Challenges in Aligning Enterprise Application Architectures to Business Process Access Control Requirements in Evolutional Changes

Roman Pilipchuk[1], Stephan Seifermann[2], Robert Heinrich[2] and Ralf Reussner[2]
[1]*FZI Research Center for Information Technology, Friedrichstraße 60, 10117 Berlin, Germany*
[2]*Karlsruhe Institute of Technology, Am Fasanengarten 5, 76131 Karlsruhe, Germany*
pilipchuk@fzi.de, {stephan.seifermann, robert.heinrich, ralf.reussner}@kit.edu

Keywords: Access Control, Business Process, Enterprise Architecture

Abstract: Business processes define requirements for software systems that support business goals. Enterprise Application Architectures (EAAs) organize the structure and behavior of the required software systems. Satisfying requirements regarding the confidentiality of information that originate from the business process design is crucial to fulfill legal obligations and corporate policies. Violating these obligations and policies can lead to high fines and lost assets. There is a gap in modeling confidentiality requirements holistically across business processes and EAAs (Alpers et al., 2019). Hence, aligning EAAs with business processes by identifying violated business access control requirements (ACRs) during the architectural design phase is vital. Thereto, three challenges need to be overcome: i) define the meaning of read and write from ACRs for EAAs, ii) identify relevant parts of the EAA affected by ACRs and iii) define rules to cope with data type refinement. In this paper, we present the challenges, solutions to them and our scientific findings that we made during the development of AcsALign, which is an approach to align the EAAs to ACRs of business processes in the early design phase and evolution scenarios using the established modeling languages Business Process Model and Notation (BPMN) and Palladio Component Model (PCM). We apply our solutions in a real-world case study. Evaluation results show satisfying accuracy of the requirements extraction and architectural alignment.

1 INTRODUCTION

Business processes describe activities that organizations carry out to generate values for customers or themselves. Software systems support employees in fulfilling their process activities. Therefore, business processes define requirements for software systems. These include access control requirements (ACRs) forbidding or granting users and systems access to services or data they need for their activities. The system structure and behavior organized in an Enterprise Application Architecture (EAA) must satisfy business process requirements. Otherwise, systems violate corporate policies and legal obligations, such as given by the General Data Protection Regulation (GDPR) (European Parliament, 2016), that business processes address. Meeting all ACRs from business processes during EAA design is challenging due to organization complexity and high stakeholder involvement. If architects fail to detect violated requirements, the implementation usually also violates the requirements, which leads to higher costs for fixing compared to an earlier development phase (Boehm and Basili, 2001). Access control issues can be even more costly because of high fines of privacy regulations. There are approaches (Löhe et al., 2014; Open Group, 2018; Urbaczewski et al., 2006; D. Brucker et al., 2012; Ramadan et al., 2017) trying to align business processes and EAAs but they are hard to apply because of their complexity and usage of uncommon models (Kotusev, 2017). A literature review (Alpers et al., 2019) identified a gap in modeling confidentiality requirements holistically across business processes and EAAs. Hence, aligning EAAs with business processes by identifying violated business ACRs during the architectural design phase and evolution scenarios is vital. To achieve this, three challenges need to be overcome. First, relevant parts of the EAA need to be identified that are affected by business ACRs. Second, an appropriate definition for data types read and written by service calls need to be specified with respect to an analysis of violated ACRs. Third, rules have to be defined on how to compare access permissions stemming from business ACRs with the relevant parts of the EAA taking the complexity of data type refinement into account.

This paper presents our scientific findings regarding the challenges that we have found during the development of an approach called *Access Permission Architecture Aligner* (AcsALign) that exploits Business Process Model and Notation (BPMN) models commonly used to describe business processes to identify violations of ACRs in Palladio Component Model (PCM) models commonly used to describe IT architectures. As EAAs organize structure and behavior of software systems with the same types of elements (e.g. system, interface), PCM is suited to define EAAs. AcsALign helps enterprise architects to identify ACR violations automatically during design time, i.e. they can align EAAs to business process ACRs. This is especially crucial during evolution scenarios. We assume business processes to be correct, i.e. we do not challenge their compliance with laws, as our goal is to align information about ACRs from business processes with architectural design. Guidelines like (AXELOS, 2011) help to design appropriate business processes. Hence, the assumption is reasonable.

In this paper, we present the following contributions: (i) a definition of relevant EAA parts regarding business process ACRs, (ii) a definition of *read* and *write* actions with respect to the analysis of violated ACRs and (iii) an algorithm to detect violated ACRs in EAAs by taking the complexity of data type refinement into account. We apply our contributions on a real-world case study of a national art gallery.

The remainder of the paper is structured as follows. After providing required foundations in Section 2 and describing the challenges in Section 4, we give a short overview of AcsALign in Section 5. Sections 5.1 to 5.3 elaborate on our contributions regarding the three challenges to align EAAs and business processes with respect to ACRs. In Section 6, we evaluate our contributions on a real-world case study consisting of ten business processes, 83 ACRs and 16 systems. We achieve a satisfying accuracy. The paper finishes with a discussion of related work in Section 3 and a summary including future work in Section 7.

2 FOUNDATIONS

Our approach makes use of BPMN for modeling business processes and PCM for modeling EAAs. Therefore, we introduce both modeling notations briefly.

The Business Process Model and Notation (BPMN) (Object Management Group, 2011) is the de facto standard language for modeling business processes in organizations. It uses graphical notations to visualize the flow of activities for interacting participants. Figure 1 shows two processes modeled in BPMN that are part of our running example introduced later. Participants such as the *store manager* are represented by lanes. The surrounding pool such as *Store* defines the organizational division of the participant. Each lane contains activities represented by rounded rectangles that are interconnected with arrows representing the flow of interactions. Data objects like *Advertisement Schedule* may be annotated to activities representing input and output flows of data in the activity. Circles indicate starting/ending conditions of a process. In order to fulfill the business process, all activities must be carried out by the participants. This requires appropriate access permissions.

