Pattern-based Context Establishment for Service-Oriented Architectures *

Kristian Beckers, Stephan Faßbender, Maritta Heisel, and Rene Meis

paluno - The Ruhr Institute for Software Technology -
University of Duisburg-Essen, Germany
{firstname.lastname}@paluno.uni-due.de

Abstract. A context description of a software system and its environment is essential for any given software engineering process. Requirements define statements about the environment (according to Jackson's terminology). The context description of a Service-Oriented Architecture is difficult to provide, because of the variety of technical systems and stakeholders involved. We present two patterns for SOA systems and support their instantiation with a structured method that guides that instantiation. In addition, we show how the pattern can be used in a secure service development life-cycle (SSDLC).

Key words: SOA, requirements engineering, secure software development

1 Introduction

Nowadays Service-Oriented Architectures (*SOA*) as part of Service-Oriented Computing (*SOC*) is a well known paradigm with raising importance [**?**,**?**]. The subject of SOC is vast and enormously complex, and SOC is based on many concepts which have their origin in diverse disciplines. SOA and software engineering (*SE*) for SOA are among the most important research fields for SOC [**?**].

Still, there are some open issues when dealing with SOA. One of them is security [**?**,**?**]. Assurance of security of a SOA is much more complicated than for other architectural styles. Since the single services used are normally distributed and loosely coupled over the Internet, standardized protocols for link-up are needed [**?**]. Another field for security issues is the fact that in common SOA scenarios, not a single person or company controls all infrastructure and services that are orchestrated [**?**,**?**]. Hence, when dealing with security for SOA, we not only have to face the usual security problems as in other IT systems, but additionally multilateral settings with many stakeholders in a distributed environment have to be considered, which even more complicates finding a solution.

*This research was partially supported by the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980).

The big number of very heterogeneous stakeholders also complicates the fulfillment other objectives, such as compliance, performance, or usability. For example, when considering compliance, each stakeholder may introduce new regulations to be met by the SOA. Such a relation can be a law, which protects this stakeholder, or requires the stakeholder to ensure a certain functionality. Additionally, most of the stakeholders, like a service provider, have a contractual relationship to the SOA to be developed.In a previous paper [**?**], we have argued that for legal requirements engineering , it is crucial to capture the stakeholders and the entire context of an IT system.

To sum up, from a software engineer's perspective, there is a need for a continuous and integrated method to consider multiple stakeholders and their different objectives in a SOA development life cycle. This method should be transparent in means of documentation and traceability to make it possible, for example, to trace a compliance regulation down to the measures taken to accomplish it.

This requirement sounds simple, but when looking at standard SE life cycles, such as Microsoft SDL [**?**] or CLASP [**?**], we see that there are so many steps with dependent in- and outputs to be performed, that achieving this requirement is very challenging.

Achieving transparency between requirements of a stakeholder, measures and proof that the measures have been implemented, means using methods that are aware of traceability for each activity or transformation or refinement. For some steps and some parts of, for example, security, such methods already exist, but missing an appropriate approach at one point brakes the whole trace. Hence, it is an important goal to analyze the whole SE life cycle, find the gaps and propose possibilities to bridge those gaps. It is surprising, that in most cases the description of the very first steps of such a life cycle is deficient. How to capture and describe the setting of a problem and to structure the context of the system-to-be is often missing. But this information is of crucial importance for understanding the problem and performing requirements engineering.

Hence, in this paper, we focus on establishing the context of a SOA application, which forms the basis for all later development steps, especially considering security.

The rest of the paper is organized as follows. Sect. 2 describes a small use case in the field of media publishing, which we use to illustrate our ideas in this paper. In Sect. 3 we give a brief background about SOA and its layers. Two patterns and the corresponding textual templates are introduced in Sect. 4. The patterns and templates help to establish the context of a SOA. Sect. 5 introduces a method to instantiate the patterns and templates. A short overview of the application of our method to our case study is presented in Sec. 6. Finally, we outline the use of our method in the context of Microsoft SDL [**?**] and CLASP [**?**] in Sect. 7, and we give a short conclusion in Sect. 8.

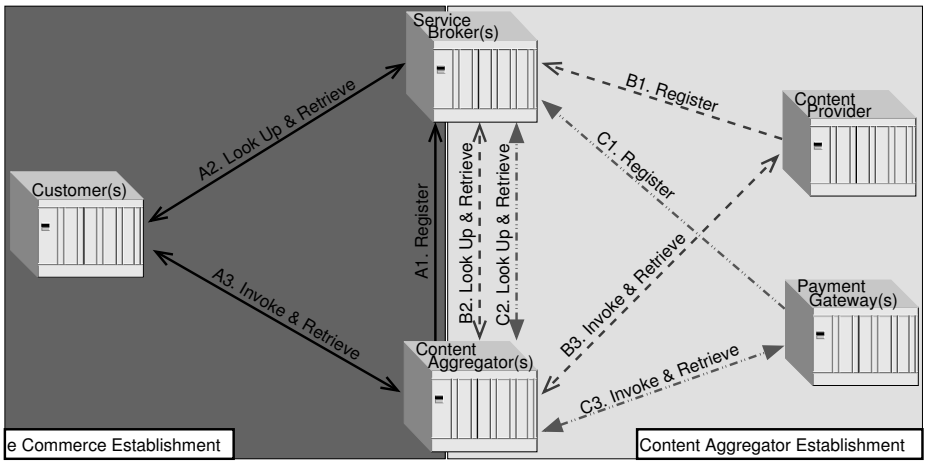

Fig. 1: SOA Scenario
A=Business Case,B=Content Provider Integration,C=Payment Gateway Integration

2 Running Example

For our running example, we have chosen a media publishing setting. In this setting, there are customers, who want to retrieve certain media. For example, this media can be a piece of software, a video or movie, a song, or an e-book. On the other side, we have various content providers. A small content provider may offer only one media type and only a small selection of media to choose from. In contrast a big publisher usually offers all media types and a big selection of media.

