A Structured Method for Security Requirements Elicitation concerning the Cloud Computing Domain

Kristian Beckers[1], Maritta Heisel[1], Isabelle Côté[2], Ludger Goeke[2], Selim Güler[3]

[1]paluno - The Ruhr Institute for Software Technology – University of Duisburg-Essen, Germany
{firstname.lastname}@paluno.uni-due.de
[2] ITESYS Institute for Technical Systems GmbH, Dortmund, Germany
{firstname.lastname}@itesys.de
[3] EASY SOFTWARE AG, Mülheim an der Ruhr, Germany
{firstname.lastname}@easy.de

Abstract. *Cloud computing systems offer an attractive alternative to traditional IT-systems, because of economic benefits that arise from the cloud's scalable and flexible IT-resources. The benefits are of particular interest for SME's. The reason is that using Cloud Resources allows an SME to focus on its core business rather than on IT-resources. However, numerous concerns about the security of cloud computing services exist. Potential cloud customers have to be confident that the cloud services they acquire are secure for them to use. Therefore, they have to have a clear set of security requirements covering their security needs. Eliciting these requirements is a difficult task, because of the amount of stakeholders and technical components to consider in a cloud environment.*

Therefore, we propose a structured, pattern-based method supporting eliciting security requirements and selecting security measures. The method guides potential cloud customers to model the application of their business case in a cloud computing context using a pattern-based approach. Thus, a potential cloud customer can instantiate our so-called Cloud System Analysis Pattern. Then, the information of the instantiated pattern can be used to fill-out our textual security requirements patterns and individual defined security requirement patterns, as well. The presented method is tool-supported. Our tool supports the instantiation of the cloud system analysis pattern and automatically transfers the information from the instance to the security requirements patterns. In addition, we have validation conditions that check e.g., if a security requirement refers to at least one element in the cloud. We illustrate our method using an online-banking system as running example.

Keywords - security requirements engineering; security standards; ISO 27001; cloud computing; requirements patterns

I. Introduction

It is hard to find the *right* cloud computing offer with regard to security, when one does not know what *right* is. The definition of precise security requirements provides the means define this right with regard to cloud security. After identifying the cloud security needs, one has to select security measures that fulfill the requirements. We provide a structured method to address this problem, so that SME's can use it with little effort. We base the method on known security guidelines and standards like ISO 27001 to ensure a state-of-the-art approach.

The term *cloud computing* describes a technology as well as a business model (Armbrust et al., 2009). According to the *National Institute of Standards and Technology (NIST)*, cloud computing systems can be defined by the following properties (Mell & Grance, 2011): the cloud customer can require resources of the cloud provider over *broad network access* and *on-demand* and pays only for the used capabilities. Resources, i.e., storage, processing, memory, network bandwidth, and virtual machines, are combined into a so-called *pool*. Using cloud computing services is thus an economic way of acquiring IT-resources. The dynamic acquisition and scalability, yet paying only what was used, makes cloud computing an interesting alternative for a large amount of potential customers. The pay-per-use model includes guarantees such as availability or security for resources via customized *Service Level Agreements (SLA)* (Vaquero et al., 2008). However, the customers are also hesitant to sign up with a cloud provider. In 2009, the International Data Corporation[3] conducted a survey to find out why customers are so hesitant. The survey showed that the lack of trust in cloud security measures is at the top of the list why people avoid using cloud services. The customers fear that managing and storing critical data and executing sensitive IT-processes beyond their grasp has an impact on the security of their data and IT-processes, respectively.

To (re-)gain this trust some well-known cloud providers have started to certify their cloud computing systems according to the ISO 27001 standard to show potential customers that they take their concerns, e.g, considering security, seriously. Unfortunately, it is not always clear to customers what their security requirements actually are.

Our approach aims at helping potential cloud customers to elicit their security requirements. We provide patterns that result in a set of security requirements once they have been instantiated. The patterns are embedded in a method that guides a potential cloud customer through the elicitation process in a structured manner. The method uses an enhanced version of the *Cloud System Analysis Pattern* (CSAP) introduced in (Beckers et al., 2011). It is possible to add security requirements patterns, if the patterns from existing patterns do not suffice. Our method proposes to use the instantiated requirements patterns to select security measures. We integrate published best practices in our method, e.g., from the BSI recommendations (2012). In addition, we recommend to use existing catalogues of security controls in this part of the method, e.g., from the ISO 27001 standard (2005).

We contribute a meta-model that specifies the structure of the CSAP. The meta-model enables us to specify validation conditions to check, e.g., the consistency of the security requirements amongst each other. In addition, we provide tool support for instantiating the cloud system analysis pattern as well as an automatic transfer from the information of the instance to the security requirements patterns.

We use an online-banking system as running example to show the applicability of our approach.

The paper is structured as follows: In Sect. II, we briefly introduce the case study serving as a running example throughout the remainder of this paper. With Sect. III, we portray our approach. Section IV provides an overview of the technical realization of our current tool support. In Sect. V, we evaluate and discuss our approach. We conclude the paper with Sects. VI stating related work and VII summarizing our work and giving directions for future research.

II. Running Example

To illustrate our approach, we consider an online-banking system as running example. The bank institute, as potential cloud customer, wants to expand its business by a structured scenario in form of an online-banking service. To operate economically, the bank is inclined to use a cloud computing service for this task.

The registered business address of the bank institute is within Germany. The bank conducts transactions in Germany, the European Union (EU) and the United States of America (US). Therefore, it must abide

[3]https://www-304.ibm.com/isv/library/pdfs/cloud idc.pdf

by the rules and regulations issued by the affected countries as well as by those of the financial domain. The online-banking service of the bank comprises a premium proposal aimed at VIP bank customers. This proposal is a specific service level agreement between the bank and its VIP customers regarding the availability of the service.

The bank identified the following requirements that should be covered by a cloud computing service:

- **Data storage**: Customer data such as account number, amount, and transaction log history are stored in the cloud.
- **Data processing**: Transactions such as credit transfers are processed in the cloud.
- **Role-based customer handling**: The roles *Bank Customer* as well as *VIP Bank Customer* are handled in the cloud.
- **Compliance**: The cloud provider guarantees that the rules and regulations the bank has to abide by, such as BASEL III, are met.

The internal development unit of the bank institute provides the software for the online-banking service. It is also responsible for building, installing and customizing the components that are necessary to run the online-banking software within the cloud. Examples for such components are the web-server and the application server.

A potential cloud provider should also provide a telephone support, to enable a prompt and straightforward management of problems and open questions.

The potential cloud provider for this bank should only provide cloud services for the finance domain. This lies in the fact that they are tailored to the needs for executing business cases from the financial domain. Furthermore, they take the appropriate security requirements into account. Cloud providers offering a non-domain specific portfolio, implement their services on a broader scope to cover the needs of customers from different domains.

III.Method

This section introduces the method supporting potential cloud customers to specify the cloud services. Furthermore, it supports the users to elicit their security requirements. A graphical overview of our method is depicted in Figures 1, 2, and 3.

A) Modeling the business case by instantiating a Cloud System Analysis Pattern

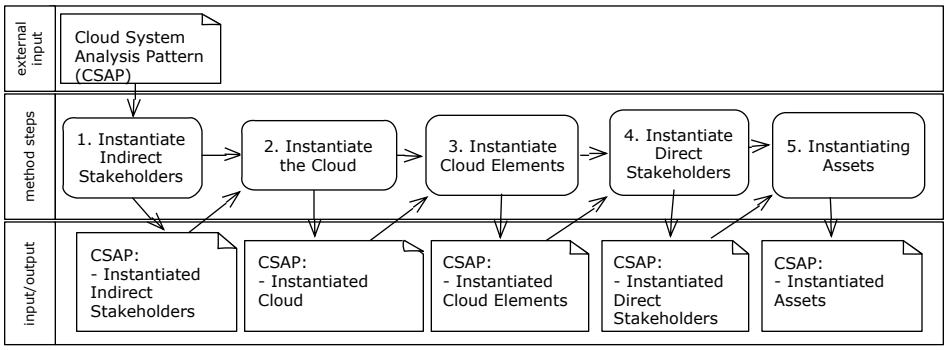

Figure 1. A Method for structured elicitation of Cloud Security Requirements (1/3)