The Palladio Component Model (PCM) is the modeling language of the Palladio approach (Reussner et al., 2016) for predicting various quality properties of architectures including performance or reliability. In our approach, we use PCM to describe EAAs, which are the landscape of systems supporting business processes, and its behavior. We make use of two viewpoints: The structural view point covers components, their interfaces and how instances of these components are assembled to a system. The behavior view point covers the behavior of users, given by a sequence of calls to system services and components, given by a sequence of actions affecting quality properties of the system.

3 STATE OF THE ART

The field of aligning business processes and EAAs with respect to security as well as the field of access control analyses in EAAs to be the most related. We briefly report on the respective related approaches in the following. The transition to the implementation (incl. the integration of access control systems during the implementation phase to enforce policies at runtime) is not in our focus.

Alignment between business processes and EAA provides considerable benefits (Giaglis, 2001). Enterprise Architecture Management (EAM) is the general term for the common approach to achieve such an alignment. In Architecture-driven IT management (ADRIMA) (Löhe et al., 2014), EAM involves initiating and establishing processes as well as governance and definition of application scenarios by determining involved models and their lifecycles. Frameworks like TOGAF (Open Group, 2018) and FEAF (Urbaczewski et al., 2006) define various types of models and their interrelations for developing an enterprise architecture. However, they are considered hard to apply (Kotusev, 2017) because of the complexity of the corresponding process and the required detailed plan-

ning. EAM is considered a challenging task (Löhe et al., 2014), in general. Even if the models provide all required means for expressing security, creating them is complicated and requires many stakeholders with deep knowledge of specific models. Approaches to extend BPMN like (D. Brucker et al., 2012; Salnitri et al., 2015; Rodríguez et al., 2007) introduce various security related information into BPMN, which leads to the same issues. They are not as widespread as BPMN itself and require extra effort of experts to model the additional information in BPMN. In contrast, AcsALign works with de facto standard modeling languages to require low additional effort for organizations. The approach around IntBIIS (Heinrich et al., 2017) also addresses mutual dependencies between business processes and architecture but focuses on performance predictions rather than access control. (Ramadan et al., 2017) proposes an approach to align architecture models and business processes with regard to security policies. Therefore, they transform security policies from secBPMN2 models to architecture models in UMLsec. While the idea is similar, the realization achieves different goals. They focus on non-standard models that introduce security specific elements. However, such extended models are not widespread and require additional effort of security experts during the design phase. We follow the same idea but focus on de facto standard modeling languages and tailored our approach in a way to impose least possible effort during its utilization.

There are various approaches to analyze architectures regarding access control that the comprehensive survey (Nguyen et al., 2015) of Nguyen et al. lists. None of the approaches that aim for the same goal as our approach work on non-extended standard models. We cannot discuss all but focus on the approaches that make use of standardized models that are extended by security concepts. Often, the intention of the analyses is different to ours, which impedes reusing these approaches. Approaches like (Ahn and Hu, 2007; Busch et al., 2014) aim for integrating security mechanisms in models or code by providing means for modeling them. We cannot make use of such approaches because they do not provide analyses of data usage. Another common goal of approaches like (Jürjens, 2005; Georg et al., 2009) is the analysis of attacker behavior. These approaches analyze information flows but require modeling attackers and their capabilities. In contrast, our architectural analysis only requires additional information tailored to the use case.

Another group of approaches like (Alghathbar and Wijesekera, 2003; Lodderstedt et al., 2002; Goudalo and Seret, 2008) aim for extending UML with security information. Some allow modelling ACRs on the architectural level. Their purpose is different as they do not focus on closing the gap between the organizational level and the IT level by aligning the models of both. Furthermore, they focus on defining security related information explicitly rather than extracting ACRs from the organizational level. Also, such approaches require additional effort of security experts.

The approach in (Abramov et al., 2012) proposes a systematic way to enforce ACRs from organizational level on architectures and databases. The difference to our approach is that they focus on architecture diagrams rather than business process models at the organizational level. ACRs and also security patterns need to be modeled explicitly by security experts in extended UML class diagrams and extended UML use case diagrams. In our approach, we focus on supporting commonly used methodologies and de facto standard modeling languages by extracting implicitly modeled ACRs to analyze the correctness of the EAA without an additional effort for security experts.

4 CHALLENGES

We identified three main challenges in aligning EAAs with business processes. To illustrate these challenges and the concepts in the remainder of this paper, we introduce a running example of a supermarket chain based on the community case study Common Component Modeling Example (CoCoME) (Heinrich et al., 2016). Initially, the supermarket supports buying goods locally. Orders of a mandatory loyalty program are used for marketing purposes. After a system evolution, there also is an online shop not eligible for the loyalty program. We focus on the business processes from Figure 1. In the following, we explain the three main challenges.

Relevant EAA parts affected by ACRs have to be identified as these parts are prone to human errors made by enterprise architects (see Section 1). The major issue in identifying corresponding parts is that EAAs refine the business processes extensively, so there is no simple one-to-one mapping. Consequently, identifying relevant EAA parts by just looking up the result of a simple mapping process is not possible because this mapping process is highly creative. In our running example, such a refinement happens for the *Prepare customer profiles* activity in Figure 1. In BPMN, this is just an activity but to provide this functionality in the EAA, there is the need for a dedicated *Marketing* component that retrieves data from a *CustomerDataStore* to calculate the profiles as shown in Figure 2. Certainly, it is not enough to just consider the marketing component here but how far should we

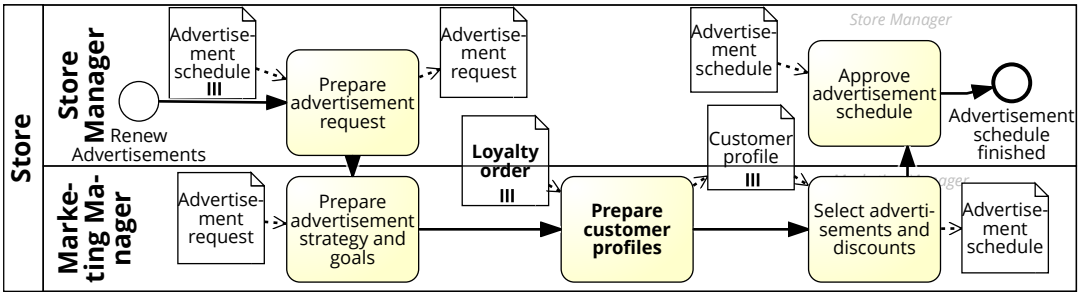

Figure 1: CoCoME business process *Prepare Advertisements and Discounts* (in BPMN).

