WILL GOVERNMENT FORMS EVER BE CONSISTENT? DETECTING VIOLATIONS IN FORM STRUCTURES BY UTILIZING GRAPH THEORY

Steffen Höhenberger
University of Muenster, Muenster, Germany, steffen.hoehenberger@ercis.uni-muenster.de

Hendrik Scholta
University of Muenster, Muenster, Germany, hendrik.scholta@ercis.uni-muenster.de

Follow this and additional works at: http://aisel.aisnet.org/ecis2017_rp

Recommended Citation
WILL GOVERNMENT FORMS EVER BE CONSISTENT? DETECTING VIOLATIONS IN FORM STRUCTURES BY UTILIZING GRAPH THEORY

Research paper

Höhenberger, Steffen, University of Muenster, Muenster, Germany, steffen.hoehenberger@ercis.uni-muenster.de
Scholta, Hendrik, University of Muenster, Muenster, Germany, hendrik.scholta@ercis.uni-muenster.de

Abstract

Forms play an important role in government service delivery since they are the central interface between the government and its citizens. However, due to the multiplicity of forms, their management in governments is complex. To assist governments in the initial form development and regular maintenance of forms, the contribution of this paper is a semi-automatic approach that identifies potential structural inconsistencies or other violations in a set of forms. The approach is based on graph theory to represent forms in a machine-readable format and to analyze them semi-automatically. While the phase “Form Transformation” deals with the abstraction of forms by means of model structures, the “Pattern Specification” comprises the creation of an issue to search for in a machine-readable format. Eventually, in the course of the “Form Checking”, the actual pattern search within the form is executed. We introduce the approach conceptually and demonstrate it in a real-world case by means of its implementation within a software tool and three exemplary issues in form structures. Based on this practical applicability, the approach aims at providing governments with support for reducing inconsistencies or flaws in forms to improve governments’ processes, save time and reduce effort and expenses.

Keywords: E-Government, Forms, Public Administration, Model Query, Patterns.

1 Introduction

In contrast to most companies, processes in governments are driven by forms (Klischewski, 2006). Whereas companies manufacture and trade goods such as cars and computers, governments are service providers that produce forms. At the beginning of government processes, governments receive application forms when a citizen requests a service. At the end, governments issue certificates that are forms to indicate the decision on the application. Thus, forms are input and output of government processes (Sourouni et al., 2008). Input forms contain the necessary information of the applicant that are necessary to deliver a service. Output forms comprise information that describes the government’s decision. Therefore, the “government is the greatest gatherer, processor, and keeper of any kind of information” (Gascó, 2003, p. 8) and forms are central artifacts of government processes. Due to their importance for e-government, especially electronic forms are relevant for the satisfaction of citizens with e-government services (Papadomichelaki and Mentzas, 2012).

Governments deliver a high amount of services to their citizens. In 2004, a typical German local government provided more than 1000 services (Becker et al., 2004). The German Service Catalogue of Public Administration (LeiKa) is a service directory for German governments of all federal levels that

1 Translated from Leistungskatalog der öffentlichen Verwaltung
is maintained by a government organization (Geschäfts- und Koordinierungsstelle LeiKa, 2016). Currently, the LeiKa comprises around 5850 entries. Since many government services require the submission of an application form, governments face an increasing complexity of form management due to the high number of services. The management of forms comprises the initial design and ongoing maintenance of forms in accordance with legal regulations and design guidelines. Although Germany has around 12000 local governments that offer similar service portfolios with many equal services, many of these governments have individual forms with a high heterogeneity (Scholta, 2017).

In this paper, we present a semi-automatic approach that supports governments in identifying violations of structural requirements in forms. The approach assists governments in the initial form development and regular maintenance of forms. The general naming of such approaches is model querying. Within model querying, a real entity such as a business process or a form is transformed into a model that abstracts from unnecessary information. The model is a machine-readable construct that can be analyzed automatically. For this analysis, we specify a pattern that depicts an issue to be searched. Finally, an automatic search can be executed and the results are provided which indicate whether the pattern was found or not. Our approach of model querying is based on graph theory. In graph theory, a structure such as a form can be represented with nodes connected by edges. We can analyze these graphs by using patterns and automatically search for them to detect possible inconsistencies, violations etc.