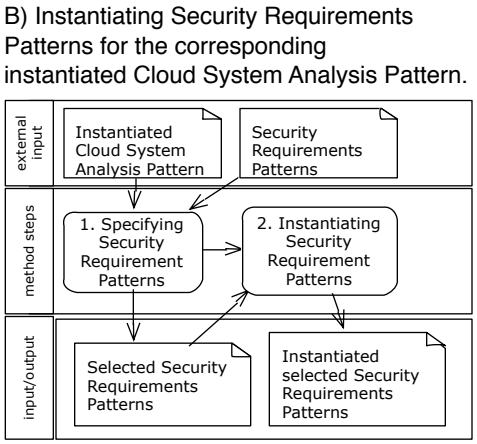

Figure 2. A Method for structured elicitation of Cloud Security Requirements (2/3)

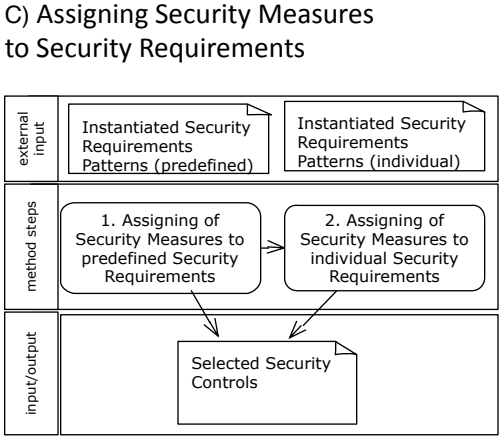

Figure 3. A Method for structured elicitation of Cloud Security Requirements (3/3)

The basis of our approach is the so-called *Cloud System Analysis Pattern* (CSAP) (Beckers et al., 2011). It provides the elements and structure to describe a cloud computing system. Furthermore, it models relations between, e.g., stakeholders and cloud elements. A cloud scenario can be represented by instantiating the different elements in the pattern.

The method is structured into two main steps, namely:

A) Modeling the business case by instantiating a *Cloud System Analysis Pattern* (CSAP).

B) Instantiating security requirement patterns for the corresponding instantiated Cloud System Analysis Pattern.

Each of the above-mentioned steps is further subdivided into sub-steps in order to gradually guide the potential cloud customer through the method. In the following, we describe the different steps in more detail:

A. Modeling the business case by instantiating a Cloud System Analysis Pattern (CSAP)

In this section, we briefly introduce our Cloud System Analysis Pattern. Our approach is based on the pattern introduced in (Beckers et al., 2011). We enrich the aforementioned pattern. We introduce and explain the "enrichment" together with our method. To ease assigning security requirement patterns to the instantiated CSAP in a later step of our method as well as a preparatory step towards tool support, we specify a meta-model. The meta-model is based on the *Unified Modeling Language* (UML) (UML Revision Task Force, 2010). We provide an overview of our CSAP meta-model in Figure 4. During the CSAP instantiation process, we explain the different elements and their meaning for the business cases of potential cloud customers. Furthermore, we describe how we realized the different CSAP elements in our meta-model.

As a first modeling step, the potential cloud customers instantiate a CSAP that represents the required cloud services supporting their relevant business case. The first modeling step is sub-divided into several sub-steps. In the following, we describe these sub-steps in detail:

1. Instantiating Indirect Stakeholders: Indirect stakeholders are contained in the indirect environment. The indirect environment is the root element of the CSAP. It contains the representations of legislations, domain specific formalities and stipulations in form of appropriate instantiations of indirect stakeholders. Furthermore, it contains the direct system environment. The CSAP meta-model specifies the indirect system environment by the class *IndirectEnvironment* (see Figure 4).

   The indirect stakeholders represent

   - legislations of accordant countries,
   - domain specific regulations, and
   - contractual arrangements that affect the business case of the potential cloud customers.

   In the CSAP meta-model, indirect stakeholders are specified by the class *IndirectStakeholder* (see Figure 4). Additionally, the abstract class *Stakeholder* defines the common properties for both indirect as well as direct stakeholders. These common properties are:

   - *name*: The name of direct/indirect stakeholders. Instantiations of direct stakeholders as well as indirect stakeholders are identified by their names.
   - *description*: Information that characterizes direct /indirect stakeholders in natural language.
   - *motivation*: Motivation of direct/indirect stakeholders considering their association with the cloud service.
   - *compliance and privacy*: Relevant compliance and privacy laws as well as regulations that are raised by indirect stakeholders or have to be considered by direct stakeholders.

The type of an indirect stakeholder is represented by the attribute *instanceType*. The different types legislator, domain, and contract are represented as the enumeration *IndirectStakeholderType*.

In our example, we have to instantiate the indirect stakeholders according to the information on regulations given in Sect. II. Therefore, it is necessary to instantiate three indirect stakeholders of instance type *legislator* with the names *Germany, European Union* and *United States,* respectively (see Figure 5). The domain specific regulations are represented by the indirect stakeholder domain instance *Finance*. Furthermore, the bank institute has to consider its contractual arrangement with the VIP bank customers. This contractual arrangement is depicted by the indirect stakeholder instance *VIPCustomization*.

During the instantiation of indirect stakeholders, all of their properties have to be set. For example, the affected regulations have to be allocated to the property *compliance and privacy* of the appropriate indirect stakeholder instances. One regulation for the indirect stakeholder instance *Germany* would be the *Bundesdatenschutzgesetz*.