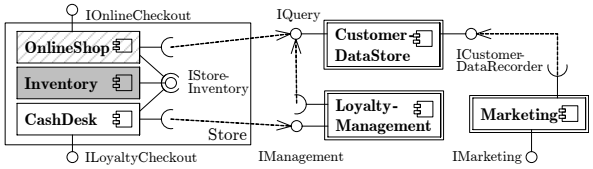

Figure 2: Simplified EAA of CoCoME (in UML-like notation).

trace relevant components back? Dozens of refinements are done to design an appropriate architecture. If unnecessary parts of the EAA are considered during the analysis, the alignment becomes slower and might require modeling of additional information. If not all relevant parts are considered, the alignment becomes incomplete and misses vital errors.

The meaning of read and write actions known from ACRs has to be defined in the context of EAAs. In business processes, data objects might be required as input (read) to fulfill an activity and activities might produce data objects as output (write) during the fulfillment of an activity. If the definition of read and write in the context of EAAs is wrong, errors are missed and false positives will occur. In the EAA there are many potential candidates to detect read or written data such as parameters and return types of services, variables in services, objects, databases and calls to IDs. Thus, it is important to understand what read and write in business processes mean in the context of EAAs. Moreover, the processing of data in service calls might become very complex due to composition and decomposition of data. For example, the data object *customer profile* is generated in a service call by combining various data like clustered customer groups, products, discounts based on the previous analysis of data in the collection *loyalty order*.

Data type refinements are done by the enterprise architect during the design of the EAA. Data objects in business processes are high-level and designed from the business point of view. The enterprise architect has to design data types that reflect data objects from the business processes, but designs them from the IT point of view. Data types usually are not standalone but are combined or collected. There are several ways of refinement: 1. define new collection data types, 2. define new composite data types and 3. define new primitive data types. Thus, data type refinements make simple equality checks insufficient. It is an important factor that imposes complexity to the identification of ACR violations, as the specification of integral parts of data objects are not known on the business level but are part of the work of the enterprise architect. A call in the EAA can transport data types that are known from the business processes but also new data types created by the enterprise architect and thus, unknown by the ACRs of the business processes. Furthermore, combined data types of these known and unknown data types may flow via a service call. The degree of composition is not limited, i.e. data types consisting of data types which themselves consist of data types and so on are normal in the EAA. The collection *loyalty order* from Figure 1 exemplifies this complexity. In the EAA, the collection loyalty order consists of the composite data type loyalty order. This composite loyalty order consists of a customerID, loyaltyID, a list of purchased products, their prices, the order number, the cashierID and a date. Some of these data types like the customerID are created newly and are not known by the ACRs of the business processes. Other data types like list of products are a collection. Furthermore, some inner data types are again a composition of known and unknown data types like the product. This opens room for various combinations of collections, compositions, known and unknown data types. Consequently, data types that flow during service calls might be completely different from the data objects in business processes. This spans many possibilities on how data objects from business processes (that are part of ACRs that need to be validated in the EAA) can be related to data types that flow in service calls of the EAA. The challenge is to decide if a data type is allowed or forbidden to flow during a service call even if there are no ACRs for exactly this data type.

5 OVERVIEW OF THE ACSALIGN APPROACH

The goal of AcsALign is to align EAAs with business process ACRs by detecting violations automatically. Mistake resolution is done manually by the architect but AcsALign supports it with tracing information and violation details.

Access Control Requirement Extraction: First, AcsALign extracts ACRs based on data used in business processes and stores ACRs linked to business processes in a trace model. Afterwards, ACRs are transformed into role-permission-pairs describing which data types a role can read and write. We do not focus on creating meaningful business processes or architectures but expect them to be already modeled. This assumption is feasible as organizations create such models for compliance purposes anyway.

Architectural Alignment: Second, an architectural analysis determines data used in services and analyzes them for ACRs violations. For the tool realization, we used a data flow analysis and extended it with data properties and propagation rules tailored to data type analyses. Afterwards, a matching algorithm (explained in Section 5.3) compares architectural data type usage with data type usage defined by business processes. A detected issue means that the architecture does not adhere to ACRs from the business process and violations have to be revised.

Mistake Resolution: Third, architects meet ACRs of business processes by addressing identified violations. AcsALign supports by providing information about violations. First, information about violated ACRs is given via a trace model, i.e. the a) affected ACR and b) affected activities in business processes incl. lanes and data objects. Such traces make analysis results comprehensible by providing a bigger picture about the business process context of service calls (with information of the affected service calls, affected roles and their work tasks and interrelations to further activities and service calls of the affected process) and by enabling to identify correct business process owners to clarify their intentions. Second, information about the architecture violation is given by the a) initial service call, b) data flow through the system and c) violating action incl. operations (read/write) and involved data types. This helps to understand which system's service is affected and the intended ACR design decisions of the business. AcsALign does not automate this step but the architect can run AcsALign again after the fix to validate changes.

AcsALign is compatible with business process languages like BPMN with characteristics to group activities and express data objects entering and leaving activities. It is compatible with architecture languages like Unified Modeling Language (UML) that have call-and-return semantics, user behavior and data propagation. These selection criteria apply to most commonly used languages. In the remainder of the paper, we use IntBIIS (Heinrich et al., 2017) that models a subset of BPMN, and PCM (Reussner et al., 2016). Still, our approach is compatible with other languages as mentioned before.

