From Problems to Laws in Requirements Engineering
Using Model-Transformation

Stephan Faßbender and Maritta Heisel
University of Duisburg-Essen, Paluno - The Ruhr Institute for Software Technology, Oststrasse 99, Duisburg, Germany

Keywords: Compliance, Law, Voting System, Requirements Engineering, Model Transformation.

Abstract: Nowadays, many legislators decided to enact different laws, which all enforce legal and natural persons to deal more carefully with IT systems. Hence, there is a need for techniques to identify and analyze laws, which are relevant for an IT system. But identifying relevant compliance regulations for an IT system and aligning it to be compliant to these regulations is a challenging task. In earlier works of ours we proposed patterns and a structured method to tackle these problems. One of the central crucial steps, while using the patterns and the method, is the transformation of requirements into a structure, allowing the identification of laws. The step is not trivial, as requirements, in most cases, focus on the technical parts of the problem, putting the knowledge about the environment of the system aside. In this work, we propose a method to structure the requirements, elicit the needed domain knowledge and transform requirements into law identification pattern instances. For this purpose, we make use of problem diagrams, problem frames, domain knowledge, and questionnaire. We present our method using a voting system as an example, which was obtained from the ModIWa DFG[1] project and the common criteria profile for voting systems.

1 INTRODUCTION

In general, every legislator demands from everyone, who lives in or is active within the jurisdiction of the legislators, to comply to the laws the legislator enacts. Hence, software engineers have to assure that the system to be developed is compliant to all relevant laws. Therefore they need to know the legal requirements for the system to be developed. Based on this knowledge the engineers can decide whether and how to address compliance.

The identification and analysis of relevant laws is considered to be difficult, because it is a cross-disciplinary task in laws, and software and systems engineering (Biagioli et al., 1987). Otto and Antón (Otto and Antón, 2007) conclude in their survey about research on laws in requirements engineering that there is a need for techniques to identify relevant laws based on requirements, analyze them, and to derive requirements from them.

Pattern-based approaches capture the knowledge of domain experts. In this way, the knowledge is made explicit and can be re-used for recurring problems. Hence, we proposed a pattern-based approach for identifying and analyzing laws in our earlier work (Beckers et al., 2012a). These patterns already allow the identification of relevant laws.

However, the identification of a relevant law alone is not sufficient for software engineers. They require a structured method that uses this approach to derive software requirements and further implementable software specifications. In (Beckers et al., 2012b) we present such a structured method. One crucial and undescribed step within this method is the transformation of functional requirements into law identification pattern instances, which allows the matching with law pattern instances.

In the following, we present a guided transformation of requirements into law identification pattern instances using a voting system as an example. We make use of the problem-based requirements engineering approach proposed by Jackson (Jackson, 2001) to structure the requirements in terms of problem diagrams in the first place. We decided to use problem frames, because they have a kind of semi formal structure and can be modeled. Furthermore, they already embody descriptions of common problems. Thus, they are suitable as input for a transformation as they have a predictable structure and transforma-

[1]Juristisch-informatische Modellierung von Internetwahlen (II). A Deutsche Forschungsgemeinschaft project: http://cms.uni-kassel.de/unicms/index.php?id=38536

tion rules can be setup on basis of the generic problems. Then we show how to turn these problem diagrams into law identification patterns. We provide detailed transformation rules for different requirement patterns, as described by (Côté et al., 2008), to get the corresponding law identification pattern instances. All these information needed for matching, relating, and transformation is provided by means of *transformation cards*. The transformation cards support and guide the requirements engineer, when preparing the requirements for the matching with relevant laws. In this way, the identification of laws gains precision and is less error prone, for example due to forgetting important domain knowledge. Besides improving the precision and lowering the chance of an error, transformation cards are a step towards a semi-automatic tool-support for requirements engineers.

Note, that at the actual point model transformation, as we use the term, is different from the understanding of the model transformation community. The transformation is not an automated transformation between an input and an output model. For using transformations card an input model is required and the transformation card specifies an output model, but does not define a overall automated transformation process. Nevertheless, main parts are candidates to be automated in the future.

In Sect. 2 we introduce the problem frame terminology and notation, the pattern for law identification, and the case study. For our case study, we structure the problem, ending up with a set of problem diagrams in Sect. 3. In Sect. 4 we present a structured method, which guides requirements engineers through the process of transforming the problem into relevant laws. Sect. 5 outlines the result of a validation. Sect. 6 discusses the related work, and Sect. 7 concludes the paper.

2 BACKGROUND

We use the problem frames (Jackson, 2001) approach to structure functional requirements and corresponding domain knowledge. And we make use of the problem frames to facilitate the transformation to the law identification pattern.

Problem Frames. Jackson (Jackson, 2001) introduced the concept of *problem frames*, which is concerned with describing, analyzing, and structuring of software development problems. A problem frame represents a class of software development problems. It is described by a *frame diagram*, which consists of domains, interfaces between them, and a requirement. Domains describe entities in the environment. Jackson distinguishes the domain types **biddable domains** that are usually people, **causal domains** that comply with some physical laws, and **lexical domains** that are data representations. To describe the problem context, a **connection domain** between two other domains may be necessary. Connection domains establish a connection between other domains by means of technical devices. Examples are video cameras, sensors, or networks. Finally, we introduced **display domains** (Côté et al., 2008), which serve to display information to some biddable domain.

Interfaces connect domains, and they contain *shared phenomena*. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain, as indicated by the name of that domain and "!". For example, shared phenomenon *login* in Fig. 1 is observable by the domains **VoterIdentificationClient** and **Voter**, but controlled only by the domain **Voter**. We describe problem frames using UML class diagrams, extended by stereotypes, as proposed by Hatebur and Heisel (Hatebur and Heisel, 2010).