In a more applied context, governments can develop a set of specific patterns that represents organization-specific design conventions for forms based on design guidelines. Besides, the legislator can deliver a set of general patterns that represents requirements specified in legal regulations. Governments can use the sets of patterns to verify their forms. The forms have to be available in a machine-readable format in order to apply patterns to them. Since governments use dedicated software to construct forms, these tools should offer according export functionalities or can implement our approach. Similarly to information sharing (Gil-Garcia and Sayogo, 2016), governments can share sets of patterns to increase the effectiveness of our approach as governments are used to cooperate than to compete (Nutt and Backoff, 1993). Thus, we provide semi-automatic support for checking forms against laws and design guidelines.

Our research is inspired by the design science research paradigm (Hevner et al., 2004; Peffers et al., 2007). Since the aim of our research is to develop and provide a method for the semi-automatic checking of forms, the outcome is an artifact. However, forms are the boon and bane for governments. On the one hand, they build the basis for many governmental processes. On the other hand, their management is complex. Due to the high and still increasing amount of forms, automatic checking approaches are relevant for shaping modern governments to be more efficient. Thus, the objective of our approach is to reduce the effort of designing and changing forms in governments (internal stakeholders of forms) by enabling semi-automatic consistency checks for huge amounts of forms. In a second step, this may reduce unclarities for citizens and enterprises (external stakeholders of forms). The underlying development idea of the approach consists of the transfer of an algorithmic component (pattern matching) to the subject of forms. Hereby, existing concepts are adjusted in the way that pattern matching can be applied on form structures. The implementation and the exemplary application on examples (our proof of concept) denote the demonstration and a first step towards the evaluation of our approach.

Therefore, our paper is structured as follows. Section 2 presents relevant research background. Subsequently, we introduce and explain our approach in section 3. In section 4, we exemplarily instantiate and demonstrate the approach. Finally, we conclude in section 5.

2 Research Background

2.1 Forms and Form Design

In the course of their processes, governments capture information from citizens through forms. This information is stored in databases that encompass all information of an entire department. Therefore,
databases support the delivery of more than one government service. Instead, a form encapsulates data that is dedicated to a single service. In such a service, a subset of a data entity’s attributes is relevant. Additionally, captions may need to be adapted. Forms prepare those data entities and their attributes for external stakeholders in certain scenarios and are therefore commonly referred to as views on data (van der Aalst et al., 2005; Yao et al., 1984).

Following Lum et al. (1982) and Shu et al. (1982), we differentiate three levels of abstraction of forms as visualized in Figure 1. Abstract forms have the highest level of abstraction. On this level, models of forms represent the structure of forms in a machine-readable format by abstracting from graphical details. In our example, we have a field group “Applicant” that consists of the fields “Given Name”, “Surname”, “Date of Birth” and “Gender”. The model can be enriched with further information on the form elements such as the data type of a field. An abstract form can be detailed by one or more display forms on the next level of abstraction. Display forms can be PDF or web forms that provide a graphical representation and arrangement of abstract forms to obtain a form that is fillable by users. For instance, in our example the field “Given Name” is represented by a text field, whereas two checkboxes indicate the “Gender” of the applicant. A display form can be instantiated by one or more form instances on the lowest level of abstraction. Form instances refer to forms that are filled with data entries. In our example, “Doe” is the “Given Name” of the applicant that has completed this form. A display form can be associated with more than one form instance since different users provide different values on their forms.

During the construction of display forms, governments can rely on design guidelines for an adequate assembling of form elements and suitable selection of their graphical representations (e.g. Bargas-Avila et al., 2010; Jarrett and Gaffney, 2009; Wroblewski, 2008). For example, forms should not request irrelevant data from the user (Jarrett and Gaffney, 2009) and the length of a field should match to the length of common inputs to the field (Bargas-Avila et al., 2010; Wroblewski, 2008). Empirical studies have shown the usefulness of such guidelines for the design of forms that lead to less submission trials and faster completions (Seckler et al., 2014).

