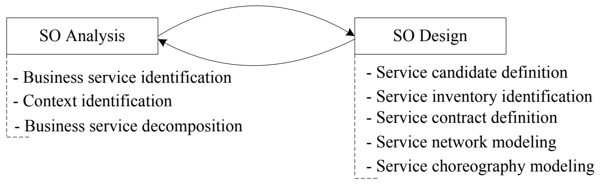

SO Analysis
SO Design
- Business service identification
- Context identification
- Business service decomposition
- Service candidate definition
- Service inventory identification
- Service contract definition
- Service network modeling
- Service choreography modeling

Fig. 1. Service Inventory Analysis and Design Methodology

insufficient. Fig. 1 depicts the steps of our SO Analysis and Design methodology, as further elaborated in the following.

Step 1: Business service identification. This step aims at identifying business services. These are elicited by use of business process models and conceptual data models. These two models can be determined either from functional models of pre-existing systems (i.e. bottom-up SOA migration), or from the target business domain, as the list of functional requirements (i.e. top-down service development).

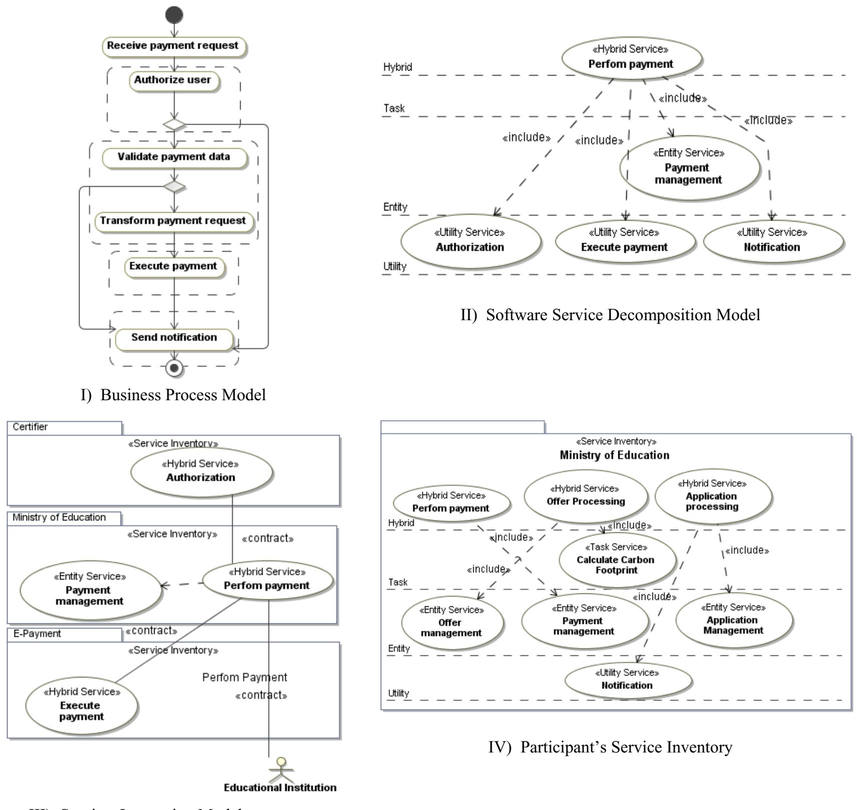
Given the business processes, we identify business services by clustering service-relevant functionality within a business process. The elements of a business process model (e.g. activities and decision elements) are examined as candidate business services. The goal here is identifying the elements representing a self-contained functionality. Accordingly, business process models represent a sequence of business services, and hence, highlight potential orchestration behavior. To model business processes, we use UML activity diagrams (see Fig. 2.I)[1]. The clusters of functionality representing the business services are shown by dashed boxes. By modeling clusters in an explicit way, the analyst is helped in recognizing/eliciting relevant candidate services. Also, modeling clusters supports the analyst in comparing similar clusters for either merging them in a unified functionality, or keeping them separate.

The second model especially important during this step is the conceptual data model. This model facilitates the identification of the business services addressing the functionality of business data entities. The conceptual data model elements (i.e. data entities and relationships) are examined as candidate business services. These business services are highly reusable since they can be leveraged in independent business processes following the principle of separation of concerns and abstract data types (as in object orientation).