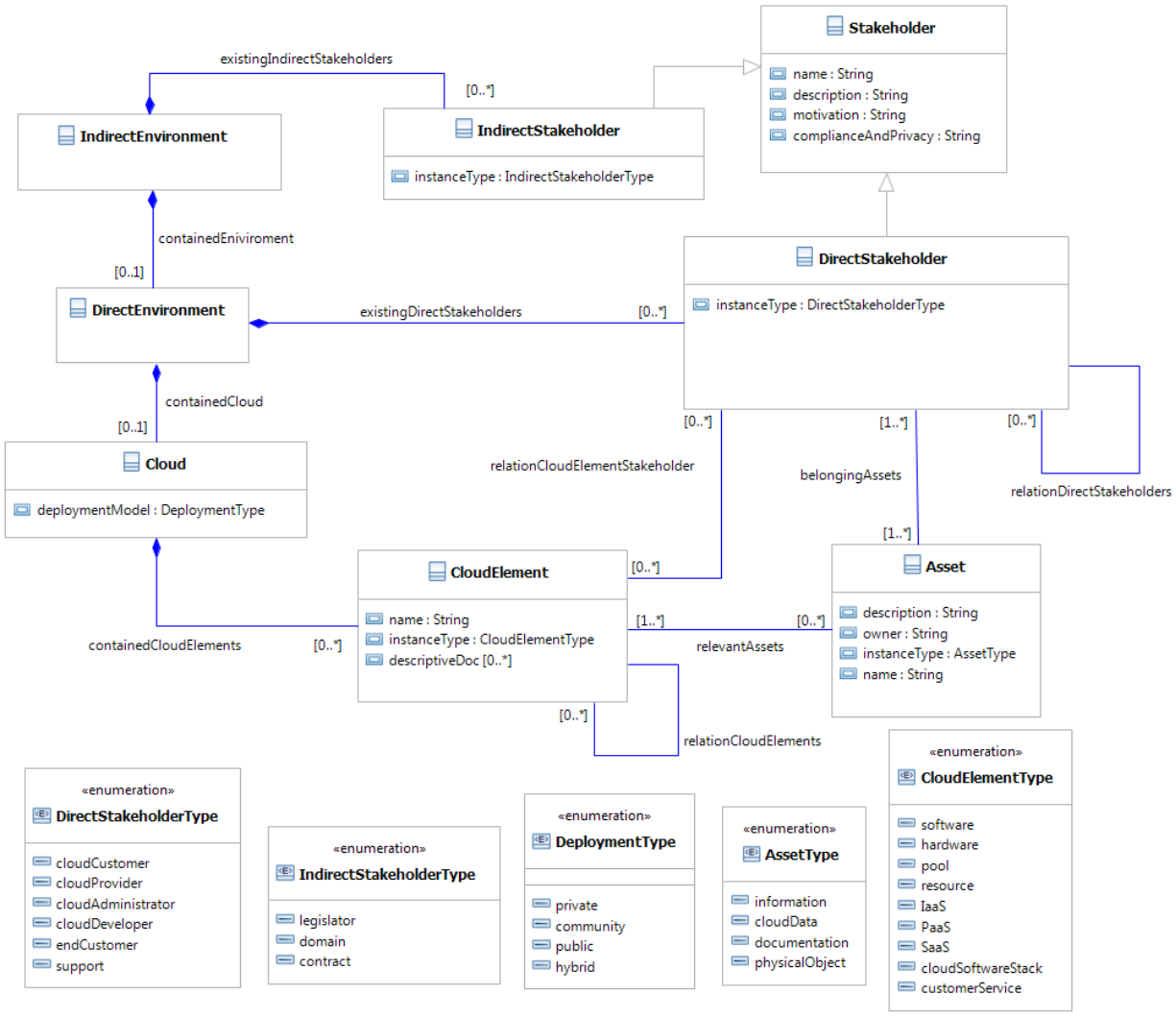

Figure 4. Meta-model of the Cloud System Analysis Pattern

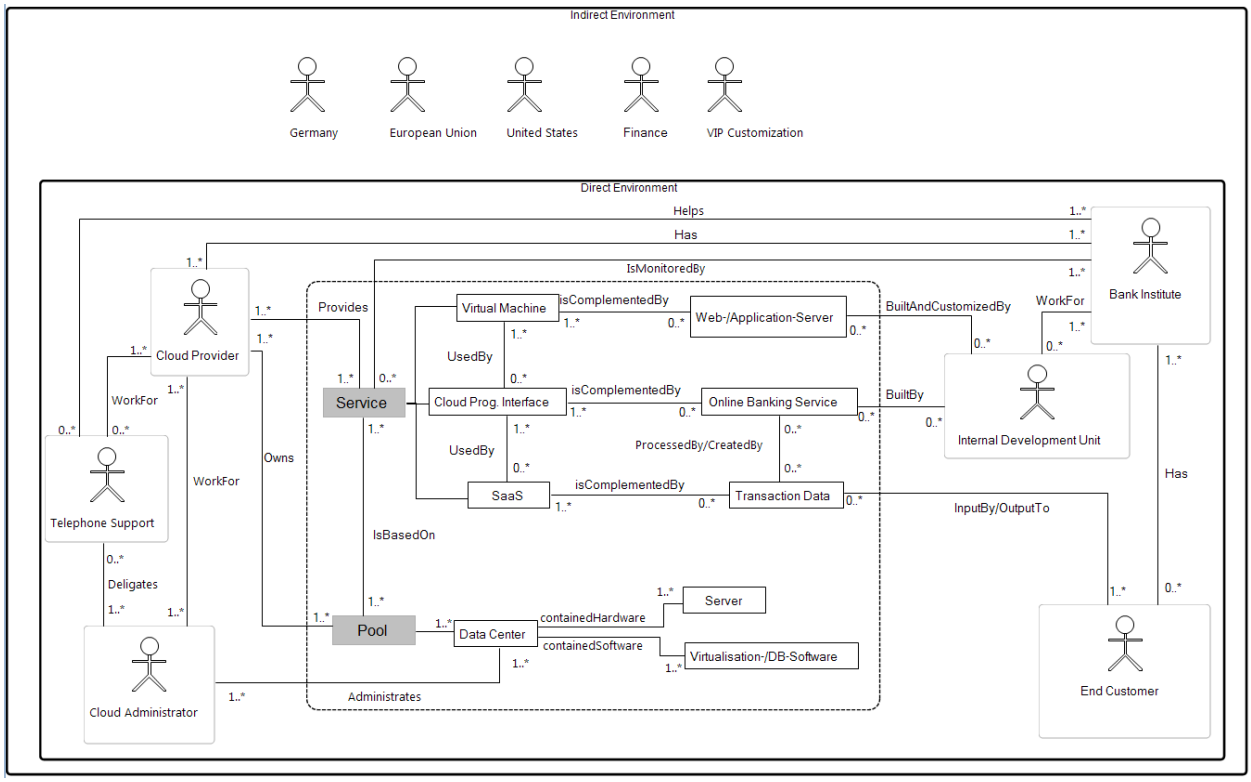

Figure 5. Instantiated Cloud System Analysis Pattern

2. Instantiating the Cloud: The second sub-step consists of instantiating the cloud. The direct environment contains the representations of the cloud and the direct stakeholders of the cloud. Relations between elements form the direct environment and indirect stakeholders contained in the indirect environment are not allowed, e.g., a legislator is usually not directly interacting with a cloud system. In the CSAP meta-model, the direct environment is specified by the class *DirectEnvironment*.

The cloud itself consists of different types of cloud elements (see sub-step *Instantiating Cloud Elements*). Furthermore, assets are contained in the cloud (see sub-step *Instantiating Assets*). In the CSAP meta-model, the cloud is specified by the class *Cloud*. The deployment model of the cloud is represented by the property *deploymentModel*. The enumeration *DeploymentType* defines private, community, public, and hybrid as values for this type. The different deployment models are explained in (Mell & Grance, 2011).

3. Instantiating Cloud Elements: Cloud elements represent the physical cloud resources and the cloud services that provide these cloud resources to the cloud customers. The resources of cloud customers that are executed in the cloud are also represented by cloud elements. In the CSAP meta-model, cloud elements are specified by the class *CloudElement* (see Figure 4). The property *instanceType* represents the type of a cloud element. The different types are defined by the enumeration *CloudElementType.*

   A cloud element can refer to additional documentation, e.g. manufacturer's documentation that provides more information about the accordant cloud element. This reference is specified by the property *descriptiveDoc*. For the representation of the different models of cloud services *CloudElementType* defines the following literals:

- *IaaS*: Infrastructure as a Service.
- *PaaS*: Platform as a Service.
- *SaaS*: Software as a Service.

A detailed consideration of the above-mentioned service models is given in (Mell & Grance, 2011)

All cloud services are contained in a container named *Service* (see Figure 5). It allows the association of all contained cloud processes at once. The service container is represented by a cloud element with the instance type *service*. It cannot be instantiated.

The bank institute, in our running example, requires an IaaS cloud service in form of a virtual machine for running components such as a web- and application servers. Based on that, a cloud element with the instance type *IaaS* has to be instantiated. In our example, this cloud element instance has the name *Virtual Machine* (see Figure 5).

For processing the online-banking service, a cloud programming interface enabling the communication with the appropriate cloud resources is required. Because this cloud programming interface represents a PaaS cloud service, a cloud element with the instance type *PaaS* and the name *Cloud Progam*. *Interface* has to be instantiated.

The bank institute requires no SaaS cloud service as they implement their own online-banking service. Based on that, no cloud element of the instance type *SaaS* has to be instantiated. Therefore, the corresponding CSAP element remains unchanged (see Figure 5).

Generally, the fact that CSAP elements were not instantiated indicates that the needed information for the instantiation is currently unavailable or not relevant for the scenario at hand. Elements, which have not been instantiated remain unchanged.

Cloud resources represent the required hardware and software supplied by cloud providers. These resources are provided via cloud services. The modeling of the cloud resources enables statements about the security of a cloud service. Hardware and software are represented by cloud elements with the instance types *hardware* and *software*, respectively. A resource is depicted by a cloud element with the instance type *resource*. All cloud resources are contained in a pool. Pool means that *"resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand"*(Mell & Grance, 2011). It allows associating all cloud resources at once. The pool is represented by a cloud element with the instance type *pool*. It cannot be instantiated.

In our example, the hardware – a potential cloud provider has to own – is represented by the cloud element instance *Server* with the instance type *hardware*. The software is depicted by the cloud element instance *Virtualisation-/DB-Software* of type *software*. The instantiated cloud element *Data Center* of type *resource* specifies that the cloud resource shall reside in a data center.

The instantiation of the cloud software stack is necessary if the potential cloud customers require an IaaS cloud process. In this case, the potential cloud customers want to execute their own software such as web servers and application servers by using an IaaS cloud service.

In our example, the bank institute needs to execute its own software such as a web server and an application server for processing their online-banking service using an IaaS in form of a virtual machine. Therefore, the cloud software stack has to be instantiated. It is represented by the instantiated cloud element *Web-/Application-Server* with the instance type *cloudSoftwareStack*.

The customer service has to be instantiated, if the potential cloud customers require a PaaS cloud service for executing their own service.

In the context of our example, the customer service has to be instantiated, because the bank institute executes their online-banking process by the use of a PaaS in form of the cloud programming interface.