In addition to design guidelines, governments have to fulfill legal requirements when developing forms (Axelsson and Ventura, 2007). Laws specify the data that governments are allowed to or have to capture from their citizens. Since form designers are encouraged to minimize the amount of requested data, governments should only capture data on forms that is required by legal regulations.
Beside the manual construction of forms with design guidelines, automatic generation approaches can be applied. While Penadés et al. (2014) create personalized forms, Stadlhofer and Salhofer (2007) generate general forms based on existing semantic descriptions. Another approach is chosen by Dumas et al. (2002) who derive web forms from ontologies. Further generation possibilities such as semi-automatically generating forms from XML documents by Kasarda and Bartoš (2011) exist.

In contrast to the generation of forms, the analysis of forms operates on already existing forms. The aims of such approaches are manifold. For instance, Nguyen et al. (2008) automatically extract field labels from web forms, Zhang et al. (2010) deduce security vulnerabilities from web forms, and Winkelmann (2012) maintains reference models by using data entities and their attributes from web forms.

Despite a lot of work regarding the development and analysis of forms, there is – to the best of our knowledge – no mechanism that allows for an automatic support for identifying violations of structural requirements in forms. As we provide semi-automatic support for the design and maintenance of form structures, we operate on the level of abstract forms.

2.2 Model Querying with Patterns

Since forms are merely collections of data entities in a structured graphical format, they can conceptually be represented by models. Abstraction through models is a widely used concept for simplifying real-world constructs and representing them in a suitable way (cf., Polyvyanyy et al., 2015). Beside the actual entities, these abstractions also include an order of their constituent elements (the form’s elements). Another example of a model would be a floor plan of a building, which abstracts from many details of the reality but keeps the order of the rooms to guide people through the building. It is usual that a real entity may result in different models depending on the purpose of the model (here: to guide the people appropriately). Even though, it is important that a model is always appropriate for the purpose it serves (Becker, Probandt, et al., 2012). In this example, it is essential to mark how a room is accessible (through another). Going further, such model structures can be searched for substructures (e.g., single rooms). For that, a human being would search the plan systematically for the designated destination (e.g., room 122). However, in terms of large models (e.g., huge buildings), the risk of missing something is high.

Semi-automatic approaches for searching model structures for substructures (named model querying) provide a remedy of this issue. Such approaches systematically go through large models and detect previously defined substructures (patterns) with highest precision. The human being simply has to specify previously, what is to be searched in a machine-readable format (that is why it is called semi-automatic) and in an expedient way. In our example, a very simple specification could be room 122. However, one could question this particular case, since a simple text search also provides this functionality. And thinking ahead: Is it not even more simple, pragmatic and faster to go through the plan manually? This is a suitable procedure for small models and rather simple patterns to search for. But how to proceed if the models to examine become larger and larger, and the patterns to search for do not include only one but several elements of a specific order, and further are flexible and adjustable in terms of structure? With rising complexity, the manual effort increases very fast, and quickly reaches a level that makes a manual search impossible.

This is the point at which the manual and even simple technical approaches (such as text search) fail, and more advanced approaches, namely model query approaches, come into play. These approaches basically execute the search as stated above but provide more functionality regarding the complexity and flexibility of patterns to search for. For enabling the search, they consist of – amongst other more detailed requirements – a model query language for specifying the pattern and an algorithm which executes the search (Delfmann et al., 2015). Model querying serves a broad field of applications and there exists a variety of those approaches (e.g., Awad, 2007; Ghose and Koliadis, 2007; Sadiq et al., 2007). However, most of them can only handle specific model types (Becker, Delfmann, et al., 2012). For instance, an application is the process model analysis in terms of efficiency improvement, compliance assurance and error prevention (Delfmann and Höhenberger, 2015; Elgammal et al., 2016; Höhenberger...
et al., 2016; El Kharbili et al., 2008; Mansar and Reijers, 2005; Winkelmann and Weiß, 2011). Many approaches only provide a limited level of pattern complexity, so that only simple patterns can be found. Even though these approaches have their place since they focus on rather complex structures to search in than on complex patterns. A comparison of model query approaches is made by Becker, Delfmann, et al. (Becker, Delfmann, et al., 2012) in terms of complexity handling and pattern flexibility. For our research, we use an approach based on graph theory, especially isomorphism and homeomorphism (cf. Diestel, 2010 for details). The approach is named Diagramed Model Query Language (DMQL) and supports a high complexity of patterns and any types of model structures that can be represented by graphs (Delfmann et al., 2015).