Step 2: Context identification. The main goal of this step is the identification of the set of external elements that the service inventory interacts with. Examples of such external elements are: external service providers, external services or SBAs, or external end users. In this step it is important to identify the external elements that rely on this business service inventory as well as those business services that rely on the external elements. The output of this step is a context model that illustrates the inventories in their environment. To

[1] In Fig. 2 we focus only on the models that are instrumental for the description of the methodology and that have been extended from standard UML and SoaML notations in order to support better analysis- and design reasoning.

draw context models we use the UML use case diagram notation. The context model represents the business services using *UML use cases*, the boundary of the inventory is illustrated using *use case system boundaries*, and the inter-relations among external entities and business services are shown by *associations*.



Receive payment request
Authorize user
Validate payment data
Transform payment request
Execute payment
Send notification
Hybrid
Task
Entity
Utility
«Hybrid Service» Perfom payment
«include»
«Entity Service» Payment management
«Utility Service» Authorization
«Utility Service» Execute payment
«Utility Service» Notification
Certifier
«Service Inventory»
«Hybrid Service» Authorization
Ministry of Education
«contract»
«Entity Service» Payment management
«Hybrid Service» Perfom payment
E-Payment
«Hybrid Service» Execute payment
Perfom Payment
Educational Institution
«Service Inventory» Ministry of Education
«Hybrid Service» Perfom payment
«Hybrid Service» Offer Processing
«Hybrid Service» Application processing
«Task Service» Calculate Carbon Footprint
«Entity Service» Offer management
«Entity Service» Payment management
«Entity Service» Application Management
«Utility Service» Notification

I) Business Process Model

II) Software Service Decomposition Model

III) Service Interaction Model

IV) Participant's Service Inventory

Fig. 2. Examples of Analysis and Design Models

Step 3: Business service decomposition. Given the set of identified business services, this step elicits the *candidate services* and their constituent *service operations*. This step results in the selection of the business services, identified during the previous step, that will be (partially) automatized by means of software services. Those business services are transformed into a number of candidate services or service operations. As the output of this step, each business service is modeled as a collection of candidate services. We model the decomposition of candidate services using the UML use case diagram notation.

Step 4: Service candidate definition. This is the first step of the SO Design phase. In this step, the service candidates of each business service (output of SO Analysis) are mapped on the service types (i.e. hybrid, task, entity and utility - refer to Section 2 for our definitions), from the perspective of a service provider.

As an example, Fig. 2.II shows the service candidates of the *Payment* business service with their corresponding service types: *Payment management* is an entity service managing the data about a payment, while *Authorization*, *Execute payment* and *Notification* are all utility services reusable across domains or applications. By identifying the service types the scope of reuse (i.e. domain-specific vs. domain-generic) of the services is identified. In this way, the service consumer is helped in reasoning about, e.g. if a service has been designed to be reused across different SBAs (utility service), or if it manages internally application-specific data (entity service) or application-specific activities (task service). This powerful yet simple modeling mechanism addresses an urgent problem in industrial practice, namely the need to reuse services developed for different purposes (e.g. in different projects or departments) and yet very similar in nature. Lack of support in this step leads to unnecessary duplication of services (silo problem) and governance problems, both major issues in IT service developing companies nowadays.

Step 5: Service inventory identification. During this step, decisions are made on which services are going to be provided by the inventories. Service inventory identification involves making a choice for the realization of service candidates. The realization alternatives include, but are not limited to, the following: *a)* reuse an existing element (e.g. component) and transform it as a service using wrappers;*b)* develop the candidate services from scratch;*c)* outsourcing the service realization; *d)* or purchasing, leasing, or paying per use (for) a service. The first three alternatives imply that the corresponding service will be provided by the service inventory. The last alternative, however, indicates that the service inventory, itself will be the consumer of that specific service which is provided by another provider. By mapping the candidate services on different service providers, the *participants* of the service candidates are identified. Fig. 2.IV represents the service inventory of the *Ministry of education* that consumes the services provided by *Certifier* and *e-Payment.* This way the participants of the Ministry of education service inventory are identified, i.e. *Certifier* and *e-Payment.* The output of this step is the service inventories representing the collection of software services of different types that each participant offers.

Step 6: Service contract definition. This step aims at identifying the service contracts between participants. As mentioned, cross-domain service interactions represent candidate service contracts. These interactions are inherently the ones crossing the participants' service inventories and the interactions can be between participants and external entities or between participants only. Given a specific business service, a contract represents how participants work together to realize the business service's goals. To identify the service contract this step models each business service with service interaction diagram (see Fig. 2.III). Using service interaction diagrams, the software services realizing each business service are mapped on their provider participant and further the interactions crossing participant's domains are identified as contracts, which are then modeled using 'SoaML contracts'. Thanks to this step, and as made explicit in the service interaction diagram, service providers are given with an holistic overview of what their service inventory is offering to potential consumers and what are the existing (or to be established) service level agreements with third-party service providers. In this way, SO reuse and governance are supported. On the other hand, service consumers (or service providers planning to consume third-party services in the service compositions) are helped in reasoning about the impact of consuming a certain service offered by a (discovered) service provider.