The main problem in the relationship between customers and content providers is that on one hand the customers prefer a uniform search and access interface and do not want browse a big number of different shops, with different access technologies, credentials, and so on. A second problem for customers is to oversee the whole market. Customers might not even know the right content provider for very special media. On the other hand, not all content providers are able or willing to set up and maintain a shop infrastructure with essential functionality such as billing.

The business idea of our example is to introduce a content aggregator as a mediator between these two sides. The aggregator collects the offers of different content providers. These joined offers are then made available to the customers.

The content aggregator decides to choose a SOA to realize its business, because of the dynamics in this setting. There is huge number of providers. The aggregator wants to be able to find and integrate these providers at run-time. Moreover, the access for the content providers to the content aggregator should be as simple as possible. Therefore, services are reasonable. Services enable the providers to wrap their existing technologies and use standard protocols.

A second reason for using services is the fact that the content provider has no direct access to the devices the customer use to search for media. A device in this case can be, for example, a smartphone, a settop box, or a tablet PC. So the content aggregator has to ensure that customer can find the solution the aggregator provides. And the content aggregator has to ensure that the technology used for providing the solution can be easily adapted for each device. Services are a good choice to achieve both goals. In Sect. 3 we shall provide detailed arguments supporting this claim.

A last reason for using services would be the large number of different banks and online payment systems, which have to be integrated to fulfill the payment needs. But this requirement is already addressed by so-called *payment gateways*, which aggregate all banks and make payment functionalities available at a single point. These payment gateways offer their functionality also by services. At this point, we use the best practice already established in the market.

The resulting scenario is shown in Fig. 1. The figure shows the common service look up and invocation process. There are three different instances (A, B, C) of the Register, Look Up & Retrieve, and Invoke & Retrieve sequences in our scenario. The content aggregator queries a service broker to find content providers (Arrow B2 in Fig. 1) and payment gateways (C2) to establish its business. Content providers (B1) and payment gateways (C1) registered themselves at the service brokers before. At run time, the content aggregator invokes the services of the providers (B3) and gateways (C3). Note that the set of gateways will be very static and slowly evolving as there are not that many payment gateways, and establishing the needed service-level agreements (SLAs) and trust cannot be achieved on the fly. In contrast, the set of content providers can be different for each business process invocation. The market of providers is changing fast, and standard SLAs can be used, so service invocation can be done automatically. To realize the business case, content aggregators register their service at a service broker (A1). The customers are now able to find the aggregators (A2) and search for and retrieve (A3) the desired content.

3 Background

In this section, we give a definition of our notion of SOA and how a SOA is structured.

3.1 SOA Definition

Various definitions of SOA exist, because the SOA concept spans a wide field of research areas and technologies. However, there is a common understanding about some core characteristics of SOA. First, a SOA is modular with a high autonomy of its parts, not only in the sense of interaction within the architecture, but also in the sense of e.g. autonomous stakeholders and development teams [**?**,**?**,**?**]. Second, services have a coarse granularity, encapsulating more or less complex tasks. As a result, a single service is a complex product [**?**].

Third, SOA is process-driven [?]. In most cases, a service performs one activity of a business process [?]. Hence, a SOA has to be designed to fulfill business requirements and goals [?]. Fourth, the services of a SOA have to be loosely coupled [?]. The business processes to be supported by a SOA change frequently. In consequence, a SOA has to evolve dynamically [?]. Hence, the services are loosely coupled to enable dynamic (re)assembly. Fifth, the re-usability of services is high [?,?,?]. The re-usability is a result of the autonomy of services and the loose coupling between them. Sixth, a SOA is a distributed system [?,?]. The reasons are that business processes cross the border of one enterprise [?] and that services can be offered by third-party service providers [?]. Seventh, SOA is technology-independent [?,?]. SOA is meant to integrate highly heterogeneous services, which means that the used technologies to implement the services can differ. Summarizing, SOA can be characterized as a process driven, modular, technology-independent, dynamic and distributed system, which relies on reusable, autonomous, loosely coupled and coarse-grained services.

3.2 SOA Layers

A SOA spans different layers, shown in Fig. 2. The first and top layer is the Business Domain layer, which represents the real world. It consists of Organizations, their structure and actors, and their business relations to each other. The second layer is the Business Process layer. To run the business, certain Processes are executed. Organizations participate in these processes. A process can be executed within one organization or across organizations' borders. These processes are supported by Business Services, which form the the Business Service layer. A business service encapsulates a business function, which performs a process activity within a business process. Besides atomic business services, there are also composite business services, which rely on other business services. These services are built by composing other business services. All business services rely upon Infrastructure Services, which form the fourth layer. The infrastructure services offer the technical functions needed for the business services. These technical functions are implemented especially for the SOA or expose interfaces from the Operational Systems used in an organization. These operational systems, like databases or legacy systems, are part of the last SOA layer at the bottom of the SOA stack. Each of the presented layers in isolation is complex and a field for research on its own. Combining these layers and devising a SE method to span the layers is a great challenge in the field of SOA research [?].

4 Patterns for SOA

We now introduce two patterns, which help us to describe the setting of a SOA, namely a SOA layer pattern, and a SOA stakeholder pattern. They are used to structure the information about the problem at hand. The patterns are based on the insights we gave in Sect. 3. In Sect. 4.1 we present a pattern, which uses the SOA layers to structure the SOA itself. Based on this pattern, we present an extended pattern in Sect. 4.2, which adds different kinds of stakeholders.