5.1 Affected EAA Parts

In order to align the EAA to business ACRs, relevant parts of the EAA have to be identified that are affected by ACRs, as these parts are prone to human errors made by the enterprise architect (see Section 4). We analyzed the effect of ACRs on all meta-elements of the architectural language that we use to represent EAAs. As the architectural language is a typical modeling language for IT architectures and we lift the identified elements to a more general level, we assume the results to be representative for other EAA modeling languages as well. To identify affected elements, we first identify the essential parts of an ACR. The data object and its operation are certainly the significant parts of the ACR. To violate an ACR, a user has to get access to data he should not have to or trigger write actions for data that he might have access to but are not allowed to write. Hence, all meta-elements that have an influence on the data flow path or the transported data objects are relevant for considerations. In the EAA data objects are represented as data types and the data flow path of services calls in systems and components.

Our research shows that four parts of the EAA are affected by business ACRs. We call them mistake types (MT) because an architect might introduce an error via these meta-elements during the design time:

MT1) parameters and return types of services (explicit data type assignment)

MT2) external Calls (wrong data type received/sent)

MT3) wiring of systems/components (wrong data type received/sent)

MT4) data type refinement (illegal composition of data)

In MT1, to much data might flow due to falsely specified service parameters or return types. This may lead to either forbidden write operations to data through storage and modification or forbidden read access if to much or wrong data is returned by the return types. During MT2, falsely specified external calls in the component behavior might lead to falsely received and sent data types. In MT3, a wrong wiring might

lead to the same problem because components and systems might specify or be connected to a wrong interface. During MT4, data types might be refined erroneously leading to a flow of too many data types. Coping with the various data type refinements is a challenge on its own. Thus, we address it in Section 5.3. To sum up, the four aforementioned MTs are the parts of the EAA that are affected by ACRs and have to be considered during an alignment.

5.2 Read and Write in EAA Context

To analyze the EAA for violations of ACRs, it has to be defined what the operations read and write of business ACRs mean in the context of EAAs (see Section 4). In the context of business processes read and write during an activity mean that data is required as input in order to fulfill the activity or is produced as output that is later on required during another activity. In both cases, the required or produced data is persisted. During our research, we found that the following definition of read and write in the context of EAAs is appropriate and allows to abstract from the various possibilities to transfer data in architectures (like parameters, objects, etc. explained in Section 4). Write in the context of activities means that new data is persisted or persisted data is modified (this includes any modification as deletion). In the EAA, data is permanently stored or modified if it is part of a database. Therefore, services are called to store or modify data in a database. Consequently, it is important whether the data to write is stored in a database and not how it is transferred between systems and components. In the context of EAAs, write means that data is stored or modified in a database by a service that is called during an activity. Transitive calls need to be considered.

Read in the context of activities means that data is required as input to fulfill the activity. In the context of EAAs, service calls fetch the required data. Thereby, it is important which data arrives at the end of the called services of an activity. It is not relevant where this data is actually coming from or which other data is read in order to fetch the previously mentioned data as the person or system who calls the services during the activity is only able to read the data that is received upon the end of these service calls (and not the data that is processed during these calls and during transitive service calls). Hence, the system facade needs to be considered as well as the data that is returned during service calls of the system facade. In the context of EAAs, read means that data is returned during the end of service calls (found in the system facade) of an activity.

To sum up, the system facade and databases that are part of the EAA need to be identified. With regard to databases service calls that store or modify data need to be considered for write access and data returned to services of the system facade need to be considered as read access.

Determining read and written data according to the aforementioned definition is possible by means of a data flow analysis. This is necessary because simply comparing data types used in service signatures is not sufficient to consider refined data types such as polymorphic data types. We extend a data flow extension for Palladio (Seifermann et al., 2019) with a taint-like mechanism to detect the origins of data flows as described above. The data flow analysis tags data with the actual data type, i.e. the actual polymorphic data type, and considers the effect of data processing. For example merging two data types into a composed data type. In addition, there are tags for the traversed operations of service calls. Every traversed operation adds a new tag. This allows to reason about the source of a call, which is important to identify if data arriving at a store has been sent by an allowed subject. By using both types of tags, we can identify actual data types at each location in the EAA and always trace a data item back to a subject that issues the call. In our running example, the actual data type of the order leaving the online shop component is the same as the type of the received order. The *CustomerDataStore* receives orders from the online shop and transitively from the cash desk, so it can return data of both types to the *Marketing* component. This component takes the data and produces customer profiles. The marketing manager preparing the customer profiles receives these customer profiles.

We determine read data types for a user by collecting all actual data types that are returned during service calls. We determine written data types by collecting all actual data types for parameters sent to a database that trace back to a system call of the user.

5.3 Data Type Refinement

In order to analyze the EAA for ACR violations, it has to be defined how to cope with the challenge of data type refinements made by architects. Due to them simple equality checks are insufficient (see Section 4).

In our research we divided the problem into data flows of collections and compositions. Primitive data types are part of compositions, as they represent a data type without inner data. Furthermore, data types in both classes of data flows might be known by a business process or not. They are known when the data types have corresponding data objects defined in the

business processes. Throughout the data type refinement, new data types of arbitrary complexity are created by the architect. These are the data types that are unknown by the business processes. We explored what an ACR means for both classes of data flows taking unknown data types into account and created rules when a data flow is allowed/forbidden in the presence of an ACR. These rules were combined and simplified forming the algorithm shown in Algorithm 1. This algorithm provides the mistake detection.