A graph basically consists of nodes that are connected by edges. Nodes can be distinguished by their type. Getting back to our example, two nodes of the type room represent two rooms, while nodes of the types window or door represent these. The types can be freely defined in advance. Furthermore, the nodes can be enriched with labels. The textual label denotes the room 122. A doorway between two rooms can be represented by an edge. Edges may have a direction or are undirected. While edges only connect directly related entities, paths are sequences of edges that can depict entities that are not necessarily directly related. Beside these basic elements, a variety of additional functionalities exist, which are – in parts – explained later, in other parts are omitted for reasons of simplicity. By utilizing the mentioned elements and the underlying algorithm, patterns can be defined and detected in larger structures. The elements, an exemplary pattern and a corresponding result are shown in Figure 2.
3 The Conceptual Approach

3.1 Overview
In order to support the initial design and maintenance of forms, we propose an approach of three phases. The aim of the first phase (section 3.2) is to transform a form from a display form to an abstract form. The input of our approach is a PDF or web form. Since these forms cannot be interpreted by machines, the user has to develop a machine-readable representation of forms. For this purpose, we abstract from some graphical details and focus on a structural representation. The user has to manually transform each form that is to be analyzed. The output of this phase is an abstract form for further analyses.

Subsequently, the second phase (section 3.3) aims at the formalization of requirements for the structure and design of forms. Accordingly to phase one for forms, we develop a machine-readable representation of those requirements in this phase. The input is a natural language specification of an issue. The user has to manually specify each issue once and can repeatedly apply the formalization to all forms s/he wants to analyze in the next phase. The result is a formal specification of a requirement for the structure of forms.

In the third phase (section 3.4), the results of the first two phases are combined. The inputs are the two outputs of the prior phases: The formal specifications of a form and requirement. The aim is to check whether the form violates the requirement. The phase is performed automatically. The result is an indicator that reveals whether a violation is existent. If yes, the violation is graphically highlighted in the abstract form to enable an easy dissolution by the user.

3.2 Form Transformation
Due to the flexibility of our model query approach in terms of model structure, we can represent forms by graphs and analyze them afterwards. The already named elements are sufficient for that. In general, a form is a collection of fields that are graphically arranged. However, the graphical layout of forms only serves the necessity that humans can recognize and understand the form in a better way. In terms of machine automation, the graphical layout does not have any additional value. Thus, a machine-readable format of a form may abstract from a graphical layout, unless the layout is used to point out relations between fields (e.g., an address that consists of a street with a house number and a city with a zip code).

Basically, every field of a form is represented by a node and nodes are connected by edges. Nodes are labeled equally to their denomination in the corresponding form. The type of a node distinguishes between groups (such as headings) that contain collections of fields, fields to fill in and different attributes for additional information provision. Attributes may enrich a field with a determination of its data type (e.g., Integer, Boolean or String) or its representation (such as simple text input, drop-down list etc.). Due to the often-occurring nesting of elements in forms, the attribute level determines the level of a node within the model. Doing that, nested groups of fields can be represented. While levels can be used for enriching fields or groups, data types and representations are only used for fields. However, these attributes are not fixed and can easily be extended or changed.

Edges represent relations of groups and fields within a form to create a coherent graph. Thereby, edges are undirected in case of a level transition (e.g., from the first to the second level) and are directed on the same level for identifying the order of elements. The steps of transforming an exemplary part of a form into its model are depicted in the following. The steps are shown in Figure 3 based on a – compared to Figure 1 – slightly extended conceptual representation of a form.

