Semi-automated Test Planning for e-ID Systems by Using Requirements Clustering

Baris Güldali, Stefan Sauer, Gregor Engels
Software Quality Lab, University of Paderborn
Warburger Str. 100, 33098 Paderborn, Germany
{bguldali,sauer,engels}@s-lab.upb.de

Holger Funke, Michael Jahnich
HJP Consulting GmbH
Hauptstraße 35, 33178 Borchen, Germany
{holger.funke,michael.jahnich}@hjp-consulting.com

Abstract—In acceptance testing, customer requirements as specified in system specifications have to be tested for their successful implementation. This is a time-consuming task due to inherent system complexity and thus a large number of requirements. In order to reduce efforts in acceptance testing, we introduce a novel approach that exploits redundancies and implicit relations in requirements specifications, which are based on multi-viewpoint techniques, in our case the reference model for open distributed processing (RM-ODP). It deploys requirements clustering and linguistic analysis techniques for reducing the total number of test cases. We report on concrete experiences with this approach within joint R&D work of the Software Quality Lab (s-lab) of the University of Paderborn and HJP Consulting, an international consulting company, specialized in planning, procurement and acceptance testing of national electronic identification (e-ID) systems. The paper is concluded with an overview on the current tool support especially for automated detection of the redundancies and implicit relations in requirements. Also the future work on the tool support for the overall test specification process is discussed.

Keywords—Acceptance Testing, Test Planning, Requirements Clustering, Linguistic Analysis, Open Distributed Processing Systems.

I. INTRODUCTION

All countries of the European Union are currently in the process of introducing electronic passports, which are based on RFID technology and on digital biometry. These technologies created a new class of distributed IT systems for the issuance and the application of such identification documents, for biometric verification and border control. We focus our work on this class of IT systems having the following characteristics:

1) Electronic identification documents – so called e-ID – having limited computing power and making use of RFID technology. These e-ID represent individuals and connect to the IT system to verify the biometric identity of these individuals.

2) Distributed system services and components for data collection, enrolment, personalization, issuance, authentication and verification.

3) Heterogeneous technologies and multiple component suppliers.

4) Strong requirements for system security: cryptography is widely used to provide confidentiality, integrity and availability of information processed in the system.

All characteristics result in a high degree of system complexity. In the following we will call such systems "e-ID systems". We will discuss the objective to specify and test requirements (acceptance testing) for such e-ID systems.

In order to specify requirements and develop coherent system architectures for e-ID systems, the reference model for open distributed processing – RM-ODP – according to [11] and [14] can be used as a framework for system specification. RM-ODP specifies an e-ID system using five different viewpoints as illustrated in Fig. 1.

Using different viewpoints, the e-ID system specification will contain dependencies between viewpoint specifications because specifications contained in different viewpoints implicitly correspond to each other [11]. Due to the inherent system complexity of e-ID systems, the number of requirements related to a viewpoint specification may easily exceed 1000 requirements. But there exist redundancies and dependencies in the requirements between and within viewpoints, which we can exploit.

Although this redundancy helps customers and designers to cope with system complexity and develop consistent system requirements, overhead is automatically introduced into
acceptance testing, because we have to verify every single requirement by at least one test case; often due to contractual reasons. We have therefore looked for a method to cluster requirements in order to reduce this overhead for specifying tests and thus to optimize acceptance testing without reducing the quality of the tests. The following chapters will present clustering algorithms for requirements being specified using informal (natural) languages.

In section II, we will present existing clustering techniques which can be applied to requirements clustering. In section III we introduce our novel approach for requirements clustering. Moreover, the process of generating test plans for acceptance testing is explained in this section. After that, we will present tools that can be used for an implementation of the requirements clustering algorithm in section IV. Finally, we have applied our clustering algorithm to the system specification of a national e-passage issuance system. The results and the lessons learned are presented in section V.

II. RELATED WORK

Clustering means to group entities (data, objects) with respect to some predefined characteristics of these entities. The main targets of clustering are to reduce the complexity of the huge space of entities and to understand it, and to extract similarities between clustered entities. In computer science, clustering is used in numerous application areas for different purposes [15]: e.g. in computer graphics clustering is used for image segmentation or object recognition, in knowledge based systems it is used for information retrieval.

When applying cluster analysis, a number of questions need to be answered [25]:
- What are the entities to be clustered?
- When are two entities to be found similar?
- Which concrete clustering algorithm do we apply?

After having answered these questions, typical cluster analysis involves the following steps [15]: (1) entity representation (optionally including feature extraction and/or selection), (2) definition of an entity proximity measure (similarity function) appropriate to the data domain, (3) clustering or grouping, (4) data abstraction (optional), and (5) assessment of output (optional).