Algorithm 1 Pseudo-code to derive if a given data type d is allowed

```
1:  function ALLOWED(d, D_known, D_allowed)
2:      allowed ← (d ∈ D_known ⟹ d ∈ D_allowed) ∧
    (d ∉ D_known ⟹ fallback)
3:      if allowed ∧ isComposite(d) then
4:          for d_i ← d.innerDataTypes do
5:              allowed ← allowed ∧
    allowed(d_i, D_known, D_allowed)
6:          end for
7:      end if
8:      if allowed ∧ isCollection(d) ∧ d ∉ D_known
    then
9:          allowed ← allowed ∧ d.innerType ∉
    D_known
10:     end if
11:     return allowed
12: end function
```

A special case is formed when an unknown data type flows in a service call. As they are unknown to business processes, these cannot define how to handle them with regard to access control. Therefore, our algorithm proposes the variable $fallback$. It has to be defined by the architect prior to execution. Using $fallback = false$ is appropriate for high risk environments because it denies access to all unknown data types. Using $fallback = true$ is more permissive and grants access to all data types not known, which can be useful in case of many data type refinements. We use the latter one in this paper. The algorithm takes all ACRs for a business process activity and compares them with the identified data access of all service calls belonging to that activity. D_{known} is the set of data types known from business processes. $D_{allowed}$ is the set of data types allowed by the ACR. A data type known to business processes has to be allowed explicitly (line 2). Otherwise, the fallback applies. For a composed data type all of its inner data types have to be allowed as well (lines 3–7). If this is not the case, more data types are read or written than allowed by the ACRs. A collection data is only allowed if it is known or its inner data type is allowed as well (lines 8–10). In the first case, it is in D_{known}. In the second case it is not in D_{known}. For the second case it has to be checked if the inner data type is in D_{known}. If true, the collection is forbidden as the inner data type is known by the business processes. The reason is that the business processes would have defined the collection for the activity if it were intended as an input or output, as the inner data type and thus, its collection is known by the business processes. If false, the collection and its inner data type are unknown and $fallback$ applies. In our running example, the marketing manager can read collections of loyalty orders during the preparation of customer profiles as illustrated in Figure 1. The data flow analysis detects that the manager accesses *LoyaltyOrder[ ]* and *OnlineOrder[ ]*. The matching algorithm shows that online orders are not allowed to be read, which indicates a mistake in the architecture. To sum up, an algorithm is required that copes with the various data type refinements as well as known and unknown data types to decide for a data flow in presence of an ACR whether the data is allowed to flow or forbidden. We evaluated the algorithm on a real-world case study in Section 6.

6 EVALUATION

This section describes the case study based evaluation of the solutions for the challenges realized in the approach AcsALign. Section 6.1 describes evaluation goals and design of the case study. The studied case is described in Section 6.2. Section 6.3 and Section 6.4 cover arguments as the first part of the evaluation. Section 6.5 presents and discusses results and Section 6.6 covers threats to validity. Finally, Section 6.7 discusses limitations and assumptions of AcsALign.

6.1 Evaluation Goals and Design

We structure the evaluation based on the Goal-Question-Metric method (R. Basili et al., 1994). Goals are the evaluation objectives to be achieved. Metrics allow answering evaluation questions that contribute to achieving the objectives.

We evaluate all of our three contributions. Contributions C1 and C2 are both definitions building the foundation of AcsALign. Definitions cannot be wrong but they can be unrealistic or inappropriate, which degrades the applicability. Therefore, we define our first evaluation goal **G1: Evaluate the appropriateness of the underlying definitions**. Contribution C1 defines the set of architectural elements that are relevant for checking compliance with access control requirements. Contribution C2 defines what

a read and write operation actually is in terms of the EAA. The important evaluation question to ask in order to rate the appropriateness of the definitions are: **Q1.1: Does the set of elements contain all architectural meta-elements that influence the detection of read and written data types? Q1.2: Does the definition cover all activities of an EAA that can be considered a read/write operation?** We answer these questions by arguments in Sections 6.3 and 6.4.

An analysis (C3) is only useful if its accuracy is satisfying. In the following, we use the term *mistake* to describe a concrete error introduced in the architecture. A *violation* is the result of the mistake, i.e. a violated ACR. Low accuracy means a user cannot trust the results and has to compensate missed or falsely reported requirement violations by manual tasks. In the worst case, the analysis provides no benefit compared to pure manual inspection. Therefore, the second evaluation goal is **G2: Evaluate the accuracy of AcsALign for discovering violations of ACRs that originate from business processes in EAAs**. Evaluating the accuracy of the overall approach refers to evaluating the accuracy of its single steps. We evaluate the steps separately to better locate potential accuracy issues. We do not consider the accuracy of the mistake resolution step because it is a manual step and highly depends on the abilities of an architect and cannot be evaluated in an objective way. We ask two evaluation questions: **Q2.1: What is the accuracy of the ACR extraction? Q2.2: What is the accuracy of the architectural analysis?** By answering these questions, we want to examine two common and relevant situations caused by low accuracy that lead to increased manual effort as motivated before: (i) is a reported violation an actual mistake (soundness) and (ii) have all mistakes been reported (completeness) We answer the evaluation questions Q2.1 and Q2.2 as part of a case study because we aim for a real-world scenario that allows for deeper understanding and better realism of the phenomena under study.

To answer Q2.1, we execute the extraction process on the case described in Section 6.2 and classify the results based on a reference list of ACRs. Two postgraduates independently analyzed all business processes by hand to develop a common reference list of ACRs. The studied case contains all model elements that can have an influence on ACRs as described in Section 5.1. We classify each individual ACR of a business process activity. We classify an ACR as true positive t_p if the ACR exactly matches the ACR in the reference list. We classify an ACR as false positive f_p if the ACR has no exact match in the reference list. We add ACRs from the reference list to f_n if the extraction did not yield these ACRs.

Based on this classification, we calculate two established metrics for accuracy. These metrics address the situations caused by low accuracy as discussed before: **M1 Precision** $m_p = \frac{t_p}{t_p+f_p}$, to address (i) and **M2 Recall** $m_r = \frac{t_p}{t_p+f_n}$, to address (ii).