Step 7: Service network modeling. Given the contracts identified in the previous step, this step models how participants work together to realize each business service using their contracts. The network architecture illustrates which participant is the provider of the contract and which one is the consumer. We model services networks using SoaML services network architecture diagram. It should be noted that, the services network architecture is modeled in a recursive manner. As such, a participant providing a service in a higher level network architecture can itself have a network architecture that shows how that service is provided using its software services.

Step 8: Service choreography modeling. Given the participants of each business service and service contracts among them this step defines the service choreography for each service contract. The service choreography model here represents the contract's behavior in terms of what is transmitted between the parties and when, without defining the internal behavior of the parties. Similar to service networks models, the choreography models are also defined in a recursive manner. This way, each participant defines the internal behavior of its own provided services. Service choreographies are modeled using SoaML choreography diagram and the internal behavior of the software services are modeled using the UML sequence diagram.

4 Extensions to UML/SoaML

UML/SoaML models revealed to be insufficient to properly support SO reasoning. To solve this problem, we extended some of the UML/SoaML diagrams by defining a UML profile, which implements the extensions to the diagrams used in the methodology. The following explains such insufficiencies and briefly describes the applied extensions.

Service Types. One of the shortcomings of SoaML is its lack of support of service types and their structuring. To fill this gap, our approach extends SoaML use case models in order to support mapping service candidates on service types. Each software service, modeled using use cases, is marked with a stereotype representing the type (see Fig. 2.II).

Service Inventory Model. Although service inventory is a first class element in SO design, SoaML does not support it. To fill this gap we use UML use case diagram and cluster software services to model the service inventories of each participant (see Fig. 2.IV) using packages marked with $<< ServiceInventory >>$ stereotype.

Service Interaction Model. While modeling collaborations through service contracts, SoaML also lacks in providing a model facilitating the identification of service contracts to be designed. To fill this gap, we devised the service interaction diagram (see Fig. 2.III). As mentioned, cross-domain service interactions represent candidate service contracts. We model such interactions using use case associations marked with << *Contract* >> stereotype. It should be noted that in service interaction model, SBAs that just have the service consumer role can be directly modeled using UML actors.

Internal Service Behavior. Regarding service behavior, SoaML focuses on choreography only. Hence the internal behavior of the provided services is not covered. Considering the service provider perspective, both cross-domain behavior (modeled using SoaML contract-based choreography) and internal behavior of provided services need to be addressed. To cover the internal behavior, we exploit UML sequence diagram, which hence complete both perspectives of a SO design, that of a service provider developing its own services (or a service consumer developing its own SBAs) and that of a service consumer reusing externally provided services (or a service provider developing its own service compositions by reusing, again, externally provided (atomic) services).

5 Discussion

Many SO approaches for design and development of services have been proposed in both industry and academia. Some of these approaches support the entire service engineering process (e.g. [6,7]), while some others cover only a few phases such as service analysis and design (e.g. [8,9]). Our approach falls under the second category by specifically covering service analysis and design.

According to the survey carried out by Gu&Lago [3], the phases of service analysis and design are covered by most of the SOSE methodologies. Those methodologies, however, mainly provide guidelines rather than defining a fully fledged methodology that guides architects and designers step-by-step. For instance, most of the existing methodologies provide guidelines for service identification from different resources such as requirements or existing legacy systems. Those methodologies assume that by identifying the candidate services, the design of services can be done in a straightforward manner. Unfortunately, devising a solid design that supports requirements and goals and is in-line with SOA principles is not straightforward and needs to be aided in a step-by-step manner. SO methodologies should support the line of reasoning needed for achieving a service architecture design from the requirements. This is one main contribution of our methodology.