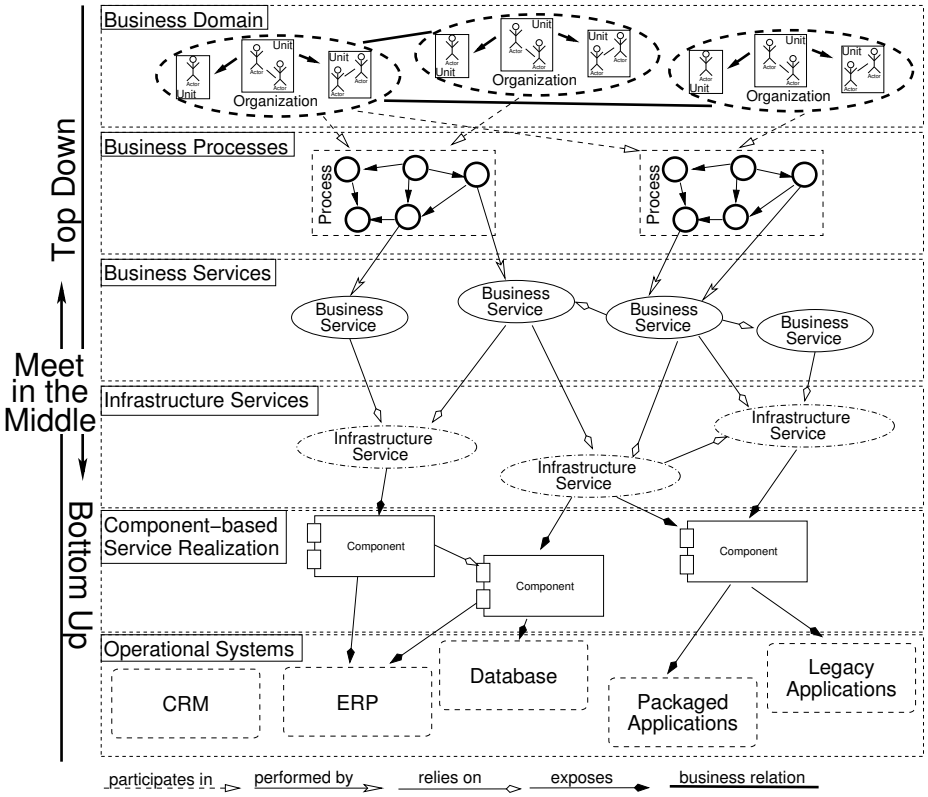

Fig. 2: SOA Layers (based on [**?**,**?**])

4.1 SOA Layer Pattern

The layers presented in Sect. 3.2 form a generic pattern to describe the essence of a SOA. Three approaches for instantiating this pattern are possible, as shown in Fig. 2. When building a SOA from scratch, a Top Down method should be used. The existing information about the organizations and processes and the related requirements can be refined to an architecture comprising business services, infrastructure services, components and operational systems. Note that for the early SE phases such as requirements engineering, not all layers are of relevance. For example the operational systems are chosen in later phases. A Bottom Up approach is of use when evolving existing systems. When the redescription and the reuse of already productive systems is the focus, one would collect the existing artifacts in the IT landscape, describe the existing processes they support and elicit the involved organizations. For developing a new SOA based on existing systems, a Meet in the Middle method is reasonable.

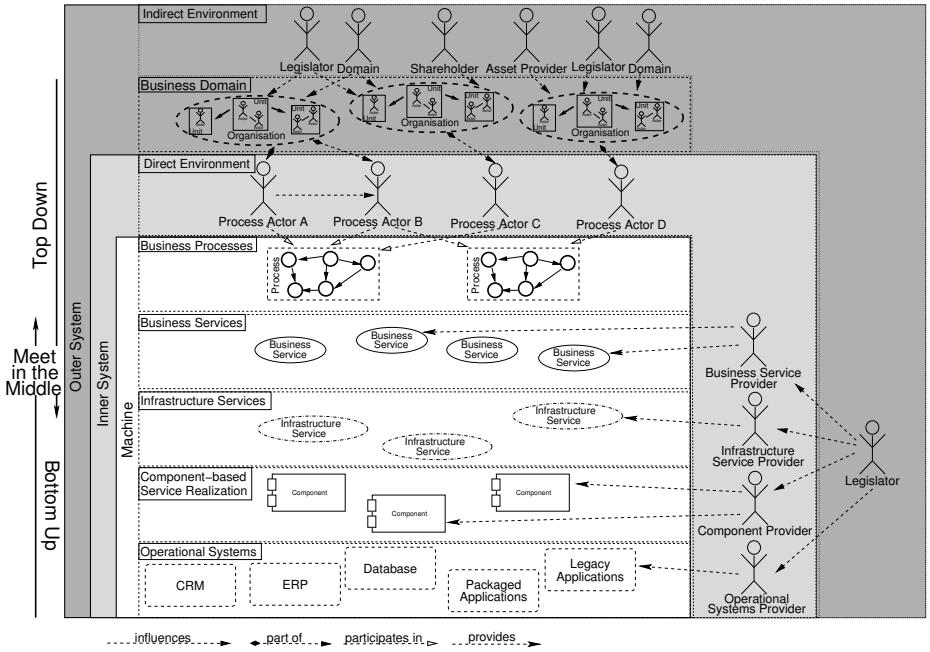

Fig. 3: Stakeholder SOA Pattern

4.2 SOA Stakeholder Pattern

An IT System is not an end in itself. It interacts with its environment and changes the environment. The way it changes the environment is based on the requirements it has to meet. The source for most of these requirements are stakeholders. There are direct stakeholders, who interact with the system, and indirect stakeholders, who do not have a direct relation to the system but to one or more direct stakeholders [**?**]. To know and consider all of these stakeholders is crucial for software engineering.

For a SOA, the knowledge about the environment and the stakeholders is even more a key to success than for other architectures. Conventional applications are often built for a generic use case, which was obtained by generalizing a set of usage scenarios. To use such an application, the environment has to be adapted to a generic use case. Thus, it often happens that organizations are built around their systems. Changing an organization is costly, and processes that are built to meet IT requirements are often inefficient. In contrast, one major aim of SOA is to enable organizations to build IT systems, which follow their business processes. To reach this aim, it is necessary to be able to adapt a certain single scenario and consider its peculiarities. Hence, the scenario for which a SOA is built has to be described in detail.