Step 1 – Extraction. As a starting point, every group (such as the address) and every field (such as the zip code) of a form have to be represented by a single node. The graphical order of nodes can be based on the form structure for facilitating the understanding afterwards, but it does not necessarily have to. Furthermore, additional information for single fields (such as the data type) can also be represented by nodes (however, this depends on the analysis purpose and is optional).
Step 2 – Typing. Once one extracted the nodes, they have to be typed according to their being within the form (Groups, Fields or attributes Data Type, Level or Representation). In terms of attributes, we suggest to use a predefined and small set for achieving highest standardization, simplicity and re-use possibilities of later defined patterns. Thus, the attribute types Data Type, Level and Representation are sufficient for transforming forms. Even though we name this step sequentially after the first one, it can also be conducted in parallel (creating a node and typing it subsequently before creating the next node).

![Diagram of Form Transformation](image)

Figure 3. Steps of Form Transformation

Step 3 – Labeling. After transforming all elements into nodes, we label the nodes according to their exact denomination within the form. This is necessary for analyzing naming inconsistencies in or between forms. Attributes are labeled with their values (e.g., Integer for a Data Type or 2 for a Level). Again, this could also be conducted in parallel to the first two steps.

Step 4 – Connection. Once the nodes are created, we have to connect them group-internally according to their order within the original form (from left to right, from top to bottom). This is necessary to detect the order of elements within a group. Due to the fact that some orders of elements are more comprehensible than others, an order-check is aimed at improving the comprehensibility of forms. Thus, directed edges are used for connecting a node to another node on the same level within the same group. Attributes are connected though undirected edges to the fields they belong to. Finally, we connect the elements to their superior element with undirected edges.
3.3 Pattern Specification

Once the models of the forms are constructed (or available), one has to specify the issues that are to be searched within the models. There exists a variety of different possibilities that can be checked. In general, every issue is represented by a so-called pattern. A pattern is a machine-readable model of a specific issue. The actual look of a pattern depends on the purpose it is used for. In the following, we show exemplarily how a pattern for analyzing the created model of section 3.2 could look like and how this pattern is created step-by-step. We discuss possible application areas and purposes as well as show corresponding pattern examples in section 3.5. In the following, the focus is set on the basic creation of a pattern. For that, we assume that the following requirement for a registration form exists:

- A registration form needs to contain a field for the Date of Birth with the Data Type Date (in terms of data consistency).

As we want to identify violations, we have to specify a violation of the text. Thus, the Data Type of a field Date of Birth must not be Date in the pattern. The creation of the pattern basically follows the same steps as shown in the form transformation (they also may be conducted in parallel/alternatingly). Only the last step is added for executing calculations or applying special rules.

**Step 1 – Extraction.** The first step comprises the extraction of nodes. Against the processing in the form transformation, one has to extract the nodes out of the text. The requirement names the Date of Birth as well as the attributional information Data Type of that field.

**Step 2 – Typing.** After the node extraction, we type the nodes according to the requirement. Therefore, the Date of Birth is set as Field and the Data Type attribute is set accordingly.

**Step 3 – Labeling.** Once we conducted the typing, we extend the field with the label Date of Birth. The Data Type attribute is valued as Date.

**Step 4 – Connection.** For connecting the nodes, the Date of Birth is connected to the attribute by an undirected edge. Attributes are always connected by undirected edges.

**Step 5 – Rule Specification.** Finally, we specify the actual violation by setting the Data Type attribute with its label Date as forbidden. This leads to a check if a node of the type Data Type with a label Date is not present. There exist several further adaptability rules that can be applied. An overview of the rules can be found at (ERCIS Competence Center Conceptual Modeling, 2017).