To answer Q2.2, mistakes are injected in the correct EAA, a reference list of violations is created, AcsALign is executed and results are classified. The initial EAA modeled in PCM is a version for the case described in Section 6.2 that initially has no mistakes. We categorize all possible mistakes into mistake types that have the same root cause. They were discussed in Section 5.1. These are mistake types that are not specific for a used modeling language but for EAAs in broad. To evaluate the accuracy of the analysis, it is sufficient to inject one mistake of each mistake type into the architecture because more mistakes of the same type would be handled in the same way and yield the same result. Two postgraduates build the reference list of violations by recording the injected mistake, its expected effect and the violated requirement. We classify a reported violation as t_p if the violation is part of the reference list. We classify a reported violation as f_p if the violation is not part of the reference list. We add violations from the reference list to f_n if the analysis did not report this violation. We calculate the previously mentioned metrics M1 and M2 based on the classification results.

6.2 Studied Case

The case we study is the result of a consulting project for a national art gallery that wanted to streamline its EAA to improve business process support. A set of business processes for the preparation of an exhibition and an EAA has already been created, i.e. the authors did not create them. The case is appropriate for the evaluation of our approach as there is a comprehensible set of business processes that contain interaction between actors, data type definitions and data usage descriptions. The EAA contains usual data processing patterns for information systems, including delegation, merging, or reading from and writing to databases. The size of the studied case is reasonable as shown in the characteristic overview in Table 1.

All business processes shown in Figure 3 are subprocesses required for the preparation of an exhibition. For further details, we provide all models as part of a data set (Pilipchuk et al., 2021) incl. analysis results. At first, a concept including the planned artworks for the exhibition is created by the curator. After reaching a budget agreement with the directors, external artwork is borrowed. Usually, the gallery lends artworks to other galleries. This set of busi-

Table 1: Characteristics of the business processes and the EAA of the studied case.

Business Characteristic	#	Business Characteristic	#	EAA Characteristic	#
Business processes	10	Activities	75	Systems	16
Lanes	34	Data objects	56	Service Calls	17
Roles	12	Access Control Req.	83		

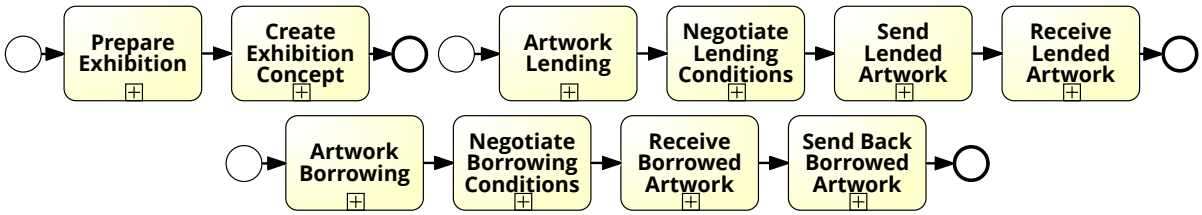

Figure 3: Overview of sub-processes for the exhibition preparation (in BPMN).

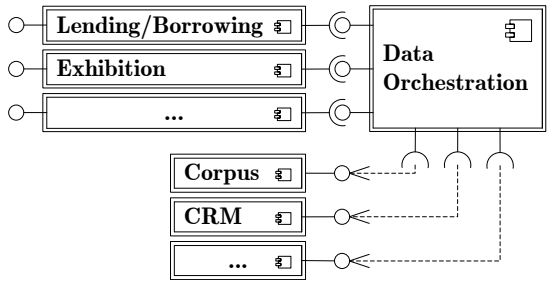

Figure 4: Excerpt of EAA of the national art gallery (in UML-like notation).

ness processes provides a suitable base for analyzing ACRs because they encompass data flows of confidential information like financial budgets, insurance values and customer data.

The supporting system illustrated in Figure 4 is a middleware that interconnects different systems and provides views on subsets of data that the different employee roles of the art gallery require. Individual systems such as *Corpus* or *CRM* provide data that the *Data Orchestration* combines and provides to systems accessible to the users such as the *Lending/Borrowing* system or the *Exhibition* system.

6.3 Appropriateness of Read and Write Definition

We start with the argument for the appropriateness of C2 because the argument of C1 builds on it. C2 defines what a read and a write operation actually is in terms of an EAA. First of all, users can basically do anything with information they already have access to, which means they can also write and read it. Therefore, restricting access to information users already have is barely possible. We exclude digital rights management here because this is not the focus of AcsALign. Therefore, the access control system has to take care of information that is exchanged between the users. These exchanges can be i) a user transitively reads information from another user and ii) a user changes information another user can read All of these exchanges require some sort of memory that stores the information because users in information systems operate in sessions that usually do not support direct information exchange. Description languages such as Palladio apply this pattern as well.

This means that if the system returns information to a user, the information is either from himself/herself or it has been loaded or derived from other stored information. In both cases, the user requires proper read access permissions for the received data. Because Role-based Access Control (RBAC) is not about information flow control, it only matters what the user actually receives. Therefore, it is sufficient to check the read permissions at the system boundary.

If the user wants to change information that is potentially exchanged with another user, he/she has to modify memory. Therefore, all information passed to memory can be considered as written and no information else. In Palladio, the only memory writable by users are stores, so we consider all information passed to a store as written. In other modeling languages, it might be necessary to include further memory implementations but the pattern still applies.

6.4 Appropriateness of Selected Modeling Elements

Contribution C1 defines the set of architectural elements that are relevant for checking compliance with access control requirements. In the previous section, we already showed that detecting read actions at the system boundary and write actions at stores is appropriate. In this section, we show why the selected architectural elements are appropriate to detect read and

Table 2: Excerpt of extracted ACRs.

Role	Read Perm.	Write Perm.
Management: CEO	–	Lending-Confirmation
Research: Curator	ForeignArtwork	Lending-Request

written data as well as to decide about permissions. To find read data, we have to know the data types returned to the user at the system boundary. Just looking at the signature is not sufficient because this does not consider polymorphic types. To determine the actual data type, we have to follow the control flow in reverse order to the point where we can decide about the data type. Such a point is reached if data of a certain type is created. Therefore, we have to consider all elements that influence the control flow, which are calls of the user to the system, interfaces including signatures, components, calls between components and the actual wiring of components. To find written data, we have to know the data types passed to a store. The procedure is the same as for read data but we follow the control flow starting from the store. To decide if a data type is allowed to be read or written, we have to match it with the access permissions. To support refined data types not directly covered by the permission set, we require information about the actual refinement. Therefore, it is necessary to also consider the structure of data types.