Cloud elements can have relations to each other. In the CSAP meta-model these relations are specified by the association *relationCloudElements*. Our example contains the relation *isBasedOn* (see Figure 5). This relation specifies that the different cloud services are based on the pool of cloud resources.

4. Instantiating Direct Stakeholders: Direct stakeholders are persons, a group of persons or organizations that have a direct association to the cloud. In the CSAP meta-model, direct stakeholders are specified by the class *DirectStakeholder* (see Figure 4). The different types of direct stakeholders are depicted by the property *instanceType*. The enumeration *DirectStakeholderType* defines the following types:

   - *cloudCustomer*: Cloud customers use cloud resources via the appropriated cloud services.
   - *cloudDeveloper*: Cloud developers work for cloud customers. Based on the models of cloud services that cloud customers require, the cloud developers are accountable for the components of the cloud software stacks and/or the cloud customers services.
   - *support*: The support employees work for cloud providers. They are the contact persons for cloud customers and end customers and delegate open questions and problem reports to the cloud administrators.
   - *endCustomer*: End customers are customers that use a cloud service. They don't provide services to other customers. An end customer can use a cloud service directly from a cloud provider or indirectly over another cloud customer.
   - *cloudProvider*: Cloud providers own a pool of cloud resources. They provide the usage of these cloud resources over accordant cloud processes.
   - *cloudAdministrator*: Cloud administrators are responsible for the administration of the cloud resources. They work for cloud customers.

   In our example, the bank institute has to be represented in the CSAP. For this, a direct stakeholder with the instance type *cloudCustomer* and the name *Bank Institute* has to be instantiated. The property description depicts informal characteristics of the bank institute, like the fact that the bank institute represents a legal person operating in the financial sector. The property motivation depicts that cost savings are the reasons for using cloud services (see Sect. II).

   Because the bank institute uses its own online-banking service and cloud software stack components, a direct stakeholder with the instance type *cloudDeveloper* has to be instantiated. In our example, this direct stakeholder instance has the name *Internal Development Unit*.

   End customers represent the customers of the bank institute. As the institute is responsible for selecting the cloud provider, the end customer is not relevant and does not need to be instantiated.

   The cloud provider, represented by an indirect stakeholder with the instance type *cloudProvider*, can't be instantiated at this point. The instantiation of the cloud administrator will delayed until the cloud provider is known.

   Because the bank institute requires a telephone support (see Sect. II), the direct stakeholder with the instance type *support* has to be instantiated. In our example, this direct stakeholder instance has the name *Telephone Support*.

Direct stakeholders can have relations to each other (see Figure 4). These relations are specified in the CSAP meta-model by the association *relationDirectStakeholders*. An example for such an association is the relation *WorkFor* between the bank institute and the internal development unit (see Figure 5).

Direct stakeholders can also have relations to cloud elements. In the CSAP meta-model, the association *relationCloudElementStakeholder* defines these relations. The relation *BuiltBy* in our example represents that the internal development unit builds the customer service (see Figure 5).

5. Instantiating Assets: Assets represent anything that has a value to potential cloud customers (International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2005). Assets can be, for example, different occurrences of information or physical objects.

   In the CSAP meta-model, assets are defined by the class *Asset*. The different types are defined by the enumeration *AssetType* with the following values: information, cloud data, documentation, and physical object. Additionally, the class *Asset* defines the following properties:

   - *owner*: Person or group of people who is in charge for the asset.
   - *description*: Characterizes the asset in natural language.

   In our example, the data that is relevant in the context of an online-banking transaction represents an asset for the bank customer. Indirectly, this transaction data depicts also an asset for the bank institute, because a loss of, e.g., integrity regarding the transaction data would have serious consequences for them. The transaction data has to be represented by an instantiated asset of the instance type *cloudData*. This asset instance has the name *Transaction Data*.

   Assets can have relations to the representations of cloud customers and end customers who own the assets. These relations are specified by the association *belongingAssets*. In our example, the relation *InputBy/OutputTo* depicts, that the transaction data is an asset for the bank customer.

   Furthermore, assets can have relations with the cloud elements that process, produce and/or store assets. In the CSAP meta-model, the association *relevantAssets* defines these relations. For example, the relation *ProcessedBy/CreatedBy* represents that transaction data is processed and produced by the online-banking service.

B. Instantiating security requirement patterns for the corresponding instantiated Cloud System Analysis pattern

According to (Fabian et al., 2010), a security requirement is typically a confidentiality, integrity or availability requirement. In our method, these kinds of requirements concern the different elements in a CSAP instance.

The idea of requirement patterns is to provide guidance on how to specify common types of requirements, to make it quicker and easier to write them, and to improve the quality of those requirements (Withall, 2007).

In this step, we describe our security requirement patterns and how to instantiate them. The resulting security requirements are related to elements in the CSAP instance.

A security requirement pattern contains always *fixed text passages* that represent the meaning of the security requirement pattern. These fixed text passages have generic text passages embedded in them that have the following structure:

- *[]*: Opening and closing squared brackets mark the beginning and end of a generic text passage, respectively.
- *instance type* of CSAP element: In this case, a generic text passage references certain elements in the corresponding CSAP. They consider all elements whose instance type correspond to the keyword in the generic text passage. For example, the generic text passage in the security requirement pattern in Example 1references all cloud elements of the instance type *cloudSoftware*. During the instantiation of a security requirement pattern, the potential cloud customer can select the elements for which the surrounded fixed text applies. The appropriate elements are identified by the values of their name and *instanceType* properties.

  In our example, the cloud software *Virtualization-/DBSoftware* has to be inserted, because virtualization and data base software have to be protected against malicious software.

  The keyword *all* before the instance type specifies that **all** appropriate referenced CSAP elements are relevant for the particular security requirement pattern. Here, potential cloud customers have to insert all referenced CSAP elements into the corresponding security requirement pattern.

  During instantiating the security requirement pattern in Example 2the representation of the cloud data *Transaction Data* has to be inserted into the generic text passage. The resulting security requirement is given in Example 3.

  Until now, keywords that reference CSAP elements apply only to CSAP elements of one instance type, respectively. There are also keywords that allow referencing several instances of CSAP elements with different instance types. These keywords are considered in the following:

- *all cloud elements* references **all** instantiated cloud elements **without** considering the instance type. In our example, the representations of the cloud elements *Virtual Machine*, *Cloud Prog. Interface*, *Web-/Application-Server*, *Online Banking Service*, *Data Center*, *Server* and *Virtualisation-/DB-Software* are inserted because they are instantiated. Since the cloud services *SaaS* is not instantiated. Therefore, it is not considered in the security requirement pattern.
- *all cloud services* references **all** instantiated cloud elements with the instance types *IaaS*, *PaaS* and *SaaS*.

  Text in bold-face is used to highlight the aspects treated in the security requirement pattern.

Protection against malicious software (viruses, Trojan Horses,...) shall be implemented in **software** [cloud software].

Example 1. Example for a security requirement pattern that references information in an CSPA instance

The loss of **cloud user data** [all cloud data] shall be prevented.

Example 2. Security requirement pattern referencing **all** instantiated cloud elements of the instance type *cloudData*

Protection against malicious software (viruses, Trojan Horses,...) shall be implemented in **software** Virtualization-/DB-Software .

Example 3. Example for an instantiated security requirement pattern

Instantiating security requirement patterns consists of specifying security requirement patterns and instantiating requirement patterns.

These two steps are explained in detail in the following:

1. Specifying security requirement patterns: In this step, the potential cloud customers have to specify their requirements for the security of a potential cloud service. For that purpose, a set of predefined security requirement patterns covering common security issues is provided. This set should serve as a starting point for the specification of security requirement patterns and has no claim for completeness. It is possible to add new and to update existing security requirement patterns whenever new or changed threats or security mechanisms arise. We are confident that the effort needed to create such new or updated security requirement patterns can be reduced by applying the previously mentioned syntactic rules, e.g., [cloud customer]. The predefined security requirement patterns are based on the works found in (Cloud Security Alliance, 2009), (Cloud Security Alliance, 2010), (Heiser & Nicolett, 2008), and (European Network and Information Security Agency, 2008). The potential cloud customers can evaluate the predefined security requirement patterns and decide which security requirement patterns are relevant to them. With this, they have the possibility to customize the predefined security requirement patterns to their needs.

   In our example, the bank institute specifies one aspect regarding the privacy of customer data by adopting the security requirement pattern given in Example 4. Another privacy aspect for customer data is specified by customizing the security requirement pattern given in Example 5.