The objective is to construct a machine (i.e., software) that controls the behavior of the environment (in which it is integrated) in accordance with the requirements. When we state a requirement, we want to change something in the environment. Therefore, each requirement *constrains* at least one domain. A requirement may *refer* to several domains in the environment of the machine.

The problem frames approach distinguishes therefore between the *requirements (R)*, the *domain knowledge (D)*, and the *specification (S)*. The requirements describe the desired system after the machine is built. The domain knowledge represents the relevant parts of the problem world. The specifications describe the behavior of the software in order to meet the requirements.

Beside the requirements, the domain knowledge about the environment of the machine to be built is crucial for understanding the problem and specifying the machine behavior later on. Unlike Jackson, we distinguish between *assumptions*, facts, and definitions and designations. Assumptions describe conditions fulfilled by the environment that are needed, so that the requirements can be fulfilled by the machine. But these can be violated at a certain point. Facts describe fixed properties of the environment irrespective of how the machine is built. Definitions and designations specify a set of special terms used for formulation requirements, assumptions and facts.

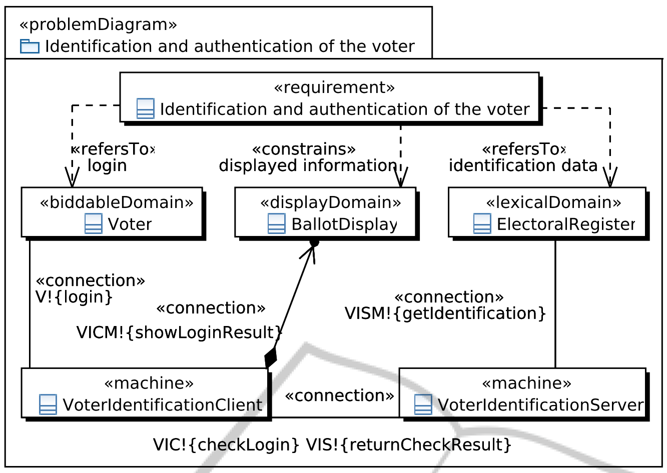


Figure 1: Problem Diagram for `R 1`.

Problem-oriented requirements analysis starts with representing the environment using a *context diagram*. Such a diagram describes *where* the problem is located by stating the relevant domains and their interfaces, including the machine to be built. Problem-oriented requirements analysis proceeds with a decomposition of the overall problem into sub-problems, which are represented by *problem diagrams*. The problem diagrams should be instances of problem frames, thereby representing *simple* development problems. In contrast to context diagrams, problem diagrams contain the requirements belonging to the sub-problem.An exampleof a problem diagram can be found in Fig. 1.

Figure 1 shows a problem diagram in UML notation. The biddable domain (UML class with stereotype ≪biddableDomain≫) **Voter** controls the *login* command (Name of the UML association with the stereotype ≪connection≫ between the classes *Voter* and *VoterIdentification-Client*), which is observed by the machine domain **VoterIdentificationClient**(UML class with stereotype ≪machine≫). The **VoterIdentificationServer-Machine** observes the phenomenon *check the login*, which is controlled by the **VoterIdentification-Client**. The **VoterIdentificationServer** controls the phenomenon *return check result*, which is observed by the **VoterIdentificationClient**, and the *get identification data*, which is obtained from the **Electoral-Register**. The **VoterIdentificationClientMachine** *shows the login result* using the **BallotDisplay**. The requirement `R 1` (for a textual description see Sect. 3) constrains the **BallotDisplay** and refers to the **Voters**, and the **ElectoralRegister**.

Patterns for Requirement-Based Law Identification. Commonly, laws are not adequately considered during requirements engineering (Otto and Antón, 2007). Therefore, they are not covered in the subsequent system development phases. One fundamental reason for this is that involved engineers are typically not cross-disciplinary experts in law and software and systems engineering. To bridge this gap we developed law patterns and a general process for law identification which relies on these patterns.

We investigated how judges and lawyers are supposed to analyze a law, based upon legal literature research. These insights lead to a basic structure of laws and the contained sections. One result of our investigations is a common structure of laws. Based on this structure of laws, we defined a *law pattern*. The law pattern itself is discussed in detail in (Beckers et al., 2012a).

Identifying relevant laws based on functional requirements is challenging. Functional requirements are often too imprecise for a sufficient law analysis, they contain important information only implicitly, and use a different wording than in laws. Therefore, we developed a *law identification pattern*, which structures requirements in a way that important terms of a requirement can be mapped to the legal wording and then matched with law pattern instances (Fig. 2 shows an instance).

First of all, a Requirement can be Related To other Requirements and dictates a certain behavior of the machine. A behavior can be a certain Activity or an entired Process. The machine resides in one or more jurisdictions represented by their Legislators. And the problem the machine has to solve is a problem of one or more Domains. The fundamental parts of the requirements are described in the Core Structure. An Activity involves

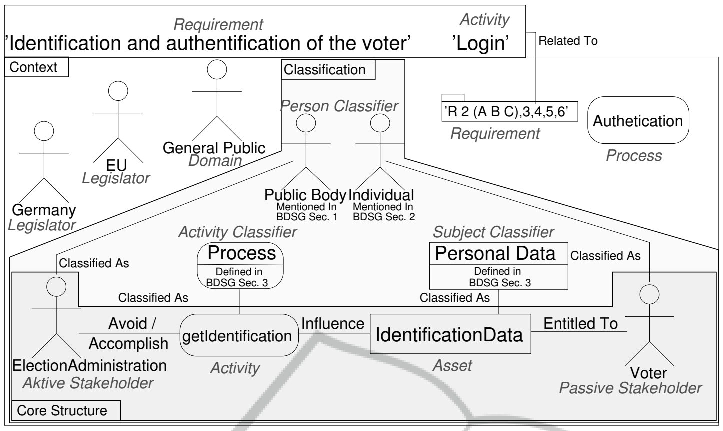


Figure 2: Law Identification Pattern Instance R 1.

an Active stakeholder and in some cases an Asset. Additionally, an Activity Influences a Passive Stakeholder in a direct way or indirectly through an Asset, to which the Passive Stakeholder is entitled to. The terms used for Activities, Stakeholders, and Assets can be Classified in the Classification part, using terms of the legal domain.