Stakeholder Extension to the SOA Layer Pattern In Fig. 3, we adapted problem-based methods, such as problem frames by Jackson [**?**], to enrich the SOA layer pattern with its environmental context. According to Jackson's terminology, a system consists of a Machine and its environment. The white area in Fig. 3 spans the SOA layers that form the machine. The business processes describe the behavior of the machine. The business services, infrastructure services, components, and operational systems describe the structure of the machine. Note that the business processes are not part of the machine altogether, as the processes also include actors, which are not part of the machine. Thus, the processes are the bridge between the SOA machine and its environment. The environment is depicted by the gray parts of Fig. 3. Unlike Jackson, who only considers one environment, we distinguish two kinds of environments. The light gray part spans the Direct Environment, which includes all entities, which participate in the business processes or provide a part, like a component, of the machine. An *entity* is something that exits in the environment independently of the machine or other entities. The direct environment reflects Jackson's environment definition [**?**]. The dark gray part in Fig. 3 spans the Indirect Environment. It comprises all entities not related to the machine but to the direct environment. The business domain layer is one bridge between the direct and indirect environment. On the one hand, some entities of the direct environment are part of organizations. On the other hand, some entities of the indirect environment influence one or more organizations. The machine and the direct environment form the Inner System, while the Outer System also includes the indirect environment.

The entities we consider in this paper are stakeholders. Information assets, for example, are put aside. There are two general classes of stakeholders [**?**]. The direct stakeholders are part the direct environment, and the indirect stakeholders are part of the indirect environment. We derived more specific stakeholders from the direct and indirect stakeholders, because these two classes are very generic. Process Actors and different kinds of Providers are part of the direct environment. Legislators, Domains, Shareholders and Asset Providers are part of the indirect environment. In Fig. 3, the resulting stakeholder classes are depicted as stick figures.

Process actors are part of an organization. A process actor can represent an entire organization, a role within this organization, or a specific person. This depends on the usage of the pattern and the level of detail needed when used. A process actor participates in one or more business processes. In some cases a process actor does not only influence other actors through the process, but also influences them directly. For example, one actor can be the supervisor of another actor. This information should be elicited as it can impact the requirements one actor has. In general, process actors are the source of most of the requirements a machine has to fulfill.

In contrast, providers are stakeholders that are not directly involved in the business process. However, they provide a part of one layer within the machine. There are Business Service Providers, Infrastructure Service Providers, Component Providers and Operational Systems Providers. A provider can be a representative

for a whole group of providers, or be a specific one. This also depends on the usage scenario, as for the process actors. The requirements and goals of the providers can be ignored for conventional applications. The reason is that the user of a part a provider offers gains full control of this part in traditional scenarios. For example, a development library once bought is a property of the buyers afterwards. They can change it according to their needs and can be sure that the provider has no influence on the library any longer. This scenario changes for a SOA. Providers are selected at run-time, and in most cases the parts offered by providers remain under their full control. In this scenario, the requirements, goals and other properties of these providers have to be considered. Moreover, the selection of providers can have an impact on other requirements. For example, when a certain information has to remain within one country, all providers from other countries will deny the fulfillment of this requirement. In general, providers introduce only few new requirements, but they have a large influence on already existing requirements.

Legislators represent the jurisdiction of an area. Areas can be very different in size and significance. For example, there are states, like Hessen, there are nations, like Germany, and there are unions like the European Union. The laws of such areas can be interconnected. Then they build a hierarchy with defined scopes. Or they are unrelated, and therefore they can be conflicting. The influence of a legislator is a very strong one. No organization can disobey a law without facing serious consequences. But to be compliant to laws is a complex goal. In general, laws are formulated imprecisely, defining high-level goals. Thus, for every action organizations take, they have to be aware of the relevant laws, and they have to take these laws into account for the problem at hand. For a SOA, one has to know which legislators are of relevance, which laws they enacted related to a SOA, and how to transform these laws into requirements for the SOA.

Domains represent another part of the environment of organizations. The term domain is used in the meaning of business domain. A domain comprises organizations of similar structure, purpose and goals. Domains often introduce specific regulations like standards. In some cases, these regulations are as binding as laws. Other regulations are more of a kind of best practice collections. And as legislators enact laws, which aim at special domains, identifying domains helps to find relevant laws. To consider a domain is important, because describing knowledge and regulations of relevant domains sharpens the view on the organizations.

A Shareholder brings in a certain asset and gets a share of the organization in exchange. In most cases the asset is money. The exchange implies that the assets is owned by the organization afterwards. The share of the organization ensures a specified degree of influence for the shareholder. Shareholders are the primary source of business goals of an organization.

Asset Providers cede a material or immaterial asset to one or more organizations. Unlike shareholders, they remain owners of this asset, and therefore a long-term contractual relation is established. Such a contract implies a certain influence on the organization. Besides a direct relation, there are also cases

where organizations have to avoid the improper use of assets they do not know beforehand. For example,when intellectual properties are stored on a file hosting platform, the filehoster must enable the owners of these intellectual properties to delete the these intellectual properties. Thus, asset providers and their influence on organizations also depend on the relevant legislators.

Knowledge about Stakeholders The pattern shown in Fig. 3 only captures an important, but small, part of the information about the stakeholders. Hence, we provide *templates* for all stakeholders to document the information to be completed. The templates are kinds of textual patterns, which structure information about stakeholders. The structure of these templates applies for all stakeholders of a given kind.

Table 1 and Table 2 show these templates. The first entries are the same for both templates. They capture the name or identifier of the stakeholder at hand, a short description to clarify its properties and position in the environment, and the motivation of interacting in our scenario. The last entry is about the level of abstraction, which is used to describe the stakeholder. Is it a specific stakeholder, who represents a group of homogeneous stakeholders, or a generic stakeholder that is used as placeholder for a group? For the direct stakeholder template (Table 2) there is an additional option for stakeholders, which describe roles within an organization.

For the indirect stakeholder template (Table 1), there are two more general entries. The first entry describes the influence the indirect stakeholder has on the target organization(s) or provider(s). The second entry describes the relation to other indirect stakeholders. Besides these general entries for an indirect stakeholder, Table 1 also shows entries, which are specific for a special kind of indirect stakeholder. For a legislator, there is an entry for the law candidates, which might be of relevance. For a domain, there is an entry about the domain-specific regulations. For shareholders, the description of the shares they own has to be added. And for the asset provider, the assets have to be described.