This customization is necessary, because the bank institute does not allow the collection of personal data by third parties.

If potential cloud customers have further security requirements that are not included in the set, they can extend the list by creating new security requirement patterns. The structure of customized and newly created security requirement patterns has to be validated. The resulting set contains all security requirement patterns that have to be instantiated. This instantiation is considered in detail in the following step.

2. Instantiating security requirement patterns: After specifying security requirement patterns the potential cloud customers have to instantiate the relevant security requirement patterns consecutively. Therefore, they choose an affected security requirement pattern and assign the information from the corresponding CSAP to the generic text passages. After that, we add this security requirement to the other already elicited security requirements.

   During the instantiation of the security requirement pattern in Example 4, the bank institute has the opportunity to select the instantiated cloud customer *Bank Institute* and the not instantiated end customer *End Customer*. They have to select both elements, because the bank institute's personal data as well as the bank customers' personal data have to be confidential. Accordingly, both element names are inserted into the generic text passage. Thus, the name *End Customer* refers to all online-banking customers in general and not to one specific online-banking customer in particular.

Confidentiality of **personal data** of [cloud customer,end customer] shall be achieved.

Example 4. Adapted security requirement pattern

Personal data of [cloud customer, end customer] shall not be collected without permission.

Example 5. Customized security requirement pattern

C. Assigning security measures to security requirements

In this step, we describe assigning security measures to the requirements specified in paragraph B. These security measures define what the providers of clouds have to implement to fulfill the specified security requirements. They enable potential cloud customers the assessment, whether their requirements are fulfilled by cloud providers. The result doesn't raise the claim to represent the exact level of security of clouds. Furthermore the result shall be used by potential cloud customers as a basis for their discussion with cloud providers.

Our method considers security measures in form of

- ISO 27001 controls in Annex A (ISO27001) and
- Security recommendations from the white paper "Security Recommendations for Cloud Computing Providers" (BSI, 2012) from the German *Federal Office for Information Security* (germ. Bundesamt für Sicherheit in der Informationstechnik (BSI))

Several well-known companies have adopted this approach such as Microsoft[4,5] Amazon[6], Google[7,8], and Salesforce[9]. The aim of the ISO 27001 standard is to establish an Information Security Management System (ISMS). To use this standard for cloud computing systems is in accordance with the German Federal Office for Information Security (BSI)[10].

The aforementioned BSI-whitepaper contains recommendations for safe cloud computing on an abstract level. These BSI-recommendations can be a guideline for cloud providers regarding the implementation of security measures. However, the realization of the BSI-recommendations **doesn't** replace the implementation of an information security management system regarding a relevant norm like ISO 27001 or BSI 100-2 for the achievement of an adequate level of security. The recommendations rather allow potential cloud customers to clarify the needed protection for their own data and application. They can use the content of the whitepaper as a basis for the discussion with cloud providers (cf. BSI, 2012).

The determination of the mapping between our security requirements and ISO27001 controls and BSI-recommendations, respectively, was conducted in two steps. In the first step, we screened the ISO 27001 Annex A and the BSI white paper. To support the assignment of the individual ISO 27001 controls and BSI-recommendations, respectively, we determined the particular assets, security goals and expressive keywords regarding our predefined security requirement patterns before the screening. With the help of this additional information the set of possibly relevant predefined security requirement patterns regarding an ISO 27001 control or BSI recommendation could be narrowed down. This resulting set of security requirement patterns was then considered in detail. In the second step, we iterated over our predefined security requirement patterns and executed full text searches on basis of the according additional information in the ISO 27001 Annex A and the BSI white paper, respectively.

As an example of our mapping to security measures we consider the predefined security requirement pattern given Example 2. It is mapped, among others, to the

- ISO 27001 control A.10.5.1: "Back-up copies of information and software shall be taken and tested regularly in accordance with the agreed backup policy" (International Organization for Standardization, 2005) and
- third BSI recommendation for data security: "Regular data backups, with customers being able to audit their basic parameters (scope, save intervals,save times and storage duration)" (BSI, 2012).

---

[4] http://blogs.msdn.com/b/windowsazure/ archive/2011/12/19/windows- azure- achieves- is0- 27001- certification- from- the- british-standards- institute.aspx

[5] http://www.windowsazure.com/en- us/support/ trust- center/compliance/

[6] http://aws.amazon.com/security/

[7] http://googleenterprise.blogspot.com.br/ 2012/05/google- apps- receives- iso- 27001.html

[8] http://www.computerweekly.com/news/ 2240150882/Google- Apps- for- Business- wins- ISO- 27001- certification

[9] http://www.salesforce.com/platform/cloud- infrastructure/security.jsp

[10] https://www.bsi.bund.de/ SharedDocs/Downloads/EN/BSI/ Publications/Minimum_information/ SecurityRecommendationsCloudComputingProviders. pdf

When an ISO 27001 control and a BSI-recommendation respectively couldn't be assigned to a predefined security requirement pattern, we defined a new security requirement pattern regarding the concerned ISO 27001 control and BSI recommendation respectively.

In the following we consider assigning security measures to

- predefined security requirements and
- individual security requirements.

C1. Assigning security measures to predefined security requirements

Assigning security measures to security requirements in such a way is relevant, when potential cloud customers have created security requirements **only** by instantiating security patterns of our predefined set (see Figure7). In this case, they can imply the relevant ISO 27001 controls and/or BSI-recommendations a cloud provider has to implement, directly form our mapping.

C2. Assigning security measures to individual security requirements

Individual security requirements are instantiated from security requirement patterns that

- are based on predefined security requirement patterns that have been modified by potential cloud customers or
- have been specified completely by the potential cloud customers.

Regarding the first bullet point potential cloud customers have to consider the security measures, which are mapped to the original predefined security pattern. Here, they have to decide for each security pattern, if it still matches to the individual security requirement.

Considering the second bullet point, potential cloud customers have to execute the mapping to security measures by themselves.

IV.Implementation

In this section, we present our current tool support.

First, we consider the *Cloud System Analysis Pattern Tool* (CSAP Tool). This tool supports the instantiation of the CSAP. It also provides modeling support that allows to extend the CSAP with additional instantiable CSAP elements and the corresponding relations between them and other CSAP elements. Because this procedure is not part of our method it will not be considered further. The architecture of our tool is shown in Figure 6.

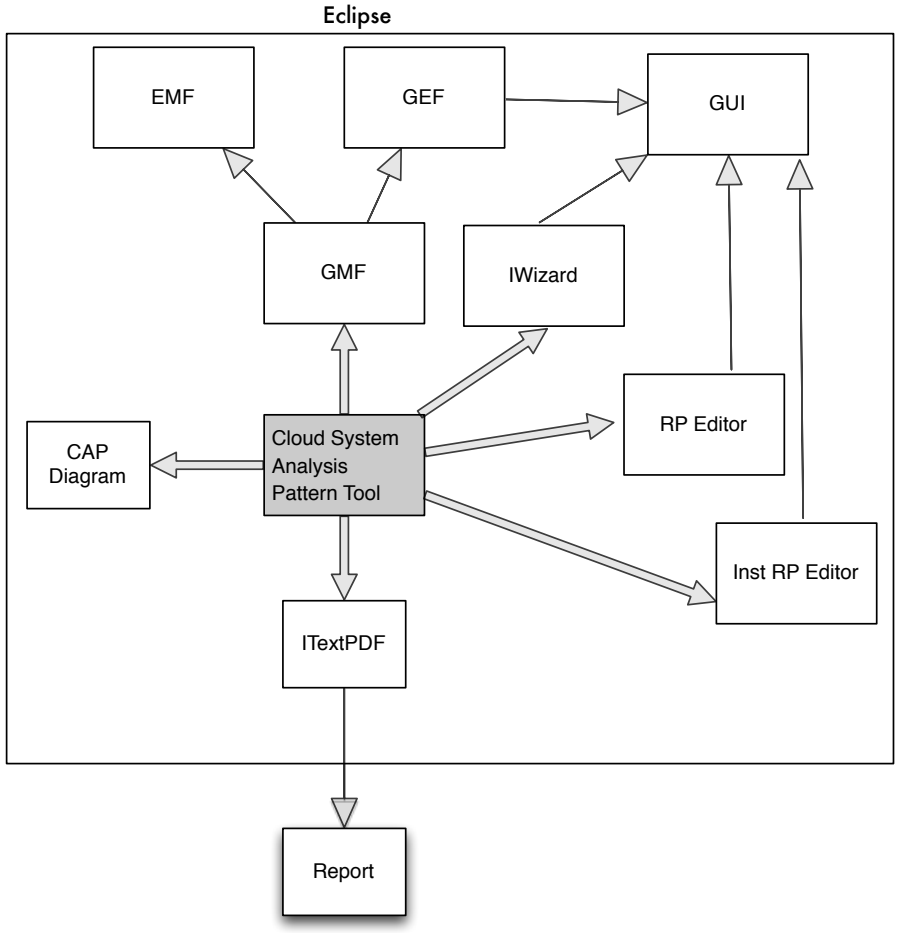

Figure 6. Architecture of the Cloud System Analysis Pattern Tool

The notation used to specify the pattern is based on UML, i.e., stickmen represent roles, boxes represent concepts or entities of the real world and named lines represent relations (associations) equipped with cardinalities.