These two kinds of patterns, the law pattern and the law identification pattern, can be used through their graphic and explicit nature to foster the communication between legal experts and software engineers. Moreover, they allow the matching of requirements and laws, because the Law Structure and the Core Structure of requirements are organized in the same way. The language gap between the legal and IT domain is bridged using the Classification part of both patterns.

The procedure for identifying relevant laws consists of five steps. The first step is to set up a database of all laws which might be of relevance for a scenario. Therefore, laws have to be analyzed and stored in the structure of the law pattern. Thus, they are stored as pattern instances. The second step uses information from functional requirements and their context to instantiate the core structure and the context of the law identification pattern. Third, the relation between laws and software requirements has to be established to prepare the identification of relevant laws for the given system. Hence, a mapping between the terms and notions of the software requirements to legal terms and notions is derived. Fourth, the law pattern instances and law identification pattern instances have to be matched. This results in a set of laws which might be of relevance for the software. Fifth, the found laws are the basis for further investigations.

In order to accomplish the process described in (Beckers et al., 2012b), law experts and software engineers have to work together for the necessary knowledge transfer. Step one can be done alone by legal experts and for step two only software engineers are needed. But in step three and four both groups are needed to bridge the gap between legal and technical world. The last step can be accomplished alone by legal experts. In this paper, we focus on the transformation step and how to turn requirements, in our case in terms of problem diagrams, into law identification pattern instances.

Relevant Laws for Voting Systems. In our case study the German law is the binding law. For simplicities sake, we focus on relevant compliance regulations for privacy. We only explain the laws and regulations in detail that we use in the example. In 1995, the European Union (EU) adopted the *Directive 95/46/EC* on the processing of personal data. Germany implements the European Privacy Directive in the *Federal Data Protection Act (BDSG)*. According to *Section 1 BDSG* all private and public bodies that automatically process, store, and use personal data have to comply with the BDSG.

Example: Voting System for Germany. By its very nature, the field of electronic voting is an interdisciplinary field, where legal and computer scientists work together. During the development of the first voting system used in Germany, this fact was neglected or inadequately considered. Hence, the federal constitutional court of Germany judged in 2009, that using this system for votings in 2005 was unconstitutional (Federal Constitutional Court of Germany, 2009).

A general problem description of the voting system and which functionality it has to provide was derived from (Brehm, 2012), and (Volkamer, 2009). The first work was conducted in the context of the ModIWa II project, while the last work was elaborated in the context of a Common Criteria (CC) Profile (Volkamer and Vogt, 2008) for online voting.

3 STRUCTURING THE PROBLEM

As proposed by Jackson (Jackson, 2001), we derive requirements and domain knowledge from the problem description, structure the overall problem using a context diagram, and decompose the overall problem into sub-problems using problem diagrams. Note, that for presenting our transformation method we only use functional requirements. Non-functional requirements are left aside. Nevertheless, the found laws demand further quality requirements, for example privacy requirements.

Requirements and Domain Knowledge. The CC profile for online voting systems (Volkamer and Vogt, 2008) only deals with the polling phase. The preliminary election preparation and the tallying are not considered in detail. Hence, the profile only defines functional requirements for the voter and the election officer, who represents the election authority. In total, the machine to be built is described in terms of 21 requirements by the CC profile. Later on we split some of the requirements for handling reasons. We only enlist one requirement of the voter at this point. It is sufficient for the rest of the paper. The requirement texts are directly taken from the CC profile (Volkamer and Vogt, 2008). (Note that the abbreviation TOE stands for target of evaluation, which is, in terms of the CC, the machine to be built)

(`R 1`) identification and authentication of the voter. Only registered voters are permitted to cast a vote. The voter [...] identifies and authenticates himself to the server-sided TOE. The server-sided TOE checks the registered voter's right to vote. [...] The registered voter receives applicable acknowledgment of the acceptance or refusal [...].[...]usually in the form of a corresponding on-screen display. [...] The voter has been identified and authenticated at the latest when the vote is cast.

Besides the requirements themselves, the knowledge about the environment of the machine to be built is crucial for understanding the problem and specifying the machine behavior later on. We only present the assumptions and facts we derived from CC profile (Volkamer and Vogt, 2008), which are of relevance in the following. In total, the CC profile states 21 assumptions, 14 facts and 35 definitions and designations.

(`A 3`) ballot display. Vote casting [...] takes place [...] from a vote-casting device which is able to display the full contents of the ballot and to implement the responsible election authority's specifications for the type of display, in particular the order of voting options.[...] The voter acts responsibly in securing the vote-casting device. It is assumed that each voter that installs or uses the client-sided TOE does so in such a way that the vote-casting device can neither observe nor influence the vote casting process. This includes the assumption that the voter does not manipulate his vote-casting device on purpose. The vote-casting device is able to properly display the ballot, to properly transfer the voters input to the election server and to delete the vote after the polling process.

(`F 3`) user relation to identification data. Secondly, the user-related data in the electoral register with which a registered voter can uniquely identify himself.