For the direct stakeholder template (Table 2), there are two additional stakeholder kind-specific entries. The first kind specifies the process and activities the process actors participate in. The second kind specifies the process participants that these stakeholders have a influence on.

5 A Knowledge Elicitation Method for SOA

We have developed a method to instantiate the patterns shown in Figs. 2 and 3 and the associated templates. We divide the method in an information structuring and a stakeholder elicitation phase.

Information Structuring Phase For structuring the information necessary to design a SOA according to the SOA layer pattern (see Fig. 2), we suggest a meet in the middle procedure. The reason is that the business services and infrastructure services layers are intertwined with the business elements. Therefore,

Table 1: Template for Indirect Stakeholder

Name What is the name or identifier of the stakeholder?
Description Which important properties does the stakeholder have? What characterizes the stakeholder? What is its place in the environment?
Motivation Which objectives does the stakeholder follow? Why does the stakeholder influence the organization(s) / provider(s)?
Kind

☐ **Specific** Is the stakeholder a real entity? Is the stakeholder not used to represent a group?

☐ **Representative** Is the stakeholder a real existing entity? Is this stakeholder used as proxy for a group of homogeneous stakeholders?

☐ **Group** Is the stakeholder not a real existing entity? Is this stakeholder used to describe for a group of homogeneous stakeholders?

Influence

On	**Description**	**Severity**
Which organization / provider is influenced?	Which kind of influence? What kind of enforcement? What is the base for the influence?	What is the rating for the severity of the influence?

Relation to other stakeholders

To	**Description**
Which other indirect stakeholder is related to the stakeholder at hand?	Which kind of relation?

optional entries

Law candidates *(Legislator)* Which laws which might be of relevance for the actual SOA to be developed?
Domain-specific regulations *(Domain)* Which domain-specific regulations including, for example, standards and best practice do exist?
Shares *(Shareholder)* Which kind of shares owns the shareholder? How many of them are property of the shareholder?
Assets *(Asset Provider)*

Asset	**Description**	**Provided To**
What are the assets owned by the asset provider?	What are the properties of the asset? How can it be characterized?	To which organization is the asset provided?

we need both information about the technology-related and the business-related layers. Furthermore, our method provides validity checks for the relations between the different layers.

Table 2: Template for Direct Stakeholder

Name What is the name or identifier of the stakeholder?
Description Which important properties does the stakeholder have? What characterizes the stakeholder? What is its place in the environment?
Motivation Which objectives does the stakeholder follow? Why does the stakeholder influence the organization(s) / provider(s)?
Kind
☐ **Specific** Is the stakeholder a real existing entity? Is the stakeholder not used to represent a group?
☐ **Representative** Is the stakeholder a real existing entity? Is this stakeholder used as proxy for a group of homogeneous stakeholders?
☐ **Group** Is the stakeholder not a real existing entity? Is this stakeholder used to describe for a group of homogeneous stakeholders?
☐ **Role** Can this stakeholder be shared through groups of heterogeneous stakeholders? Are there well-defined rights and permissions for this stakeholder?

optional entries

Takes Part In *(Process Actor)*

Process	**Activity**
In which process does the actor participate?	Which activity does the actor enact?

Influence *(Process Actor)*

On	**Description**
Which other actor is influenced by the actor at hand?	Which kind of influence does the actor at hand have to the target actor?

The external input for all steps of this phase (see Fig. 4) is the Unstructured Scenario Description. The SOA Layer Pattern is an additional external input for the first step Describe Organizations. It is instantiated layer by layer in separated steps, based on the Unstructured Scenario Description.

In the step Describe Organizations, we have to collect all relevant organizations. For each such organization we have to collect statements about this organization, describing it further. Next, we have to analyze these statements if they describe business relations between organizations. Last, we have to check for inconsistencies. For example, we have to ensure that no organization is isolated. Being isolated means that there are no business relations to other organizations. In case we find an isolated organization, this organizations is either not of relevance for our scenario, or we are missing important information.

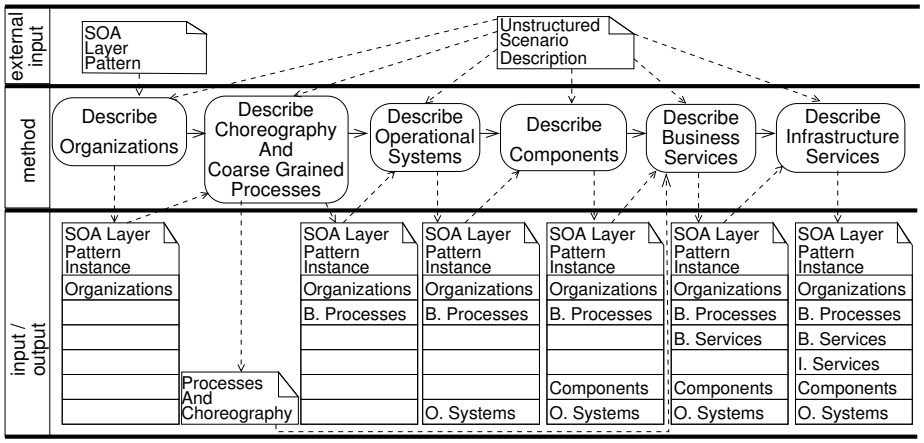

Fig. 4: Description Structuring Phase

In the next step Describe Choreography And Coarse Grained Processes, we have to structure the interaction between organizations. Additionally, we structure the available information about internal processes of organizations. The input for this step is the partly instantiated SOA Layer Pattern. We start with the organizations and their choreography. The choreography describes the interaction between the organizations. We recommend to document the choreography using an appropriate notation. For example, UML and SoaML collaboration diagrams [**?**] can be used. UML and SoaML activity diagrams are of use for more detailed interaction and internal process description. The described processes do not have to be complete, but processes and process steps already mentioned or explained in the scenario description should be captured by such diagrams. Next we have to ensure the coherence of the SOA Layer Pattern instance for finishing the step Describe Choreography And Coarse Grained Processes. The interactions described by the choreography should reflect the business relations found for the organizations. Moreover, the detailed process descriptions must be coherent at the points of transitions between organizations.