Our tool is based on the Eclipse platform (Eclipse Foundation, 2011) as well as its plug-ins Eclipse Modeling Framework (EMF) (Eclipse Foundation, 2012) and the Graphical Editing Framework (GEF) (Eclipse Foundation, 2012). We further use the Graphical Modeling Framework (GMF) (Eclipse Foundation, 2011) to generate graphical editors.

Our CSAP meta-model (see Sect. III-A) serves as the basis for generating the appropriate (ecore) model using EMF. This (ecore) model enables the generation of components representing CSAP information within the CSAP Tool. The cloud system analysis pattern GUI is generated by GMF using our CSAP meta-model.

The CSAP Tool uses the eclipse interface *IWizard* to create a wizard to support the instantiation of a cloud pattern. The wizard provides a graphical interface that asks the user for the necessary information to instantiate stakeholders, cloud elements and assets. It asks, for example, for the name and owner of an asset. In addition, the wizard supports instantiating several instances of one instantiable CSAP element. For example, the wizard can instantiate four indirect stakeholders in form of legislators at once.

Furthermore, we equipped the wizard with validation capabilities. An example for an already implemented validation condition is to check whether all fields of a stakeholder in the corresponding template have entries.

It is also possible to generate a report, called CSAP *report*. It contains the graphical representation of the model as well as the texts provided in the stakeholder template. We use the *iTextPDF* interface of Eclipse to generate the pdf-files for the report.

For the management of security requirement patterns, we provide a *Requirement Pattern Editor* (RP Editor). Its implementation is also based on the Eclipse platform (Eclipse Foundation, 2011) as well as the aforementioned plug-ins. The RP Editor provides functionality for displaying, creating, modifying and deleting security requirement patterns. Figure 7shows the modification of a predefined security requirement pattern. For creating and modifying security requirement patterns, the RP Editor provides keywords for referencing CSAP elements. These keywords represent the instance types of the corresponding CSAP elements.

Considering a newly created or modified security requirement pattern, it has to be ensured that its structure is valid. Based on that, the RP Editor provides an appropriate validation function that is executed before adding a newly created or modified security requirement pattern.

The management of security requirements is provided by another editor, the so-called *Instantiated Requirement Pattern Editor* (InstRP Editor). The functionality of the InstRP Editor comprises instantiating security requirement patterns as well as displaying, modifying, and deleting security requirements.

The implementation of the InstRP Editor is based on the same technologies as the RP Editor. During the instantiation of security requirement patterns, the names of the referenced CSAP elements are inserted into the corresponding generic text passages, automatically. For this procedure the InstRP Editor provides an appropriate wizard, which allows the selection of the relevant CSAP element names.

The representation of predefined security requirement patterns in the InstRP Editor is shown in Figure 8. Potential cloud customers have the possibility to select the relevant predefined security requirement patterns that shall be instantiated. In Figure 8, for example, only the security requirement pattern from Example 1 is selected.

Figure 9shows the instantiation of the above selected security requirement pattern in the InstRP Editor. Here, the InstRP Editor shows the security requirement pattern that shall be instantiated and the current and only keyword *cloud software* that shall be substituted by the appropriate information of the corresponding CSAP instance. In our instantiation, the CSAP instance from Figure 5 is considered. According to this, the only selectable CSAP information is *Virtualization-/DB-Software*. The resulting security requirement is given in Figure 10.

In Sect. III-B, we mentioned that the set of security requirements has to be consistent to the corresponding CSAP instance. This consistency has to be ensured by an appropriate validation function. To enable this the instance type of a referenced CSAP element has to be captured in the representation of a security requirement. Based on the name and the instance type, the validation function can compare the references in the security requirements against the elements in the corresponding CSAP instance.