Each of these activities can be conducted in many different ways. For step (1) quantitative or qualitative features of entities can be used. Based on the chosen entity representation, the similarity function designed in step (2) can use distance functions, association coefficients or probabilistic techniques [25]. Also the clustering algorithms in step (3) differ. Two important categories of clustering algorithms are graph theoretical algorithms and hierarchical algorithms. Other categories are defined in [25, 15]. Graph theoretical algorithms represent entities to be clustered as nodes in a graph. Edges between nodes represent their relations. Graph algorithms like spanning tree are used to construct clusters. Hierarchical algorithms construct a hierarchy tree for clusters. By defining a cut point in the hierarchy tree, final clusters are formed [25]. For clustering analysis it is important to experiment with alternatives and to optimize the steps for finding the best clustering for your own purpose.

Clustering techniques are also used for various purposes in requirements engineering for software: e.g. functional decomposition [10, 17, 20], remodularization [25], incremental delivery [10, 9], feature modeling in software product lines [4]. These approaches mainly aim at organizing related requirements specified in requirements documents as clusters for forming subsystems that reveal certain desired properties. Most of them use graph theoretical algorithms for clustering where each requirement is represented by a node in a graph. The edges between nodes represent relations between requirements which are used for building semantic clusters of requirements. The edges are labeled with similarity measures which are computed in different ways. Using linguistic analysis some approaches split requirements sentences into atomic parts like subject, object and verb and compute the degree of overlapping [17, 20] as a similarity measure. Having extracted qualified sentence parts, the computation of similarities and clustering can be done automatically. However, the extraction of sentence parts is done manually which is expensive and error prone. Another problem with these approaches is that synonyms in sentences are not considered in computing the similarity (e.g. citizen ↔ applicant). Thus useful semantic information indicating similarity can get lost. Other approaches propose a manual activity in defining similarity relations between requirements [10, 9, 4]. Requirements are manually analyzed for functional relations, object relations and dependency relations. These relations are scaled and weighted for computing similarity measures for each pair of requirement. Since the similarity between requirements is measured manually, this approach is also expensive. Moreover it is subjective so that reproducibility of clustering analysis is not guaranteed.

In our work we apply requirements clustering to improve the planning and analysis for acceptance testing. The process of acceptance testing is illustrated in Fig. 2 based on the fundamental test process of ISTQB (International Software Testing Qualifications Board) [19]. The process starts with planning the testing phase, continues with analysis, implementation and execution of test cases and ends with evaluating and

![Figure 2. Process of acceptance testing based [19]](image-url)
archiving of test results. In the literature of test automation many approaches are presented mainly about the test case generation and test execution [5, 6]. In our approach we focus on earlier steps, i.e., planning the testing activities and analysis of the test basis. We want to support these steps using automated techniques and finally optimize the whole test process. Generation of concrete test cases or their execution is not the subject of this paper.

We are interested in answering the following questions:

- How can we create a minimal test specification which optimally covers all the requirements?
- How can we identify redundant requirements which could lead to unnecessary testing activities?
- How can we compute some ordering of requirements such that they can be tested in this order, thus saving effort for additional pre- and post-processing?

The clustering techniques addressed in this section seem to be very promising for answering these questions. However we are aware of the problems of the cited work (especially because of the high manual intervention) and want to improve these points by systematizing and automating the manual steps.

III. APPROACH

As stated in the introduction, we use RM-ODP for specifying requirements on e-ID systems. RM-ODP addresses five viewpoints on e-ID systems to be modeled: enterprise view, computational view, information view, engineering view and technology view. Redundancies and dependencies may occur by capturing requirements in these viewpoints, because some aspects are described in different contexts. Redundancies and dependencies lead to redundant testing activities during acceptance testing later on. This is because of the current procedure for acceptance testing where each requirement must be tested explicitly by a test case. Thus mainly due to contractual reasons for \( n \) requirements at least \( n \) test cases are needed. A typical system specification contains 1,000 to 5,000 requirements but there are examples of national e-ID systems with even more than 10,000 requirements. By identifying the redundant requirements and by clustering related requirements, we aim at reducing the number of test cases by only testing a representative of redundant requirements. Therefore we aim at keeping the test quality as unaffected as possible.

Testing activities in acceptance testing of e-ID systems not only cover software testing, but also testing (i.e., validating) many other aspects like organizational aspects, security aspects, hardware and infrastructural aspects as specified in RM-ODP viewpoints. Thus test cases not only include validating of software requirements by dynamic testing, but also validating of further requirements by static testing. ISTQB defines many activities for static testing, e.g. informal review, technical review, walkthrough and inspection [19].