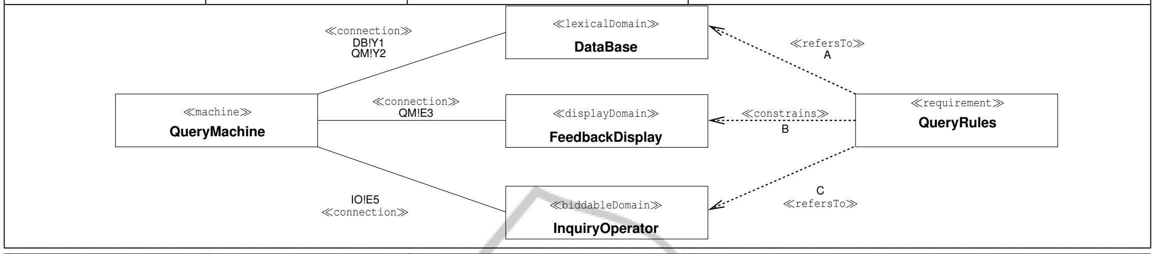
(`DD 8`) identification data. Firstly, a measure to be used by every registered voter to identify himself on the TOE. This can be, for example, a membership number, a name, a data of birth or an address.

For analyzing and structuring requirements we use problem diagrams. Figure 1 shows the problem diagram for the requirement `R 1`. This requirement demands that **Voters** can authenticate and identify against the machine. The diagram was already described in Sect. 2.

4 TRANSFORMING PROBLEM DIAGRAMS TO LAW IDENTIFICATION PATTERN INSTANCES

So called transformation cards are the central tool for executing the transformation in the following. We developed a transformation card for each problem frame, which helps requirements engineers to fit the problem diagrams into the according law identification pattern instances. The transformation card contains information for *matching problem diagrams and frames*, and information how the problem frame, and therefore the matching problem diagram, is related to the Core Structure of the law identification pattern. It also contains information for *collecting potentially*

Table 1: Query Transformation Card: Identification Part.

Problem Frame: Query			
Referred domain type(s)	**Constrained domain type(s)**	**Comment**	**Sequence of Phenomena**
B, X	D	Query	IO!E5;[QM!Y2;][DB!Y1;]CM!E5
 ≪connection≫ DB!Y1 QM!Y2 ≪lexicalDomain≫ DataBase ≪refersTo≫ A ≪machine≫ QueryMachine ≪connection≫ QM!E3 ≪displayDomain≫ FeedbackDisplay ≪constrains≫ B ≪requirement≫ QueryRules IO!E5 ≪connection≫ ≪biddableDomain≫ InquiryOperator C ≪refersTo≫			
...			

missing domain knowledge, which is important transforming for problem diagrams matching the problem frame at hand, and the *transformation rules* themselves. As a result, the transformation card supports and guides requirements engineers, when preparing the requirements for the matching with relevant laws. In this way, the identification of laws gains precision and is less error prone, for example due to forgetting important domain knowledge. Beside improving the precision and lowering the chance of an error, transformation cards are a step towards a semi-automatic tool-support for the requirements engineer.

Matching Problem Diagrams with Frames. Table 1 shows the part of a transformation card, which can help to identify the matching problem frame. In the problem frame part the structure of the problem frames by means of contained domains, phenomena, their sequence, and *refers, constrains* relations to the requirement are described. A matching problem diagram must have the same characteristics as described in this part of the transformation card.

The problem diagrams have sometimes to be modified for matching. For example, big and complex problem diagrams have to be partitioned or domains have to be merged for fitting the problem diagram at hand to a problem frame. For reasons of space we skip the full discussion. The interested reader is referred to Hatebur and Heisel (Hatebur and Heisel, 2010), which also outlines the possibilities for automated identification of problem frames.

For our example (Fig. 1), we now check if a transformation card is applicable or not. The first thing to be checked is, if the domains which are referred or constrained have the correct type as described in the identification part. The **Voter** is a biddable domain and the the **ElectoralRegister** is a lexical domain. Both are referred to. This matches the *Referred domain type(s)* of the identification part. This is also true for the *Constrained domain type(s)*, which has to be a display domain. The constrained domain **Ballot-Display** in our problem diagram is a display domain. Thus, this transformation card remains an applicable candidate.

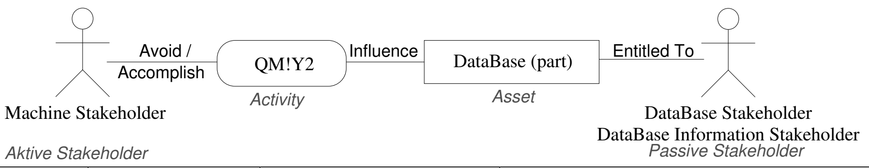
Next, we have to check if the overall structure of the problem diagram matches the problem frame. For the structure, the problem frame, as described in the graphical part of the identification part, is quite similar. The only difference is that we have two machines in Fig. 1. But this is the case because of the distributed nature of our problem. For matching, we can merge those two machine domains. Hence, the structure is the same.

Last, we have to check whether the implicitly described concern of the problem frame, by means of the phenomena, matches the phenomena of the problem diagram or not. The sequence of *V!*{*login*}; *VISM!*{*getIdentifictaion*}; *VICM!*{*showLoginResult*} matches the regular expression as given by the *Sequence of Phenomena* in the transformation card. As a result, the transformation card for the problem frame query has to be applied.

Collecting Domain Knowledge. After the successful matching, the transformation card contains further guidance for preparing the transformation. It contains several core structure variants, which describe the possible Core Structure instantiations of the law identification pattern for the problem frame at hand.

The core structure variants not only relate problem frame and Core Structure, but also consider typical domain knowledge for a problem frame. To ensure that this domain knowledge is collected properly, there is a questionnaire for each core structure variant. The questionnaire is structured into the parts "Necessary Information", which describes for which information we are looking for, "Details", which states which domain is the target of the question, the "Question" it-

Table 2: Query Transformation Card: Transformation Part.