For the step Describe Operational Systems, we look for statements mentioning IT systems already used within one of the organizations. The operational systems found have to be analyzed for those, which are to be replaced, and those, which should be wrapped in the new SOA. Only the latter kind of operational system should be added to the SOA Layer Pattern instance.

At least one component should be described in the step Describe Components for each operational system. Whenever for an operational system there is no statement about a component, which should be re-used, the information is missing in the Unstructured Scenario Description, or the operational system is unnecessary. Note that components can exist, which are not part of an operational system, but are already mentioned in the scenario description. But to be sure not to miss any relevant operational system, for each component mentioned

in the Unstructured Scenario Description, we have to check if it exposes such a system.

Up to this point, we have structured the business and the technical parts of the description. Next, we close the gap between those parts. We search for statements, which describe business services for the step Describe Business Services. We add those services to the business service layer. For each business service, we have to check if there is a corresponding process step in the processes described in step Describe Orchestration And Coarse Grained Processes. If such an activity is missing, it should be added. Additionally we have to check if the business service at hand directly exposes a component, which is already part of the SOA Layer Pattern instance. Last we have to check whether the business services at hand is atomic or a composition of other business services. If it is a composition, the business services used to compose it have to be added to the business service layer, too.

The procedure for Describe Infrastructure Services is almost the same. One difference is that infrastructure services are mapped to business services instead of activities within the process, and that they can be orphans. It is not necessary that an infrastructure service is used by an already known business service. The reason is that infrastructure services may provide a more general functionality.

Stakeholder Elicitation Phase In this phase (see Fig. 5), we elicit the stakeholders of our SOA. Therefore we inspect each element of the SOA Layer Pattern instantiated in the previous phase. First, we instantiate the direct system environment (see Fig. 3). We start with the organizations given in the SOA Layer Pattern Instance. For each process related to an organization, we identify the process actors, which act on behalf of the organization in this particular process. There has to be at least one process actor for each organization-process-relation. For all process actors, we have to instantiate the corresponding Direct Stakeholder Templates. Finally, we have to establish the relations between associated process actors.

Next, we inspect each business service, infrastructure service, component and operational system, whether there are already known provider(s) or not. When the providers are already known, we instantiate Direct Stakeholder Templates for them and add them and the corresponding relations to the SOA Stakeholder Pattern instance.

Further, we instantiate the indirect system environment. We also start with the organizations. We analyze for each organization at hand, if there are relevant legislators, domains, shareholders and asset providers. For each identified indirect stakeholder, we instantiate the corresponding Indirect Stakeholder Template, and we add the indirect stakeholder and their relations to SOA Stakeholder Pattern instance. We repeat this procedure for all providers we find in the direct system environment.

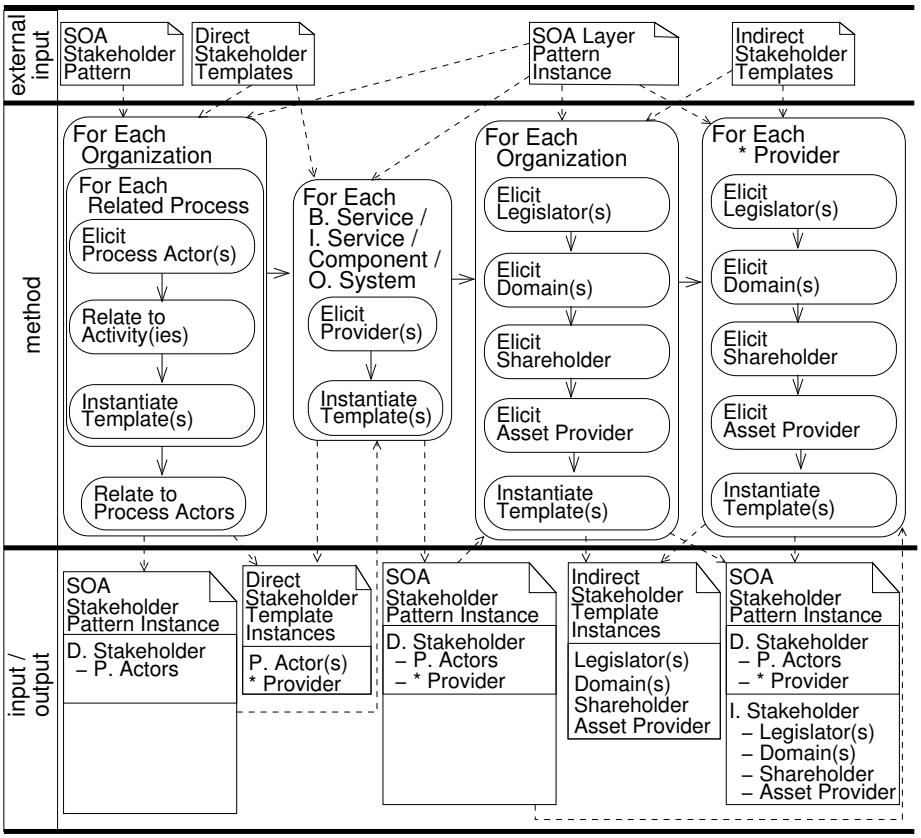

Fig. 5: Stakeholder Elicitation Phase

6 Using the Method

We now apply the presented method to the use case described in Sect. 2. We derive the organizations Customer, Content Aggregator, Content Provider and Bank from the description in Sect. 2. While the Content Aggregator only represents the specific aggregator for whom the SOA has to be developed, the other organizations represent groups. The Content Aggregator is the mediator between Customer and Content Provider. Hence, the business relations reflect this mediation. To accomplish payment, all organizations have business relations to Banks.