![Figure 4. Steps of Pattern Specification](image)

3.4 Form Checking

Once a pattern has been specified, we can search it within the transformed forms automatically. For that, an algorithm for model querying based on graph theory is executed. It considers all settings that we made in the pattern specification. Figure 5 depicts a simplified execution of the algorithm. Indeed, the actual execution differs in some points, even though the full explanations fill other publications (Delfmann et al., 2015; ERCIS Competence Center Conceptual Modeling, 2017). Thus, we only want...
to touch its execution for a general understanding. The basic steps are depicted in Figure 5 figuratively and are as follows:

**Step 1 – Type and Label Checking.** First, the nodes are structurally analyzed based on their type and the label. It is important to evaluate those in combination as there may arise situations (especially in terms of forbidden nodes) in which a specific combination of type and label is forbidden but not the type in general. In case a node does not fit to the requirements, it is discarded from the result set (greyed out). This also applies to edges for which both tied nodes have been discarded. This step is performed iteratively until a node fits to the pattern requirements (framed in red).

**Figure 5. Steps of Form Checking**

**Step 2 – Connection Checking.** Once a first node is detected in step 1, its connection to other nodes is checked. Therefore, all possible edges that fit to the specified one are considered (here, paths are not included and thus, only direct neighbors are analyzed). This node is, in turn, assessed as explained in step 1. Once a match is found, it is framed in green in Figure 5.

**Step 3 – Iteration.** The procedure of step 1 and 2 is performed until all nodes have been evaluated. Eventually, we can review the results.

### 3.5 Exemplary Patterns

As described above, our approach relies on the specification and application of patterns to check forms. The patterns represent the requirements that forms need to fulfill. In the following, we describe three categories of patterns to illustrate potential requirements that can be checked with our approach. We provide one exemplary pattern for each category.

**Category I – Legislation.** Legal regulations specify data that governments can or must capture from citizens in the course of a service’s delivery. Therefore, patterns of the category *Legislation* check whether all legally necessary information is captured by a form. Additionally, patterns of this category
can indicate data that is requested on forms but legally not allowed to be captured. A semi-automatic support by our approach is appropriate as the patterns rely on checks for containment. However, these patterns are more complex than a simple text search since we have to consider the context of a field. A field “First Name” in the group “Applicant” has a different meaning than a field “First Name” in the group “Applicant’s Spouse”. To create maximum value for governments, the legislator can provide such patterns when a law is passed.

In Germany, there is a Federal Registration Act (BMG) that regulates the resident registration. In §24 of this law, the legislator states that governments save the data specified in §3 when an applicant wants to register a residence. In §3, the legislator provides a list of attributes that includes, among others, the applicant’s surname, first name, doctoral degree, date of birth, place of birth, gender and nationalities. Our exemplary pattern (depicted on the left in Figure 6) can be specified in different ways (depending on the desired result). We choose a pattern that includes all data distinguished in separate fields that are connected to one group via undirected edges (undirected edges are used, as nodes are always connected to a superior group through undirected edges). The fields have corresponding labels. During model querying, a violation is detected if the pattern is not found in a form (then, either one of the fields may be missing or the structure of the form does not correspond to the specified requirement). As there may exist some characters/words around the specified labels (“Which gender does the applicant have?” instead of “Gender”), the stars (*) serve as placeholders to enhance the flexibility of model querying by allowing an arbitrary number of characters at the place of a star. The capitalization is also ignored for more flexibility.

**Category II – Usability.** Whereas laws make specifications regarding the content of a form, in most cases laws do not contain regulations regarding the design and arrangement of the content. Therefore, the category usability deals with the design of forms, e.g., the operationalization of legal requirements. Patterns of this category check whether forms are designed in a way that they are easy to be completed by users. For this purpose, design guidelines are to be transformed to patterns. Governments can develop such patterns individually or share their knowledge with other governments. Automatic support for checking forms against design guidelines is useful since it is a time-consuming task and can be of arbitrary complexity.