Problem Frame: Query			
...			
Core Structure Variant 2			
 Avoid / Accomplish QM!Y2 Activity Influence DataBase (part) Asset Entitled To Machine Stakeholder Aktive Stakeholder DataBase Stakeholder DataBase Information Stakeholder Passive Stakeholder			
Necessary Information	**Details**	**Question**	**Result**
Structure Database	DataBase	Which information is contained in the database and which structure does it have?	**For each** new found part of the database **:** **Add** a ≪lexical≫ domain for the part to the model. **Add** a aggregation relation between database and database part domain to the model.**roF**
Database Stakeholder	-	Who are the stakeholders owning the database?	**For each** found stakeholder **:** **If** stakeholder does not exists in model **then Add** new ≪biddable≫ domain for the stakeholder to the model.**fI** **Add** a ≪entitledTo≫ association between database (part) domain and stakeholder domain.**roF**
Database Information Stakeholder	-	Who are the stakeholders entitled to information contained in the database (part)?	**For each** found stakeholder **:** **If** stakeholder does not exists in model **then Add** new ≪biddable≫ domain for the stakeholder to the model.**fI** **Add** a ≪entitledTo≫ association between database (part) domain and stakeholder domain.**roF**
Machine Stakeholder	-	Who is responsible and in control of the machine?	**For each** found stakeholder **:** **If** stakeholder does not exists in model **then Add** new ≪biddable≫ domain for the stakeholder to the model.**fI** **Add** a ≪controls≫ association between stakeholder domain and machine.**roF**
Instantiation Rule:		**For each** database and its parts: **For each** machine stakeholder: **For each** database (information) stakeholder: Instantiate core structure variant 2. **roF roF roF**	
...			

self, and "Result", which describes how to model the collected domain knowledge. While answering these questions, necessary domain knowledge, which might be missing, is collected.

Tab. 2 shows one core structure variant. There are three more variants, which we have to skip due to reasons of space. The first information, which might be missing, is about the structure of the **DataBase**. Normally, in problem diagrams a lexical domain represents different information, which can be seen as parts of the overall database. But for law identification we need to know the specific piece(s) of information, which are relevant for the problem at hand. In case the database can be partitioned further, we have to add a separate lexical domain for each found piece of information. For the **ElectoralRegister** we already collected the information by means of the fact `F 3` and the definition `DD 8`. Hence, we know that identification data is part of the electoral register. Furthermore, we know from the problem diagram that this information is referred to. Thus, we add the identification data as part of the electoral register. This database has one **DataBase Stakeholder**. It is the **ElectionAuthority**, who owns the database. It is the **Voter** about whom information is stored, because he own. The database has one **DataBase Information Stakeholder**, to which the next question refers. It is the **Voter** about whom information is stored, because he own. As this stakeholder is already modeled as a biddable domain, we do not have to change the model. For the question about the database stakeholder and the last question we have to add the biddable domain **ElectionAuthority**, because it is in control of the machine, but not part of the model yet. Finally, we have all information in place to start the transformation from problem diagram to the core structure of the law identification pattern.

Executing the Transformation. With the newly obtained domain knowledge at hand, we can transform the problem diagram into a law identification pattern core structure instances. How often a core structure has to be instantiated is described in the instantiation rule (see Tab. 2) of each core structure variant part of the transformation card. Hence, it is possible that one problem diagram is transformed into several law identification pattern instances.

For `R 1` (Fig. 1) and *Core Structure Variant 2* (Tab. 2) we have to instantiate the core structure for each combination of **Database (part)**, **Machine Stakeholder**, **DataBase Stakeholder** and **DataBase**

Information Stakeholder. The *Activity* is the same for all core structures of variant 2. This results in a core structure for **ElectionAuthority**, *getIdentification*, **ElectoralRegister**, and **Voter**, a core structure for **ElectionAuthority**, *getIdentification*, **ElectoralRegister**, and **ElectionAuthority**, a core structure for **ElectionAuthority**, *getIdentification*, **IdentificationData**, and **ElectionAuthority** and a core structure for **ElectionAuthority**, *getIdentification*, **IndentificationData**, and **Voter**.

After adding the context information to the pattern and discussing with a legal expert the mappings of terms, we get a law identification pattern instance as shown in Fig. 2. For the process of adding the classification and context and finally the matching to laws, we refer the interested reader to (Beckers et al., 2012a; Beckers et al., 2012b).

5 VALIDATION

For the validation of our proposed method we analyzed the voting system in a case of action research. We modeled and used the transformation cards for all requirements regarding the voter. We matched the resulting law identification pattern with selected laws of Germany. These laws were modeled in terms of law patterns for the case study or were already modeled for previous case studies. There were two theses to be tested, namely *"The transformation cards are sufficient to integrate in the overall identification process as described by (Beckers et al., 2012b)."* and *"Using the transformation cards leads to an identification of all relevant laws."*.

To answer the first question, we tested the usage of transformation cards by conducting it in the overall process as described in (Beckers et al., 2012b). To be able to discuss the sufficiency of the transformation cards, we tracked the generation of core structures in terms of number and effort. We also documented our experiences while conducting the method. The result is more of a qualitative nature than quantitative.

To answer the second question, we conducted a literature research about the voting system and the relevant German laws for this matter. The main source was the judgment of 2009 by the Federal Constitutional Court of Germany (Federal Constitutional Court of Germany, 2009), followed by discussions with several domain and / or legal experts. These insights lead to expectations whether a requirement will match with a particular law or not. These expectations were documented in terms of a table listing the expectations for each law and requirement. This table was compared to the matching based on the generated law

Table 3: Validation results.