Next, we have to structure and describe the choreography and coarse grained processes. Figure 1 already shows a high level choreography. We refine this choreography further and find three major processes: The Content Look up process, in which Customer, Content Aggregator, and Content Provider take part, the Content Delivery process, in which also these three organizations take part, and the Content Payment process, in which all organizations take part. We refine the choreography and these major processes further and end up with an integrated

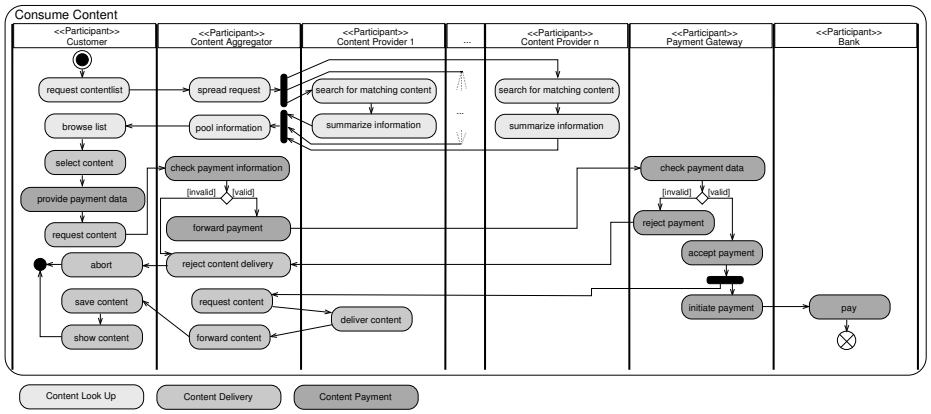

Fig. 6: Business Process

process description as shown in Fig. 6. In this process, we left out the establishment steps as mentioned in Fig. 1. Hence, we focus on the actual business process.

The business process shown in Fig. 6 should enable the customer to consume content offered by different content providers. The content providers want to be paid. The process is simplified at some points. For example, the payment part of the process is much more complex than shown in Fig. 6.

The process starts with a customer who requests a content list. This request contains some search criteria related to the desired content. The content aggregator forwards this request to all currently registered content providers. Each provider responds with a content list considering the search criteria. They also add payment information for the content they offer. The lists are pooled and forwarded to the customer. Note, that for this step the activity diagram is simplified. In a real setting there would be a need to model some sort of timeout and how to deal with it. Next, the customer selects the desired content, provides payment data, and requests the content from the aggregator. The aggregator checks if the payment information is valid. This means that a validity check is executed that decides if the order matches the payment information. If it is not valid, the content delivery is rejected. Otherwise the payment information is forwarded to the payment gateway. The gateway checks if the payment data is valid. The difference between payment data and payment information is that the payment data contains the payment information and the order data. In case the data is invalid the request is rejected and aborted. If the data is valid, the payment is initiated by the payment gateway and fulfilled by the bank. The payment gateway also acknowledges the request, and the content aggregator contacts the relevant content provider. This provider sends the requested content to the aggregator, who forwards it to the customer. The costumer saves and consumes the content.

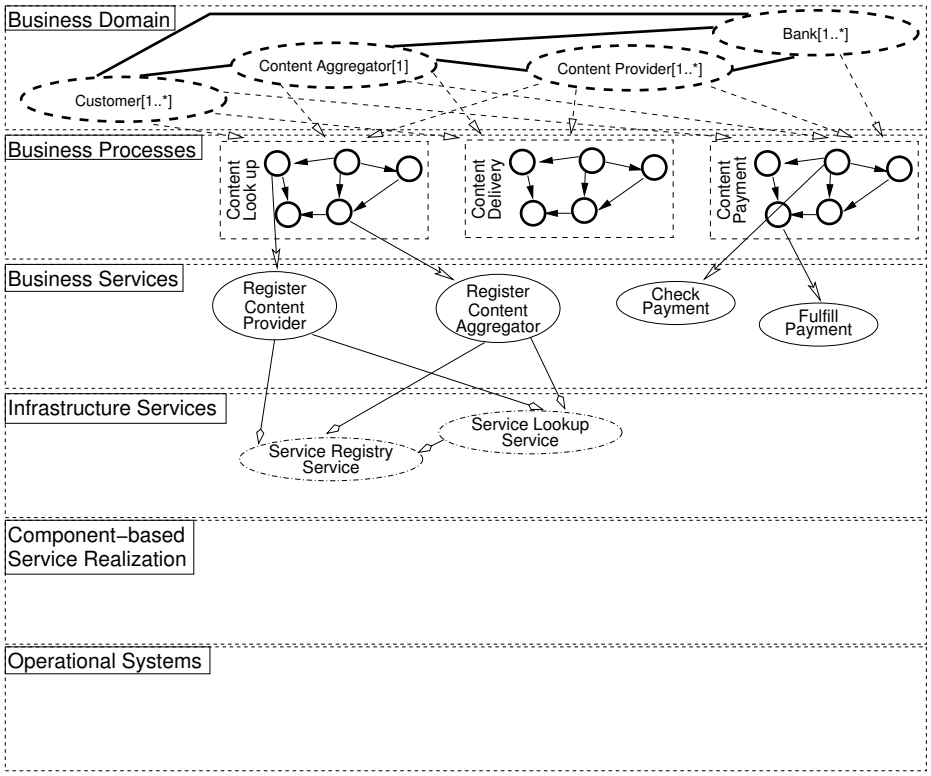

Fig. 7: Instance of SOA Layer Pattern

We skip the description of the operational systems and the components, because our scenario is a new development of a SOA, and no already existing technologies are mentioned in the scenario description.

As mentioned business services, we find Register Content Provider, Register Content Aggregator, Check Payment, and Fulfill Payment. Register Content Provider and Register Content Aggregator rely on the infrastructure services Service Registry Service, and Service Lookup Service. All in all, we obtain the instance of the SOA Layer Pattern shown in Fig. 7.

The result of the stakeholder elicitation phase of our method (Fig. 5) is depicted in Fig. 8. For the Direct Environment, we identified four process actors and two providers. Customer, Content Provider and Bank are stakeholders representing a group. The members of these groups can change dynamically in our scenario, and the members are not that homogeneous that we can refine them further, for example, in roles. For the Content Provider we would be able to do so, but since the processes are to be designed, there is no information we can collect in this early stage. Payment Gateway and Service Broker were already mentioned in the initial unstructured scenario description, but as there is no decision for a specific provider up to this point, they also represent groups.