In our methodology we put special emphasis on modeling (in the most straightforward and pragmatic way possible) the elements that are left implicit in the existing SO notations and that in our experience aid reasoning. For example, modeling the clusters of activities in the Business Process Model of Fig. 2.I) helps analysing how to identify reusable business services; modeling the different inventories in the Service Interaction Model of Fig. 2.III) is necessary to elicit which dependencies among services cross different inventories, and therefore require the establishment of contracts; etc.. Thanks to such modeling extensions, we bridge the gap between the 'description of the artifacts' and 'how to create those artifacts'. According to Tang [10], if a design problem is well-structured, designers tend to have a better reasoning. Our approach supports such structure in the inventory design problem, by supporting all essential seamless steps required for the reasoning throughout the service analysis and design. We have been using and refining it in the past five years, by teaching it to Master students and letting them apply it in their SO design practicals. As the students need to be trained in their reasoning, we observed the problems they were encountering and refined the way we support them in realizing their SO design. We also presented our methodology to various industrial partners who all expressed vivid interest to use it for their clients migrating to CC business models. We are currently applying it in a 700M Euro project in the field of airlines and airports, with great feedback so far.

Throughout the steps of our methodology, we support the perspectives of both the service provider and the service consumer. We support the service provider in seamlessly transforming business requirements into business services and all the way down to their supporting software services. In this way, we ensure that the resulting services expose the right functionality with the desired business value. In addition, we explicitly support the provider in analyzing its service inventory and the existing or needed contracts with third-party providers. We support the service consumer in identifying the characteristics of the services to be reused via e.g. service types, the explicitly clustering of business services, and their mapping on software services. Also, by supporting the identification and design of service contracts we support the provider that consumes external services for service compositions - this time the service provider and service consumer perspective together.

According to [3], most of the existing methodologies provide some guidance on the required models and notations. Those modeling methods, however, do not fully support the associated activities of the methodologies. This implies that some activities are not supported by the required modeling techniques. For instance, in order to identify the services contracts, one needs to identify the interactions among the services provided by the participants. To identify these interactions, a modeling technique must represent the participants, their provided services and the interactions among them. Although such a modeling technique is required, it is not supported by existing modeling notations. This reveals gaps in existing SO modeling notations. Our work identifies such gaps and fills these gaps by extending SoaML and UML.

Finally, by supporting the identification of business services and their transformation into supporting software services, our methodology helps bridging business requirements and IT solutions. This alignment of business and IT solutions has been one of the promises of SOA. Unfortunately, industry still suffers major problems in realising it. For example, while SOA promises to ease the communication of business- and IT stakeholders, business requirements coming top-down from the first are often not well understood by the latter, and hence realized ad hoc. This hinders effective reuse of software services and causes the 'silo problem' (e.g. similar business services across different development projects are not identified and hence their development is duplicated). Ultimately, this causes governance problems. Our methodology is a first step to help stakeholders in achieving business-IT alignment by shaping the SO Design phase around the notion of business service.

6 Conclusion

In this paper we introduced a pragmatic modeling methodology focused on SO analysis and design. This methodology and its models stem in five years of teaching software- and service oriented design. It has been applied each year on different large industrial design projects and underwent continuous improvement. Recently we are using it also in collaboration projects with industrial partners, where it is catching increasing and positive attention.

Our methodology has various novelties. It is simple, pragmatic, focusing on the essential steps to go from business requirements to a design blueprint. It makes explicit the two service consumer- and provider perspectives by explicitly addressing the identification of service inventories and the offered services behavior (for providers), and by developing cross-domain interactions with services networks and contracts (for consumers). In this way, architects are compelled to *think and reason in a SO way* and are facilitated in better decision making driving migration toward CC business models.

Acknowledgments. This research received funding from projects Jacquard SAPIENSA (contract 638.001.206) and FP7 NoE S-Cube (contract 215483).

References

1. Cockburn, A.A.R.: The impact of object-orientation on application development. IBM Syst. J. 32, 420–444 (1993)
2. Gu, Q., Lago, P.: Exploring service-oriented system engineering challenges: a systematic literature review. Service Oriented Computing and Applications 3, 171–188 (2009)
3. Gu, Q., Lago, P.: Guiding the selection of service-oriented software engineering methodologies. Service Oriented Computing and Applications, 1–21 (2011)
4. Bell, M.: Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley Publishing (2008)
5. Baresi, L., Di Nitto, E., Ghezzi, C.: Toward open-world software: Issue and challenges. Computer 39, 36–43 (2006)
6. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Gariapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. IBM Syst. J. 47, 377–396 (2008)
7. Papazoglou, M.P., Heuvel, W.J.V.D.: Service Oriented design and development methodology. Int. J. Web Eng. Technol. 2, 412–442 (2006)
8. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design (2004)
9. Allen, P.: SOA best practice report: the service oriented process. Technical report, CBDi Journal (2007)
10. Tang, A.: Software designers, are you biased? In: Proceedings International Workshop on Sharing and Reusing Architectural Knowledge, p. 8. ACM (2011)