			Core Structures		**BDSG**		**BwahlG**		**SigG**		**PassG**	
Requirement	**Name**	**Problem Frame**	**Generated**	**Redundant**	**Matched**	**Relevant**	**Matched**	**Relevant**	**Matched**	**Relevant**	**Matched**	**Relevant**
R1	Identification and authentication of the voter	Query	16	8	8(4)	2(2)	0(0)	0(0)	16(8)	0(0)	0(0)	0(0)
R2A	Show the ballot	Query	20	16	12(2)	4(1)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
R2B	Completion of the ballot	Simple Workpiece	12	7	8(3)	2(1)	4(2)	4(2)	0(0)	0(0)	0(0)	0(0)
R2C	Correction of the ballot	Simple Workpiece	12	7	8(3)	2(1)	4(2)	4(2)	0(0)	0(0)	0(0)	0(0)
R3	Initiation of vote casting	Commanded Transformation	20	6	5(3)	5(3)	12(7)	8(4)	0(0)	0(0)	0(0)	0(0)
R4A	Hasty voting protection	Query	24	14	16(6)	4(2)	5(4)	4(3)	0(0)	0(0)	0(0)	0(0)
R4B	Cast confirmed vote	Commanded Transformation	8	4	6(2)	1(1)	6(2)	6(2)	0(0)	0(0)	0(0)	0(0)
R4C	Acknowledgment of vote	Model Display	12	4	3(2)	1(1)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
R4D	Delete completed ballot	Simple Transformation	11	6	7(3)	2(1)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
R5	Abort of the polling process	Commanded Display	6	4	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
Sum			**141**	**76**	**73(28)**	**23(13)**	**31(17)**	**26(13)**	**16(8)**	**0(0)**	**0(0)**	**0(0)**

identification pattern. The result is of a quantitative nature in terms of false positives and false negatives, and evidence by the number of matches.

For the validation, we excluded some laws even though they were identified as relevant based on the literature research and discussions. We selected the highly relevant laws as discussed by the Federal Constitutional Court. These laws are necessary to find the weaknesses of our method in terms of false negatives. A false negative would be missing match with a law for a certain requirement. To identify false positives, we also added laws which are somehow related to voting systems, but not relevant. Hence, here we expected to find matches, which are not of real relevance. For the validation, we selected the four following laws:

- The BDSG as highly relevant law concerning personal data.
- The BWahlG (Bundeswahlgesetz), which is the law for federal state elections in Germany and also highly relevant.
- The SigG (Signatur Gesetz), a law which regulates the use of digital signatures. This particular law was selected not due to its relevance, but due to the matter that it is related to our case study, e.g. in terms of technological background.
- The PassG (Pass Gesetz), which regulates the use of passports in Germany. This law is clearly irrelevant, nevertheless the passport is a possible authentication means during elections.

Table 3 shows the results of using the transformation cards. In the first two columns, the requirement and its name are listed. In the next column the identified problem frame, for the requirement at hand, is listed. The next column states how many core structures were generated by executing the transformation described by the transformation card for the problem frame. While executing the transformation it turned out that the transformation rules generate a noticeable number of duplicated core structures. Hence, we added this information in the next column. Next, the four laws are listed. For each we tracked how often it was matched by a law identification pattern and whether this match was relevant or not. The number before the parentheses states the number considering all law identification patterns, which include the duplicates. The number within the parentheses states number without the duplicates. This table allows some reasoning about the transformation cards.

From our observations, the transformation cards integrated well in the overall law identification process. After the core structures were generated, no further modifications were required for instantiating the full law identification patterns. The preparations of the core structures were slightly more structured and therefore less error prone compared to the hands on instantiation we used previously.

A major downside is the pure number of generated core structures. Generating the core structure takes a significant amount of time and afterward not every core structure is necessary for a successful matching. The high share of duplicates worsens the situation even worse as they devalue more than half of the work (76 duplicates out of 141 as shown in Tab. 3). The generation of duplicates should be reduced in the future by detecting them early on. For the non duplicated ones the effort to be spent can be reduced in terms of tool support. From our experience, none of the non duplicated ones can be removed as they are all necessary for a detailed detection of relevant laws. Sometimes only the minor differences between core structures revealed the most important parts of a requirement for the relevance of a law. These differences then help to understand how to address the law. Furthermore, we could not identify common characteristics for filtering out the core structures, which match all relevant laws. The laws are too different to do so. Hence, the big number of core structures is necessary and even helpful, but regarding the effort to generate and analyze them, some further research is needed to minimize the effort.

Speaking of the effort, there are several things to consider. The matching of problem diagrams with problem frames is straight forward and takes no time. The modeling of the problem diagrams by an experienced user took about 5 person hours (0.5 hour for one requirement). This seems reasonable if the law identification gives sufficient results afterward. Additionally, one can benefit of setting up the diagrams as their use is not limit to law identification alone. Answering the questionnaire and modeling the resulting information takes some more time. But this process of answering and modeling speeds up significantly with rising number of already analyzed requirements. Mostly, the questions consider the domains directly. And as the number of domains is limited, so is the information needed about them. Thus, most information is already known and modeled for later requirements. Including discussions it took about 10 person hours for this step (1 hour for one requirement). This seems to be a significant amount of work, but this information collected is crucial for the success of the application of the transformation card. Hence, we spent some more time for this discussions and searching for the necessary information (e.g. in the protection profile (Volkamer and Vogt, 2008)). Setting up the core structures is an easy task, but also time con-

Table 4: Validation Results: Expected and found matches.