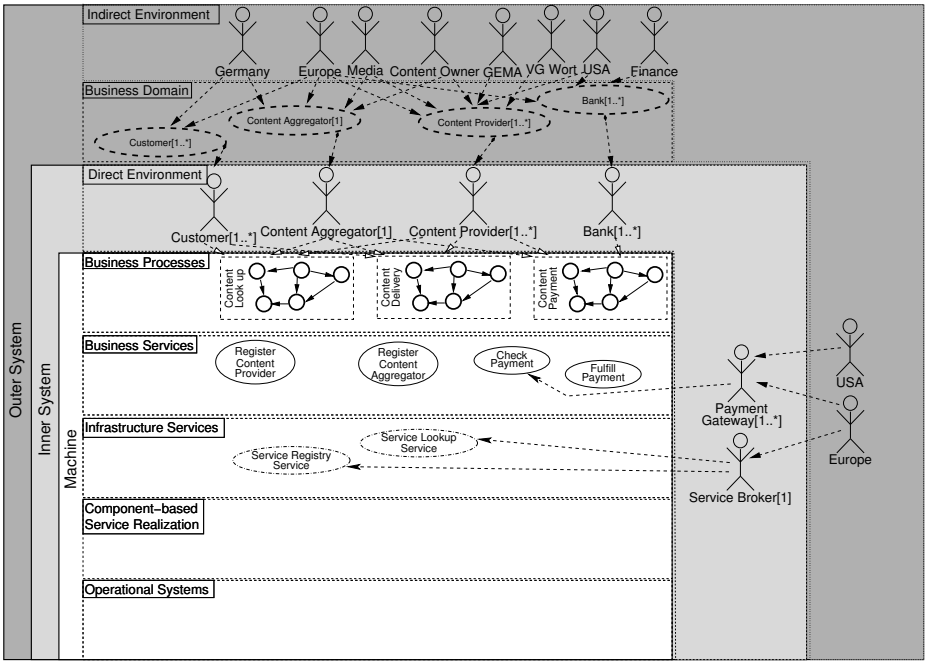

Fig. 8: Instance of the SOA Stakeholder Pattern

For the Indirect Environment we identify Germany, Europe and the USA as legislators, Media and Finance as domains, and GEMA, VG Wort and Content Owner as asset providers.

In our setting, the content aggregator wants to serve the German market, so it is likely that also the customers are from Germany. Thus, the legislator Germany has to be introduced. And as Germany is a part of Europe, Europe has also to be introduced. Since the big media companies reside in the US, we also add USA. To add USA is also necessary for the Payment Gateway, because some important gateways reside in the US. For the Service Broker, we take the decision to aim for a European one.

In Germany, there exist two special kinds of asset providers, besides the Content Owner, which directly plead themselves against content provider and aggregator. GEMA and VG Wort are right distributors, who plead all media owners against media consumers in Germany.

The content aggregator and the content provider are part of the Media domain. Bank is part of the Finance domain.

For reasons of space, we do not show instances of the stakeholder templates (Tables 1 and 2).

7 Using the SOA Pattern in Secure Software Development Life Cycles

The well-known security development lifecycles (SDL) approaches Microsoft SDL [**?**,**?**] and the Comprehensive, Lightweight Application Security Process (CLASP) by the Open Web Application Security Project (OWASP) [**?**] define processes to develop secure software. Gregoire et al. [**?**] compared both methods and derived the following phases: *Education and Awareness*, *Project Inception*, *Analysis*, *Design*, *Implementation, Testing and Verification*, and *Deployment and Support*.

The education and awareness phase of both SDLs can benefit from our pattern-based approach due to a structured security education of SOA. Security experts can use the instantiated pattern and templates to explain the security requirements of the stakeholders, as well as threats and attacks by pointing out their relations to other stakeholders or SOA elements in the pattern.

In the project inception phase of the Microsoft SDL, the pattern can be used to define the roles in the project, which are stakeholders in our pattern. The CLASP approach focuses on security metrics. The instantiated pattern and templates can also be useful for eliciting, e.g., the number of assets in the SOA.

The SOA context description is useful as a starting point for defining security requirements for each stakeholder. The CLASP SDL has always considered security requirements in this phase, while the Microsoft SDL only considers security requirements in a recent update [**?**]. The SOA pattern can help in the elicitation of security requirements. For example, the relation of a stakeholder to a part of the SOA might reveal a security issue, e.g., for the owner of a media content in the SOA an integrity security goal, and a corresponding requirement might be needed.

In the design phase, both approaches demand a specification of the architecture -to -be and also threat modeling. The pattern can be used as a starting point for deriving a SOA architecture. CLASP emphasizes designing a security architecture. This is an extension of the software architecture with security features. These security features can be checked against the security requirements elicited using the SOA pattern. Moreover, CLASP defines an activity for assessing the security of third-party software that shall be integrated into the architecture. The instantiated pattern can support the identification of these third-party components. Microsoft SDL presents a product risk assessment activity, which determines where to focus security efforts. The instantiated SOA pattern can also support this activity for an initial high level risk analysis.

We do not see any use for the pattern in the implementation, testing and verification phase. However, deployment and support can benefit from our pattern. Both approaches require documentation of security, e.g., in the form of manuals. The instantiated patterns and the templates can be used as part of this documentation.

8 Conclusion

We presented *patterns for the context establishment of Service Oriented Architectures*. This approach serves as a proof-of-concept that context descriptions of complex software systems can largely benefit from patterns. Our approach comprises the following main benefits:

- Domain(here: SOA)-specific context establishment based on patterns
- Systematic pattern-based identification of all stakeholders and technologies involved
- Easing the burden of setting up an initial description of a SOA
- The approach has the potential to improve the outcome of service development within a SOA environment

The work presented here will be extended to support requirements elicitation and design description of SOA in a software engineering process. We plan to identify relations between our patterns and specific tasks in typical processes.

In addition, we intend to develop a general method to integrate our SOA patterns as a pre-phase for context establishment into existing security requirements engineering approaches, e.g., KAOS [**?**], Secure-Tropos [**?**] and Security Engineering Process using Patterns (SEPP) [**?**].