The exemplary pattern is visualized in Figure 6 (in the middle). According to design guidelines, form developers should use adequate data types and representations of fields on forms (Bargas-Avila et al., 2010). For example, fields that require the citizen to enter a date should have the data type Date. Thereby, the input is restricted to dates and citizens cannot provide their input in an incorrect format. For this purpose, a pattern has to detect all Data Type fields that have a different label than Date. For that, the pattern indicates all Fields that contain the term Date in their label. Furthermore, we connect the Data Type attribute with the label Date to the field via an undirected edge (an undirected edge is used as attributes are always connected through undirected edges), and we set the attribute as forbidden. This leads to results that contain a field with Date in their label (the term Date is, again, extended with stars) that is not connected to a Data Type attribute labeled with Date.

**Category III – Standardization.** The prior category aims at the adequate preparation of a single form. In contrast to that, this category deals with consistency across several forms. Patterns of this category ensure that equal scenarios are represented equally on different forms. Existing approaches for form standardization (Folmer et al., 2016) can benefit from automatic pattern checking in terms of efficiency and accuracy if they incorporate our approach. Depending on whether standardization is to be achieved across forms of an individual government or of an entire state or country, the local, state or federal level specifies patterns of this category.

A prominent example of form standardization is the order of elements. This is not relevant for machine-readability as an automatic processing ignores the element order in most cases. However, it is relevant to prevent human mistakes during the entering of data. Who has never mixed up the first name and surname by mistake during the completion of a form? This is only one of several examples in which a standardization can prevent less corrupted data submissions and increase data consistency.
For example, some forms place First Name before Surname whereas other forms may do it the other way around. Therefore, Figure 6 (on the right) presents a pattern that investigates whether a form does not fulfill an assumed requirement: First Name before Surname (one could also perform this analysis vice versa). Thus, a pattern aims at detecting violations of such requirement. For that, we include two fields with corresponding labels and connect each of them to the same superior group via undirected edges. The labeling of a group is optional as we are searching for the field labels in particular. The order of the elements is determined by a directed path from Surname to First name. The direction is converse to the requirement, as the violation is depicted. The path allows other nodes to be in-between for more flexibility, as we are – for now – only interested in the two name fields. The application of these conceptual specifications in a real-world setting is depicted in the following section.

4 Real-World Application and Prototype

Up to now, we provided conceptual insights into our approach. However, to show that it works with real forms, we choose an original form of a German local government that deals with the registration of residences. First, we transform the form into a model and, second, we analyze the model with regard to the introduced patterns of the previous section. The analysis is enabled by the meta modeling tool \( \varepsilon m \). The tool supports arbitrary modeling languages as one can define own languages based on a meta model that needs to be specified. Once we have specified the meta model in terms of which nodes and edges are allowed (according to our conceptually shown approach), we can conduct the form transformation. The pattern specification is enabled by a DMQL plugin for \( \varepsilon m \) that supports the graph-based analysis of models. This plugin contains a graphical pattern editor and the analysis algorithm. It can import the previously defined modeling language, transform a graphically specified pattern into that language and analyze the model. Thereby, the pattern editor is very powerful and contains much more adjustable options than we can show here. However, only the settings shown here are necessary for the presented pattern specifications. Even though, more complex patterns are possible. The tool is freely available at (ERCIS Competence Center Conceptual Modeling, 2017).

For the real-world application, we transform the mentioned registration form. As the model grows per field of the form and gets very large, we only show an exemplary excerpt here. Figure 7 presents the entire original form and the translated excerpt. Following the stated steps of extraction, typing, labeling and connection, we create the corresponding model in \( \varepsilon m \), which is also depicted in Figure 7. Afterwards, the patterns are specified by following the steps of section 3.3. The patterns shown in Figure 8 (on the left) look similar to their conceptual representations in section 3.5 with some graphical adjustments. Nodes in \( \varepsilon m \) are depicted as circles, the types of a node are depicted by an attached symbol.
Edges and paths are depicted equally (paths include their length; $1|\leftrightarrow|\infty$ depicts an arbitrary number of intermediately allowed elements) and the label is moved above a node. The red marking of an edge or a node indicates that this node or edge is forbidden and must not be present in the model.