		Requirement									
		R1 identification voter	R2 A show ballot	R2 B complete ballot	R2 C correct ballot	R3 init vote	R4 A hasty vote protection	R4 B cast vote	R4 C ack vote	R4 D clear buffer	R5 abort vote
Law	BDSG	X	**X**	**X**	**X**	X	**X**	X	X	X	
	BwahlG		~~X~~	X	X	X	X	~~X~~	~~X~~		
	SigG	$\underline{X}$									
	PassG										

suming. It took 8 person hours to generate all core structures by hand. Which is about 4 minutes per core structure. In total, executing the proposed method took 23 person hours, which makes it about 4 person days.

This effort seems to be significantly high. But there are two things to consider: First, the time for generating the core structures can be reduced to almost zero by tool support. Second, the elicitation of law relevant information has to be invested even if someone uses another method to find relevant laws. Thus, the effort related to the use of transformation cards is limited to the modeling of problem diagrams and modeling of the collected information. And this pays as it enables the automated generation and guided analysis.

Some additional effort not unique to the transformation cards has to be considered regarding the overall process, starting with modeling the laws and ending with a legal revision of the matches. The modeling of the laws is quite time consuming as many discussions are required and therefore should not be taken out by one person alone. The time spent for a particular law depends on the size and complexity of the law. From our experience, it can take up to an hour for one particular section of a law. But every law has to be modeled only once and is reusable afterward. Discussing the matches is not that time consuming as the law modeling. The understanding of the system analyzed is already established and all relevant information is known. Thus, to take a decision takes less time. And using the transformation cards support in the decision taking, as a high share of found matches is also relevant (e.g 13 relevant out of 17 matches for the BWahlG as shown in Tab. 3). Hence, the overall effort to be spent seems to be reasonable, as long as the results of the identification process are precise enough.

From our experience, the transformation card method is sufficient to integrate in the overall identification process as described by (Beckers et al., 2012b).

For judging the precision and recall of the law identification using transformation cards, we setup Tab. 4. Symbols should be read the following way:

- A normal cross indicates an expected and observed match.
- A bold cross indicates an unexpected but correct match.
- A canceled cross indicates an expected but not observed match and the missing cross turned out to be correct.
- An underlined cross indicates an unexpected and irrelevant match.

For the precision the identification turned out to be remarkable. The precision for the voting system and the four selected laws is at 0.94 (true positive / (true positive + false positive) = 16 / (16+1)). Thus, almost every match points out a relevant law. This is not true for a single core structure as one can see from Tab. 3. But whenever there are matches for core structures regarding one requirement, at least one proves to be relevant.

For the recall the result is perfect. The recall is at 1 (true positive / (true positive+false negative) = 16 / (16 + 0)). Thus, not a single relevant law is missed.

Having a high recall is more important than a high precision in our case. The method should find all relevant laws. The impact of a missed law is much more serious to the development and success of the system-to-be, than the extra effort spent on the legal revision for an irrelevant law. Hence, having recall value near to 1.0 is the main objective of our method. But a precision of 0.94 adds some surplus value. In terms of precision and recall, it is reasonable to use our method.

Compared to our expectations based on the requirements and legal insights alone, using the identification method is superior. For the BDSG we neglected the fact, that the information about the candidates and their relation to parties is personal information, which falls under the BDSG. In fact, issue is not of high importance as this information is public available. But nevertheless it makes the BDSG relevant for these requirements. For the BWahlG it turned out that this law and its sections only deal with the expression of opinion alone. Thus, those requirements that are not directly related to the voting itself are not in the focus of this particular law. The only point in favor for our prediction is that we rejected the SigG, while it was matched once by the law identification process. Overall, the precision and recall of the law identification process is significantly higher than for our educated prediction.

"To sum up, using the transformation cards leads to an identification of all relevant laws with a high precision.". And it significantly improves the situation compared to an unguided method.

6 RELATED WORK

Breaux et al. (Breaux et al., 2006; Breaux and Antón, 2008) present a framework that covers analyzing the structure of laws using a natural language pattern. This pattern helps to translate laws into a more structured restricted natural language and then into a first-order logic. The idea of using first order logic in the context of regulations is not a new one. For example Bench-Capon et al., (Bench-Capon et al., 1987) made use of first-order logic to model regulations and related matters. In contrast to our work, the authors of those approaches assume that the relevant laws are already known and thus do not support identifying legal texts based on functional requirements.

Siena et al. (Siena et al., 2008) describe the differences between legal concepts and requirements. They model the regulations using an ontology, which is quite similar to the natural language patterns described in the approaches mentioned previously. The ontology is based in the Hohfeld taxonomy (Hohfeld, 1917), which describes the means and relations between the different means of legal texts in a very generic way. Thus Hohfeld does not structure a certain law at all, but he aims at the different meanings of laws. So the resulting process in (Siena et al., 2008) to align legal concepts to requirements and the given concepts are quite high-level and cannot be directly applied to a scenario. In a second work Siena et al. (Siena et al., 2009) try to bridge the gap between the requirements engineering process and compliance using a goal-oriented approach. In this work they propose to derive goals from regulations and apply those goals to the actors within a requirements engineering scenario. In contrast to our approach they do not identify relevant laws and do not intertwine compliance regulations with already elicited requirements.

Maxwell et al. (Maxwell and Antón, 2009) developed an approach to check existing software requirements for regulatory compliance, i.e., to discover violations and missing requirements. While our approach focuses on the identification of relevant laws, we could imagine using it to detect violations, too. We consider dependencies between different laws or regulations, which the approach from Maxwell et al. neglects.

Álvarez et al. (Álvarez et al., 2002) describe reusable legal requirements in natural language, based on the Spanish adaption of the EU directive 95/46/CE concerning personal data protection. We believe that the work by Álvarez et al. complements our work, i.e., applying our law identification method can preceed using their security requirements templates.