By planning the testing activities for acceptance testing, we differentiate between requirements which describe systems’ interactions with users or states of system artifacts during interactions. We also treat requirements specially which describe general system properties. Fig. 3 shows our domain model for test plan specification in acceptance testing. For testing behavioral requirements we refine them into Actions. Actions represent dynamic activities (Process) which are conducted by a user or a system component (Actor). One or more artifacts (Object) can be involved in an Action. State requirements describe the properties of Objects for a certain point of time. Objects can be linked to other Objects or to Actors which are also Objects. Rule requirements describe properties of system artifacts which must hold the whole time. A TestPlan represents a sequence of testing activities including interactions with/within the system and validations of expected behavior. It maps an Action to a TestStep which tests an interaction with the system. For doing this, we first examine the conductibility of the behavior described by the Action, and then we use Asserts for verifying the states of involved artifacts and/or the fulfillment of rules.

![Domain model for planning acceptance testing](image)
The process of test plan specification aims at creating a set of test plans covering the requirements optimally. Later on testers should create and execute concrete test cases conducting the test steps and asserts using concrete test data. Up to now we have created test plans and mapped them to the requirements manually. This is an expensive task. Another problem with the manual activity is that resulting test plans are not optimal due to the following reasons:
- redundant test steps and asserts remain undetected which increases the efforts later by testing unnecessarily,
- the relation between the test steps and the asserts remain disregarded, thus the tester have to conduct similar testing activities several times unnecessarily,
- test steps and asserts are not ordered optimally, thus the tester have to take care of pre- and post-processing manually.

We intend to use clustering techniques to automate some of the manual activities in order to systemize, optimize and speed up our process. Based on the reference process for cluster analysis introduced in section II our approach has three main phases (Fig. 4):

1. **Annotation**: Here we prepare requirements documents by restructuring and splitting texts into sentence parts for further processing.
2. **Clustering**: In this step we compute the similarity between requirements and group them into clusters.
3. **Test plan specification**: In this step we create test plans by selecting test steps and asserts. Later on testers should design concrete test cases with respect to the test plans, which however is not a part of our process explained in this paper.

The phases of the process are automated to different degrees. While annotation and clustering can be automated well, test plan specification needs still much human interaction. However, also for the third phase we can profit from automation to assist tester’s activities.

We propose an iterative process where after the steps the results should be reviewed and optimized, respectively, which is mainly a manual activity. In the next sections we explain these steps and their relations to the reference process of cluster analysis in detail. Afterwards, in section IV, we explain the current prototype tool support we have implemented.

### A. Annotating Requirements

Clustering analysis demands for definition of entity features which will be used for computing the similarity between entities [15]. For requirements clustering, attributes for requirements have to be defined. In the requirements engineering literature, many attributes for representing organizational information are proposed (e.g. unique identifier, version, status, priority, type: functional/non-functional, etc.) [22]. Most of the requirements engineering tools not only support editing these attributes, they also allow for defining new attributes [7]. Having optimization of test planning in mind, we need much more semantic information about requirements than just organizational ones. These semantic attributes should help us by creating a test plan specification, i.e., instances of the domain model given in Fig. 3. More concretely, we want to analyze requirements documents and identify the types of requirements (Action, State, and Rule) and annotate them with semantic attributes addressing the relevant aspects (Actor, Process, and Object).

We use linguistic analysis for annotating requirements. We parse sentences in requirements documents and split them into sentence parts according to some syntactical rules (this approach is similar to [8] and [1]). As next we explain these activities in detail and give examples.

#### 1) Splitting

In our requirements documents each requirement is captured by a unique identifier and sentence block (upper table in Fig. 5). The Id “SYC545M” has the following meaning: “SY” symbolize that this requirement belongs to the requirements document for system architecture (“IN” for interfaces, etc.). “C” stands for computational viewpoint (E: enterprise, I: information, G: engineering, T: technology). The viewpoint character is followed by an ordering number. “M” at the end stands for a mandatory requirement (O: optional, R: recommended). The sentence block contains one or more sentences including a key verb SHALL, SHOULD or MAY showing the obligingness of this requirement [2]. For requirements capturing this can be meaningful, however for testing we need to split sentence blocks into single sentences. Thus the atomic sentences can be handled by further steps of our approach separately.

![Figure 4. Process overview](image-url)
SYC545M-2 Delivery SHALL notify the passport life-cycle management about the cancellation.

SYC545M-1 An old e-passport of the recipient that is still valid SHALL be cancelled.