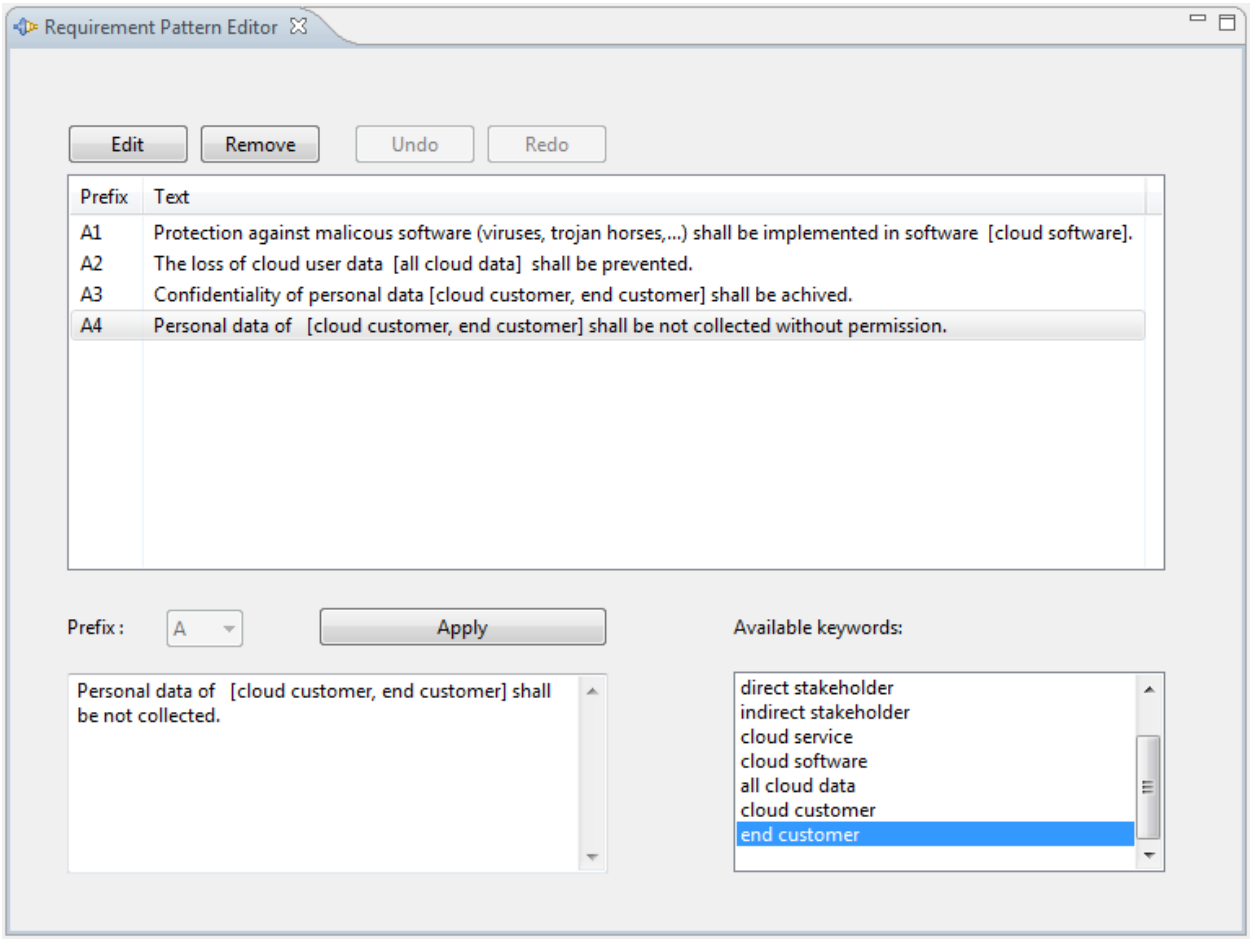

Figure 7. Modification of a security requirement pattern in the RP Editor

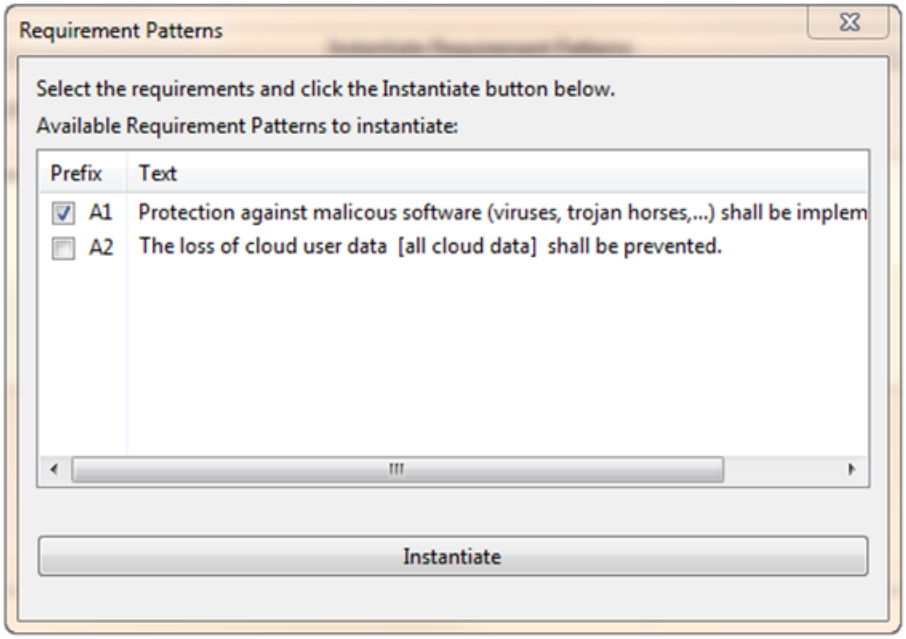

Figure 8. Selection of predefined security requirement patterns in InstRP Editor for instantiation

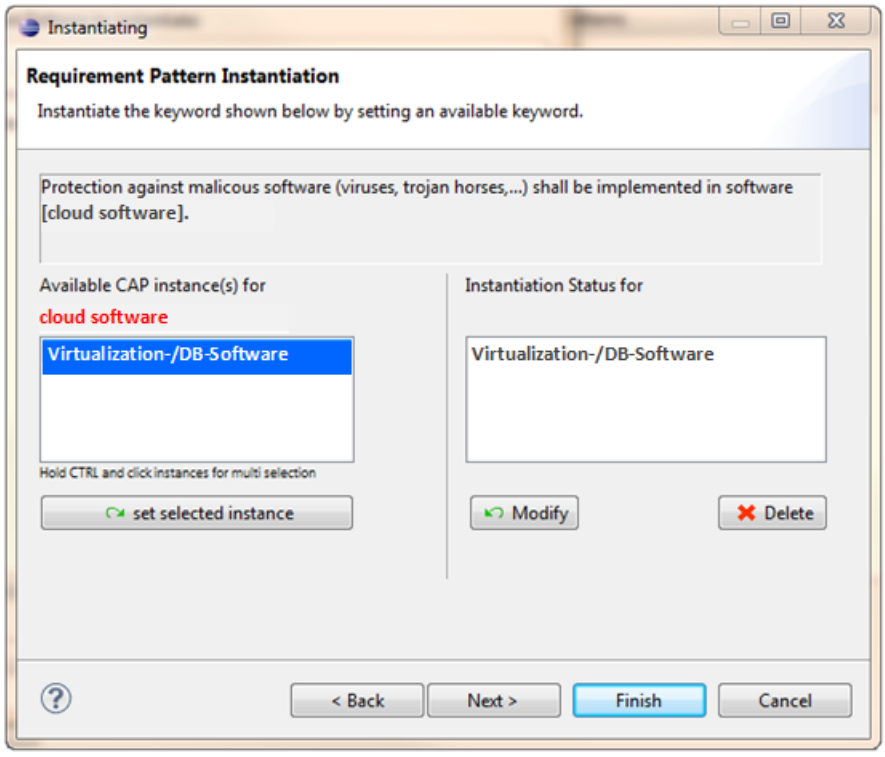

Figure 9. Instantiation of a security requirement pattern in the InstRP Editor

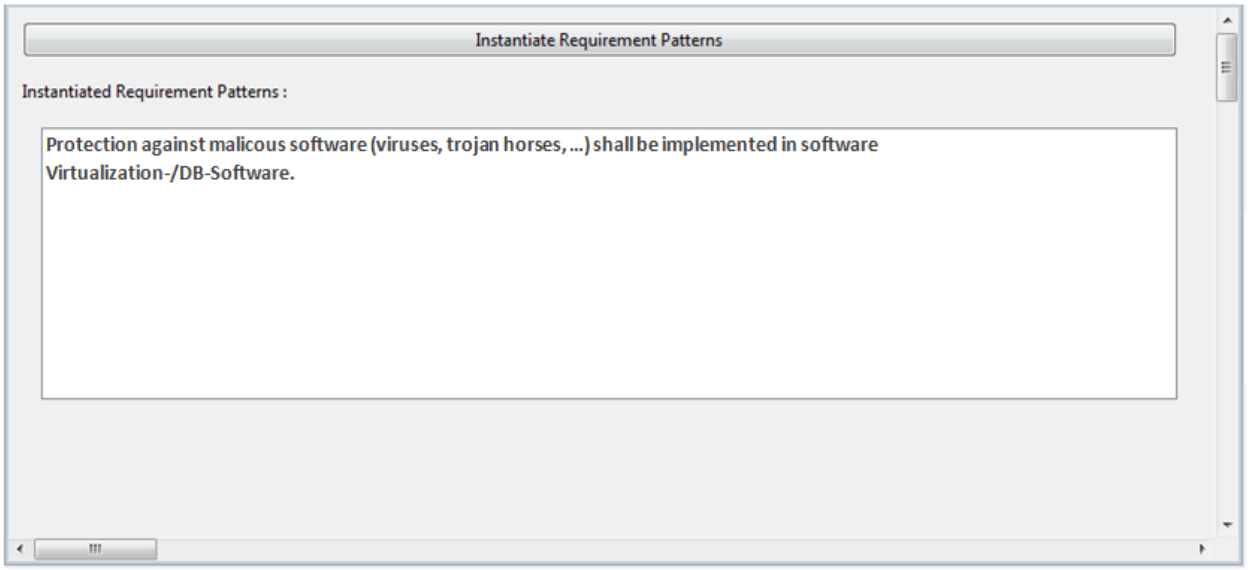

Figure 10. Representation of security requirements in the InstRP Editor

V.Evaluation/Discussion

The procedure presented in this paper was developed based on discussions with practitioners from cloud security projects. Parts of our method have been discussed with security consultants. The security consultants mentioned that this structured method

- helps to understand the scope of the analysis of a cloud and to consider all relevant parts of it.
- supports the identification of security requirements.
- increases the use of models instead of texts in standards, which eases the effort of understanding the system documentation significantly.
- provides the means for abstraction of a complex system and structured reasoning for security based upon this abstraction.

One issue that needs further investigation is scalability, both in terms of the effort needed by the requirements engineers in order to enter all information about the organization as well as the requirements elicitation proposed. We will use the approach for different scenarios to investigate if the method scales for different fields, as well.

Our tool will also undergo a series of usability tests, which shall discover issues with its use in a productive environment. We aim to identify usability issues and resolve these in order to further improve the user experience.

We aim to conduct also an empirical study with our tool in order to analyze the amount of time that can be saved when using it and the amount of security requirements identified when using it. We aim to compare it against conventional text based approaches.

Security requirement patterns and CSAP are linked together. So, if an element in the CSAP instance is changed, e.g., name or type, it is necessary to change the security requirement, as well. Otherwise, we create an inconsistency.

VI.Related Work

Schumacher et al. (2006) propose patterns specifically for security. The authors defined simple solutions to security problems during the software engineering design and implementation phases. The resulting pattern catalogues support specifically design and implementation phases of software engineering processes, while our work focuses on the analysis phase of software engineering.