7 CONCLUSION

In this work, we introduced a structured method for transforming functional requirements into law identification pattern instances. This enables us to find relevant laws for a software system to be built. The transformation makes use of problem diagrams for structuring the functional requirements, problem frames for transformation instructions, domain knowledge for considering the context of the system, and questionnaires for refining the domain knowledge. We illustrated the method using a case study in the field of online voting. The contributions of this work are:

- Reuse of results of an existing requirements engineering (here problem frames) approach for law identification.
- Transformation cards, which enable software engineers to
  - identify the problem class of the requirement at hand.
  - identify the needed domain knowledge for the transformation.
  - obtain instructions how to model the domain knowledge.
  - execute the transformation.
- A structured and guided method for software engineers to transform functional requirements into law identification pattern instances.
- An improvement of law identification in requirements engineering by augmenting a crucial step of the law identification process as described in Beckers et al (Beckers et al., 2012b).
- The basis for semi-automatic tool support.

We will also publish a technical report containing all transformation cards, the full case study, the resulting law identification pattern instances, and the law pattern instances used.

For the future we plan to investigate the matter of quality requirements. Quality requirements itself are too vague to be directly transformed into law identification pattern instances. But they contain additional, relevant information about the functionality and context of a system. It seems to be promising to integrate this information to improve the precision of our method.

In general, our law identification process was used in the field of cloud computing, health-care and for this paper in the domain of voting systems. The transformation cards were only used for the latter. From our experience, our method is usable regarding the German law for different domains without adaption. We also found evidence that this observation is also true for laws from other countries as long as the law system of the country is a statue law. For example, Biagioli. et al (Biagioli et al., 1987) describe the very same structure for Italian laws like we use for German laws. For case law systems like the one of the US, our method needs to be adapted. The use of our method on more domains, for other countries with statue law, and even for case law countries is under research.

We also plan to formulate rules for the reduction of core structure candidates. At the moment, we get a large set of core structures for some problem frames. This makes the modeling of law identification pattern instances tedious. Hence, we will improve the situation by giving guidance to identify irrelevant core structures or core structures which can be merged.

Finally, we plan to speed up the method execution by providing tool support for identifying problem frames for problem diagrams, wizards and recommenders, which are based on the questionnaires, for collection the domain knowledge, and automatic transformation based on this information.

REFERENCES

Álvarez, J. A. T., Olmos, A., and Piattini, M. (2002). Legal requirements reuse: A critical success factor for requirements quality and personal data protection. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 95–103. IEEE.

Beckers, K., Faßbender, S., Küster, J.-C., and Schmidt, H. (2012a). A pattern-based method for identifying and analyzing laws. In *REFSQ*, pages 256–262.

Beckers, K., Faßbender, S., and Schmidt, H. (2012b). An integrated method for pattern-based elicitation of legal requirements applied to a cloud computing example. In *ARES*, pages 463–472.

Bench-Capon, T. J. M., Robinson, G. O., Routen, T. W., and Sergot, M. J. (1987). Logic programming for large scale applications in law: A formalization of supplementary benefit legislation. In *Proceedings of the International Conference on Artificial Intelligence and Law (ICAIL)*. ACM.

Biagioli, C., Mariani, P., and Tiscornia, D. (1987). Esplex: A rule and conceptual model for representing statutes. In *ICAIL*, pages 240–251. ACM.

Breaux, T. D. and Antón, A. I. (2008). Analyzing regulatory rules for privacy and security requirements. *IEEE Transactions on Software Engineering*, 34(1):5–20.

Breaux, T. D., Vail, M. W., and Antón, A. I. (2006). Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 46–55. IEEE.

Brehm, R. (2012). Kryptographische Verfahren in Internetwahlsystemen. Technical report, Technical University of Darmstadt.

Côté, I., Hatebur, D., Heisel, M., Schmidt, H., and Wentzlaff, I. (2008). A systematic account of problem frames. In *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 749–767. Universitätsverlag Konstanz.

Federal Constitutional Court of Germany (2009). Verwendung von Wahlcomputern bei der Bundestagswahl 2005 verfassungswidrig.

Hatebur, D. and Heisel, M. (2010). Making pattern- and model-based software development more rigorous. In *Proceedings of 12th International Conference on Formal Engineering Methods, ICFEM 2010, Shanghai, China*, LNCS 6447, pages 253–269. Springer.

Hohfeld, W. N. (1917). Fundamental legal conceptions as applied in judicial reasoning. *The Yale Law Journal*, 26(8):710–770.

Jackson, M. (2001). *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley.

Maxwell, J. C. and Antón, A. I. (2009). Developing production rule models to aid in acquiring requirements from legal texts. In *Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE*, RE '09, Washington, DC, USA. IEEE Computer Society.

Otto, P. N. and Antón, A. I. (2007). Addressing legal requirements in requirements engineering. In *Proceedings of the International Conference on Requirements Engineering*. IEEE.

Siena, A., Perini, A., and Susi, A. (2008). From laws to requirements. In *Proceedings of the International Workshop on Requirements Engineering and Law (RELAW)*, pages 6–10. IEEE.

Siena, A., Perini, A., Susi, A., and Mylopoulos, J. (2009). A meta-model for modelling law-compliant requirements. In *Proceedings of the International Workshop on Requirements Engineering and Law (RELAW)*, pages 45–51. IEEE.

Volkamer, M. (2009). *Evaluation of Electronic Voting: Requirements and Evaluation Procedures to Support Responsible Election Authorities*. Springer Publishing Company, 1st edition.

Volkamer, M. and Vogt, R. (2008). *Common Criteria Protection Profile for Basic set of security requirements for Online Voting Products*. Bundesamt f"ur Sicherheit in der Informationstechnik.