![Figure 7. Form Excerpt Transformation](image)

![Figure 8. Pattern Specification and Results](image)

Once the model and the patterns are created, the form checking can be executed by means of the DMQL algorithm. Depending on the kind of pattern, two cases can be distinguished: In case we specify a violation of a requirement (e.g., patterns of categories II and III), improvement potential is indicated by a match. In case we define the requirement itself (e.g., pattern of category I), improvement potential is indicated by the absence of a match. Once a pattern has been detected in the model (indicating a match), it is highlighted within the model as shown in Figure 8 (on the right). Additionally, one can inspect the result by means of the color marking. Figure 8 indicates that the standardization and the usability patterns lead to matches and show possible improvement potential. Given the case that there is no match
detected (e.g., for the legislation pattern), a short message appears. Eventually, the results are saved and can be reviewed as often as needed. In a larger extent, this analysis can be conducted automatically for an arbitrary number of form models and patterns (both have only to be created once and kept up to date).

5 Conclusion and Outlook

The contribution of this paper is a semi-automatic approach that identifies violations of structural requirements in forms. Following the design science research paradigm, we designed a three-phase approach. In the first phase Form Transformation, a machine-readable representation of a form is developed. Subsequently, the Pattern Specification phase deals with creating a formal representation of a requirement regarding the forms’ structures. In the third phase Form Checking, the model querying is applied in which the pattern is searched within the form. When applying a set of patterns to a government’s forms, a user of our approach receives a list of potential violations and improvement potential. We have shown the practical applicability and effectiveness of our approach by the implementation in a software tool and the demonstration with a real-world case.

Our approach is valuable for both researchers and practitioners. To the best of our knowledge, literature does not provide an approach for the automatic support to identify violations of requirements or improvement potential in form structures. Researchers can integrate our approach into their design guidelines and extend it. Practitioners can apply our approach to check their forms of government service delivery. The areas of application result from the categories of patterns: check whether forms conform to legal regulations, usability requirements and standardization attempts. To achieve a high reach and enable an immediate application, legislators can provide patterns when they modify existing laws or pass new laws. Additionally, governments should share sets of patterns with each other to gain value and save effort due to the re-use possibilities.

Of course, our approach is subject to limitations. First, there is a certain level of modeling knowledge required to apply this approach. However, against that, the graphical representation supports the modeler and can – once the basic knowledge of modeling exists – even more facilitate the understanding of possible issues in forms. Second, our approach is time-consuming since the formal specifications for all forms and patterns are to be developed manually. Here, the inclusion of text mining for an assisted transformation of forms and patterns could provide a remedy for reducing the necessary effort. Doing that, a draft of a form model or pattern could be created automatically and would only need to be adjusted afterwards. To reduce time efforts, the model size and the required modeling expertise for now, the specifications’ complexity can be reduced by decreasing the number of model elements. Our way of modeling is a suggestion that is to be adapted to each scenario as models always serve a certain purpose that varies from case to case (Becker, Probandt, et al., 2012). For instance, the order of fields is relevant in some case, and in some it is not. Then, the directed edges between elements on the same level can be omitted. Whether the application of our approach is timely beneficial is to be decided for each scenario individually. Third, the semi-automatic principle of our approach contains the risk of errors made during the manual form transformation process. Again, methods like text mining could support here by pre-transforming forms into graph structures to reduce the amount of completely manual transformation steps, though it cannot fully prevent such mistakes. Fourth, we do not claim that our approach will detect all violations and that all potential violations identified by our approach are actual violations. Instead, our approach supports the users in maintaining forms. It accelerates and complements a purely manual procedure. However, it is a heuristic whose results are to be confirmed by the user.

In future work, we will conduct a comprehensive evaluation with a high amount of forms in real-world scenarios. Interviews with employees can reveal insights on practitioners’ views on the usefulness of our approach. Additionally, the set of patterns and their categories can be extended. Researchers can develop a catalogue of patterns that can be applied to check forms in governments. As already indicated within the limitations, an automatic parser is envisaged to omit the error-prone manual form transformation. Shifting the investigation from a reactive to a proactive one, our approach could also be useful during the construction of a form. The algorithmic check can prevent such violations from the beginning.
References