6.5 Evaluation Results and Discussion

We conducted the case study as described before. For question Q2.1 regarding the accuracy of ACRs extraction, we achieved 83 true positives, zero false positives and zero false negatives. This brings us to a precision $m_p = \frac{83}{83+0} = 1.0$ and a recall $m_r = \frac{83}{83+0} = 1.0$. Hence, the extraction algorithm successfully identified all ACRs that could be identified from business processes according to the previously defined rules and did not report any false positives. Table 2 shows an excerpt of the extracted ACRs. The reference list of ACRs has the same format and matching ACRs is unambiguous when ignoring the order of data types in read/write permissions. This is feasible as the order has no meaning for the permissions. All extracted ACRs have an equal counterpart in the reference list.

For question Q2.2 regarding the accuracy of the architectural analysis, we achieved four true positives, zero false positives and zero false negatives. This brings us to a precision $m_p = \frac{4}{4+0} = 1.0$ and a recall $m_r = \frac{4}{4+0} = 1.0$. This means that the analysis successfully identified all injected mistakes in the architecture and did not falsely report mistakes. The four correctly found mistakes are shown in Table 3. In MT1, the system returns a lending request but it must not return anything. In MT2, the system must return a foreign exhibit object but it returns an own exhibit object because a wrong service has been called internally. In MT3, the system returns two data items because a service has been wired wrongly. The system must not return a foreign exhibit object in this case. In MT4, the system returns the correct data types but the lending contract has been refined in a way that it now contains an exhibit object instead of a public exhibit object. This violates an ACR because the permission for writing arbitrary exhibit objects is missing. All reported mistakes match mistakes of the reference list.

6.6 Threats to Validity

We discuss the four aspects of validity according to Runeson et al. (Runeson et al., 2012, pp. 71) that have to be distinguished in case study research.

Construct validity ensures that taken measures represent what shall be researched. We used common accuracy metrics and provided a reasonable classification scheme aligned with the analysis definitions to collect the input data for the metrics. As part of the evaluation design, we already explained the relation between the evaluation question and the used metrics.

Internal validity ensures that an expected influencing factor is not affected by other factors. We expect the extraction algorithm and the analysis algorithm, the analyzed models, the result classification and the injected mistakes to influence the results. The extraction and analysis algorithms are the factors we want to check. We ensured that the models contain all relevant scenarios to cover the whole algorithms. This eliminates a positive influence of the models because a certain type of requirement or mistake that could potentially be missed is not part of the models. The mistakes were injected by the authors. However, we categorized all possible mistake types that influence ACRs and injected one mistake for each category. This is sufficient to evaluate accuracy because more mistakes of the same type would be handled in the same way and yield the same result. We derived the scenarios from general definitions that could apply to other algorithms as well to avoid creating scenarios that are tailored to the approach. We also ensured that the models correctly reflect the intention of the modeler to eliminate unintended modeling flaws as an influencing factor. We derived the scenarios and the classification scheme in a reasonable way as explained in the evaluation design to avoid a positive or negative effect on the metrics by a flawed case study.

External validity ensures that results can be gen-

Table 3: Architectural compliance analysis results (mistakes set in italic).

Mistake	Entry Level System Call	Read Data Types	Written Data Types
MT1	SaveLendingConfirmation	*LendingRequest*	LendingConfirmation
MT2	GetForeignArtwork	*OwnArtwork*	–
MT3	GetArtwork	Artwork, *ForeignArtwork*	–
MT4	CreateLendingContract	–	*LendingContract*

eralized. As Runeson et al. (Runeson et al., 2012, p. 71) state, case study results cannot be generalized in a universal way because no statistically relevant sample has been drawn. However, results can be applied to cases with comparable characteristics. With respect to the evaluation, relevant characteristics are the used modeling languages BPMN and PCM because we did no case studies using other languages. However, both languages are established, so the results apply to various other cases using these languages. We expect that results are applicable to other languages with the characteristics mentioned in Section 5.

Reliability ensures that results do not depend on the conducting researcher. The steps necessary to conduct the evaluation are creating the models, running the analysis and classifying the results. Creating the models is not part of the evaluation design, so we ship them and the analysis code as part of a data set (Pilipchuk et al., 2021). In addition, we provide the metric calculation instructions and the classification criteria in the evaluation design. Therefore, we consider the evaluation to be replicable by others. While analyzing the evaluation results, the effects of interpretation by a specific researcher must be eliminated. In order to analyze the accuracy of our approach, we apply established metrics which give a reasonable evidence and reduce the need for interpretation. Due to the study design, there is hardly an interpretation that may lead a researcher to another conclusion.

6.7 Assumptions and Limitations

AcsALign aligns the EAA with ACRs extracted from business processes by identifying violations automatically. In order to do that, we assume that both business processes and EAA are available. As stated before, this assumption is appropriate as bigger organizations are required to create those models e.g. to fulfill regulations or to cope with business complexity. Under this assumption the usage of AcsALign does not impose additional overhead as all input data is available and the analysis is automatic. Moreover, our case study showed that AcsALign is able to identify relevant mistakes in the EAA. We assume business processes to be correct, i.e. they are compliant with regulations and laws, and that modeled data processing matches intended system behavior. We also do not address compliance of implementation and EAA. Certainly, AcsALign is limited to identify what is modeled. If there are only few business processes, we can only discover few ACRs automatically. In that case, manual effort is required to complete ACRs or to model more business processes. The same restriction applies to the architectural analysis as we can only detect violations if the modeled data processing completely reflects the intended behavior.