Withall (2007) provides guidelines and examples for formulating software requirements based upon project experience. The author also explains the need for documentation of 9 requirements including assumptions, glossary, document history and references. Withall's work aims at writing textual requirements, which also consider domain knowledge in the form of assumptions to these requirements. Our work differs from Withall's, because we provide patterns for context descriptions and requirements elicitation.

Fernandez et al. (2007) design several UML models of some aspects of Voice-over-IP (VoIP) infrastructure, including architectures and basic use cases. The authors also present security patterns that describe countermeasures to VoIP attacks. Our work provides additionally tool support and requirements validation. We can also envision to use the models and information from Fernandez et al. to provide an adaptation of our approach for VoIP scenarios.

Hafiz (2006) described four privacy design patterns for the network level of software systems. These patterns solely focus on anonymity and unlinkability of senders and receivers of network messages from protocols, e.g., HTTP. The works of Fernandez et al. and Hafiz focuses exclusively on specific kinds of systems: Voice-over-IP and network-based software systems. We focus on any kind of cloud computing system.

Alexander (1978) developed patterns for building houses and even towns. The author proposed to use patterns to support architects in the process of building houses and towns. In addition, Alexander invented a pattern language, which all patterns are based upon. Alexander defined the term pattern language as a structured method for describing proven design practices for a specific domain. Understanding the language of the pattern supports the development of new patterns and using existing patterns. The difference to our work is that we use a structural description of software systems in combinations with textual requirements pattern. Hence, our work is focused on the analysis phase of software engineering.

Gamma et al. (1994) propose simple textual patterns for design problem the authors encountered frequently during software engineering projects. These textual patterns or templates contain a structure that is similar for each pattern. For example, every pattern has a name, a motivation for its usage and consequences. A complete description can be found in (Gamma et al., 1994). These patterns support software engineers during the design and implemented phases of given software engineering processes. The authors decided not to rely only upon graphical models, because these cannot capture the decisions, alternatives, and trade offs that led to successful design decisions. Nevertheless, graphical representations of the structure of design solutions in classes based on the object modeling technique (OMG). The behavior of diagrams is shown in interaction diagrams. Both kinds of diagrams are explained in detail in (Gamma et al., 1994). In contrast to Gamma et al. we do not focus on design solutions, but rather on a structured elicitation of requirements in the analysis phase of software engineering.

Fowler (1996) developed patterns for the analysis phase of a given software engineering process. The patterns of this author describe organizational structures, processes like accounting, planning and trading. In contrast to the patterns from Gamma et al. Fowler did not rely upon a fixed structure for his patterns. He states that each pattern should have a name, but other than that the fields in the textual templates can be different. The author also uses a graphical notation he defines in the book that contains e.g. class and interaction diagrams. The patterns are derived from experience in projects. Fowler's work differs from

our own, because it focuses on describing economic domain knowledge in particular.

Hatebur and Heisel (2009). proposed patterns for expressing and analyzing dependability requirements. The approach defines important elements of these requirements and checks for consistency in these requirements. In addition, the authors support reasoning for these requirements based upon Jackson's problem frame method (Jackson, 2001). Beckers and Heisel (2012) proposed a similar approach for privacy. Both methods differ from ours, because these focus on formulating quality requirements and in particular security requirements.

VII. Conclusion and Future Work

We have presented a structured method for describing the context of clouds and to elicit corresponding security requirements. We provide a structured selection of security measures that fulfill the security requirements, as well. The controls can be taken from a standard such as the ISO 27001 or from other sources.

Our method provides the means to analyze cloud scenarios with regards to security in a structured manner. The method relies upon patterns to describe the context and elicit the security requirements, which eases the effort for these activities.

Our approach offers the following main benefits:

- A structured method for describing the context and eliciting security requirements and selection of security controls.
- A support tool that contains a graphical representation of the pattern for describing the cloud scenario as well as textual patterns for security requirements and security controls.
- Validation conditions to check the instantiation of all patterns.

In the future, we intend to

- implement all currently developed validation conditions.
- document the context description, security requirements definition, and security controls that fulfill the requirements.
- automatically generate standard compliant reports.
- validate our method further by applying it to different case studies

Acknowledgements

This research was partially supported by the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4Trustworthy ICT, Grant No. 256980) and the Ministry of Innovation, Science, Research and Technology of the German State of North Rhine-Westphalia and EFRE (Grant No. 300266902 and Grant No. 300267002).

References

Alexander, C. (1978). A Pattern Language: Towns, Buildings, Construction. Oxford, United Kingdom :Oxford University Press.

Armbrust, M. , Fox, A., Griffith, R., Joseph, A. D. , Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A. Rabkin, A., Stoica, I. & Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report, Berkeley University, California, United States.

B. Fabian, S. Gürses, M. Heisel, T. Santen, & H. Schmidt (2010). A comparison of security requirements engineering methods. *Requirements Engineering – Special Issue on Security Requirements Engineering*, vol. 15, no. 1, pp. 7–40. Berlin / Heidelberg / New York: Springer.

Beckers, K. & Heisel, M. (2012). A foundation for requirements analysis of privacy preserving software. In Proceedings of the International Cross Domain Conference and Workshop (CD-ARES 2012). Lecture Notes in Computer Science. Springer, 93–107. Berlin / Heidelberg / New York: Springer.

Beckers, K., Küster, J.-C., Faßbender, S. & Schmidt, H. (2011). Pattern-based support for context establishment and asset identification of the ISO 27000 in the field of cloud computing. In Proceedings of ARES, pp. 327–333. Washington, DC, USA: IEEE Computer Society.

Cloud Security Alliance (CSA) (2009). Security Guidance for Critical Areas of Focus in Cloud Computing. Seattle, USA : CSA.

Cloud Security Alliance (CSA) (2010). Top threats to cloud computing. Seattle, USA : CSA.

Eclipse Foundation (2011). *Eclipse - An Open Development Platform*, http://www.eclipse.org/.

—— (2011), Eclipse Graphical Modeling Framework (GMF), http://www.eclipse.org/modeling/gmf/.

—— (2012), Eclipse Modeling Framework Project (EMF), http://www.eclipse.org/modeling/emf/.

—— (2011), Graphical Editing Framework Project (GEF), http://www.eclipse.org/gef/.

European Network and Information Security Agency (NIST) (2009). Cloud computing - benefits, risks and recommendations for information security. Heraklion, Crete, Greece: NIST.

Fernandez, E. B., Pelaez, J. C. & Larrondo-Petrie, M. M. (2007). Security patterns for voice over ip networks. In Proceedings of International Multi-Conference on Computing in the Global Information Technology (ICCGI 2007), vol., no., pp.33,33, 4-9. Washington, DC, USA: IEEE Computer Society.

Fowler, M. (1996). Analysis Patterns: Reusable Object Models. Boston, United States: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Boston, United States: Addison-Wesley.

German Federal Office for Information Security (BSI) (2012). *Security Recommendations for Cloud Computing Providers*. Bonn, Germany: BSI.

Hafiz, M. (2006). A collection of privacy design patterns. In Proceedings of the 2006 conference on Pattern languages of programs (PLoP '06), Article 7, 13 pages. New York, United States : ACM.

Hatebur, D. & Heisel, M. (2009). A foundation for requirements analysis of dependable software. In Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP), B. Buth, G. Rabe, and T. Seyfarth, Eds. LNCS Series, vol. 5775. Springer Berlin / Heidelberg / New York, 311–325.

Heiser, J. & Nicolett, M. (2008). Assessing the security risks of cloud computing. Stamford, USA: Gartner.

International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2005). ISO 27001- Information technology - Security techniques - Information security management systems - Requirements. Geneva, Switzerland: ISO.

Jackson, M. (2001). Problem Frames - Analyzing and structuring software development problems. Boston, United States: Addison-Wesley.

Mell, P. & Grance, T. (2011). The NIST definition of cloud computing (Special Publication 800-145). Gaithersburg, United States: The National Institute of Standards and Technology (NIST).

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F. & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering. Hoboken, New Jersey, United States: Wiley.

UML Revision Task Force, OMG Unified Modeling Language: Superstructure, Object Management Group (OMG), May 2010.

Vaquero, L. M., Rodero-Merino, L. , Caceres, J. & Lindner, M. (2008). A break in the clouds: Towards a cloud definition. *SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55. New York, United States : ACM.

Withall, S. (2007) Software Requirement Patterns. Redmond, United States: MICROSOFT PRESS.