In general, AcsALign is limited to data types rather than classes of data. Compared to data types, a class of data can have the same type but different ACRs. For instance, a data type *report* might be confidential in case of financial information or might be public in case of information about a certification. The architectural analysis framework provides means for expressing such analyses but BPMN does not. This would require an extension that we explicitly did not want to use in AcsALign to maintain broad applicability and low overhead. However, if considering this limitation at the beginning of business process design, the limitation can be overcome by explicitly modeling classes of data as distinct data objects.

7 CONCLUSION

In this paper, we elaborated on three challenges to align EAAs with business process ACRs and presented solutions for the challenges that we found during the development of an approach called AcsALign that ensures that EAAs adhere to ACRs of business processes. AcsALign consists of the ACRs extraction from business processes, of a data flow analysis to identify read/written data and of a matching algorithm to detect ACR violations. We realized the approach for business processes modeled in BPMN and EAAs modeled in PCM. In the evaluation we evaluated our solutions to the proposed challenges as well as the AcsALign. The evaluation with a real-world case study shows that AcsALign provides satisfying accuracy. AcsALign provides benefits for practitioners and scientists. Practitioners can use the approach to align EAAs at design time and during evolution scenarios where mistakes are done often. Because we use standard modeling languages, the overhead for in-

tegrating the approach is low. The analysis provides valuable insights in the minimum set of required access rights or the data a role has access to. Otherwise, gathering this information can be cumbersome and error-prone. Researchers can use the automated analyses as benchmarks for their approaches or even semi-automated processes. Evaluating more elaborated extraction or matching concepts is possible as the modular tool chain allows replacing individual parts easily.

In our future work, we plan to evaluate more elaborated matching algorithms between data and requirements that consider heavily refined data models in EAAs and to research how versions of EAAs affect business processes in terms of ACRs.

ACKNOWLEDGEMENTS

The DFG (German Research Foundation) – project number 432576552, HE8596/1-1 (FluidTrust) and the KASTEL institutional funding supported this work.

REFERENCES

Abramov, J., Anson, O., Dahan, M., Shoval, P., and Sturm, A. (2012). A methodology for integrating access control policies within database development. *Comput. Secur.*, 31(3):299–314.

Ahn, G. and Hu, H. (2007). Towards realizing a formal RBAC model in real systems. In *SACMAT'07*, pages 215–224. ACM.

Alghathbar, K. and Wijesekera, D. (2003). Authuml: A three-phased framework to analyze access control specifications in use cases. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*, FMSE '03, page 77–86. ACM.

Alpers, S., Pilipchuk, R., Oberweis, A., and Reussner, R. (2019). The current state of the holistic privacy and security modelling approach in business process and software architecture modelling. *Information Systems Security and Privacy*, pages 109–124.

AXELOS (2011). ITIL Edition 2011.

Boehm, B. and Basili, V. R. (2001). Software Defect Reduction Top 10 List. *Computer*, 34(1):135–137.

Busch, M. et al. (2014). Modeling security features of web applications. In *Eng. Secure Future Internet Services and Sys.*, LNCS, pages 119–139. Springer.

D. Brucker, A. et al. (2012). SecureBPMN: Modeling and enforcing access control requirements in business processes. In *SACMAT'12*.

European Parliament (2016). Regulation (EU) 2016/679. *Official Journal of the European Union*, 59:1–88.

Georg, G. et al. (2009). An aspect-oriented methodology for designing secure applications. *Information and Software Technology*, 51(5):846–864.

Giaglis, G. M. (2001). A taxonomy of business process modeling and information systems modeling techniques. *Int J Flex Manuf Syst*, 13(2):209–228.

Goudalo, W. and Seret, D. (2008). Toward the engineering of security of information systems (esis): Uml and the is confidentiality. In *SECUREWARE*, pages 248–256.

Heinrich, R. et al. (2017). Integrating business process simulation and information system simulation for performance prediction. *SoSyM*, pages 1–21.

Heinrich, R., Rostami, K., and Reussner, R. (2016). The CoCoME platform for collaborative empirical research on information system evolution. Technical Report 2016,2; Karlsruhe Reports in Informatics, Karlsruhe Institute of Technology.

Jürjens, J. (2005). *Secure systems development with UML*. Springer.

Kotusev, S. (2017). Critical questions in enterprise architecture research. *IJEIS*, 13(2):50–62.

Lodderstedt, T. et al. (2002). SecureUML: A uml-based modeling language for model-driven security. In *UML'02*, pages 426–441.

Löhe, J. et al. (2014). Overcoming implementation challenges in enterprise architecture management: a design theory for architecture-driven it management (adrima). *ISeB*, 12(1):101–137.

Nguyen, P. H. et al. (2015). An extensive systematic review on the Model-Driven Development of secure systems. *IST*, 68:62–81.

Object Management Group (2011). Business process model and notation (BPMN) v2.0.2.

Open Group (2018). Togaf standard, version 9.2.

Pilipchuk, R., Seifermann, S., Heinrich, R., and Reussner, R. (2021). Evaluation data set. https://doi.org/10.5281/zenodo.4700594.

R. Basili, V., Caldiera, G., and Rombach, D. (1994). The goal question metric approach. *Encyclopedia of Software Engineering*, 1.

Ramadan, Q. et al. (2017). From secure business process modeling to design-level security verification. In *IEEE MODELS*, pages 123–133.

Reussner, R. H. et al. (2016). *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press.

Rodríguez, A., Fernández-Medina, E., and Piattini, M. (2007). A bpmn extension for the modeling of security requirements in business processes. *IEICE - Trans. Inf. Syst.*, E90-D(4):745–752.

Runeson, P. et al. (2012). *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, Inc.

Salnitri, M. et al. (2015). From secure business process models to secure artifact-centric specifications. In *Enterprise, Business-Process and Information Systems Modeling*, pages 246–262. Springer.

Seifermann, S., Heinrich, R., and Reussner, R. H. (2019). Data-driven software architecture for analyzing confidentiality. In *ICSA'19*, pages 1–10. IEEE.

Urbaczewski, L. et al. (2006). A comparison of enterprise architecture frameworks. *IIS*, 7(2):18–23.