SYC545M An old e-passport of the recipient that is still valid SHALL be cancelled.

SYC545M Delivery SHALL notify the passport life-cycle management about the cancellation.

Figure 5. Example for requirement splitting

Splitting sentence blocks makes parsing sentences easier; however, semantic relations between single sentences can be lost. In order to avoid this, the origin of the sentences is kept in the new Id which includes the original Id as prefix and a sequence number. This information will be used in further steps of clustering.

2) Language Restrictions

Once we have atomic sentences, these can be linguistically analyzed and syntactic sentence parts can be extracted. Since requirements are captured in natural language, syntactic structure of sentences varies strongly. This variety can hinder the automated syntactic analysis of sentences. That is why we restrict the grammar of our language by using templates. In Fig. 6 we give an example for templates based on [22]. It shows how subject, predicate (verb), object and other syntactical elements should be structured in requirements capturing. Another assumption is that the sentences are written in active voice. Such restrictions simplify the syntactic analysis; thus semantic information we need for clustering such as actors, objects and process can be extracted more easily.

Figure 6. Template for requirements based on [22]

3) Annotation

After having requirements in a structured form, the linguistic analysis can start. For that, sentences have to be parsed and transformed into syntax trees. Fig. 7 shows how the requirement specified in SYC545M-2 is represented as a syntax tree. The nodes of the syntax tree are composed of the following parts: the label of the node identifying the type of the node, and a list of other nodes, e.g. (S (NP ...) (VP ...)). A special node is the leaf node which contains the type of the node and the value of the node, e.g. (NN Delivery). Examples for node types are S, NP and VP. S is the root element of the syntax tree and represents a sentence. NP stands for nominal phrase and VP stands for verbal phrase.

(S (NP (NN Delivery)) (VP (MD shall) (VP (VB notify) (NP (DT the) (JJ passport) (JJ life-cycle) (NN management)) (PP (IN about) (NP (DT the) (NN cancellation))))))

Figure 7. Syntax tree for SYC545M-2

Considering the structure given by the template defined in Fig. 6, we extract the semantic attributes as follows:

(1) The first NP node of S represents the Actor of the requirement.
(2) The first VP node of S contains the Process leaf node.
(3) NP node extracted in step (2) can contain one or more NP nodes representing the Objects. We assume that the first NP in VP is the direct object of the sentence which is primarily affected by the Action.

In this way the computation of similarity is not only based on the similarity of annotations, but also different terms having similar semantics are considered by the similarity measure.

In textual specification of requirements it is common practice that important terms are defined in a glossary. Also acronyms and synonyms of these terms are given in the glossary. In annotating requirements we also make use of the glossary. Thus, extracted nouns in sentences like “ePMS” or “electronic passport management system” or verbs like “notify” or “inform” are treated as same. Rupp et al. define word lists for verbs in order to standardize them [22]. This kind of word list can be used for systemizing definition of synonyms. In this way the computation of similarity is not only based on the similarity of annotations, but also different terms having similar semantics are considered by the similarity measure.

Besides the glossary, our requirements documents also include UML [21] models for refining requirements in the viewpoints. For example, use cases are used for the refinement of the system functions and their dependencies in the enterprise viewpoint; the information viewpoint contains class diagrams for representing data artifacts and their dependencies, and life-cycle models of data are represented by state charts; interfaces of computational components are represented by component diagrams. The annotation also considers the model elements in these models.

After the annotation step the result should be reviewed as shown in Fig. 4 for completeness and appropriateness. If the annotations contain words which are synonyms or acronyms

<table>
<thead>
<tr>
<th>Id</th>
<th>Actor</th>
<th>Process</th>
<th>Direct Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYC545M-2</td>
<td>Delivery</td>
<td>notify the passport life-cycle</td>
<td>management</td>
</tr>
</tbody>
</table>

Figure 8. Annotated requirement
to currently used words in the glossary, the word list for synonymous and acronyms should be extended and the annotations can be either corrected by hand or annotation step can be repeated.

B. Clustering

Having annotated requirements with semantic attributes, we can now cluster requirements with respect to their similarity. By doing this we aim at identifying redundancies and similarities between requirements such that the redundancies can be represented just by one representative in the test planning. In this way detected similarities are exploited for specifying efficient test plans. In the first step of clustering, we group requirements into three coarse grained clusters: Actions, States and Rules. In the second step, we group requirements in cluster Actions into fine-grained clusters.

1) Grouping: Actions, States, Rules

As explained in section III, a test plan consists of test steps and asserts (see also Fig. 3). Test steps refer to behavioral system requirements and thus comprise the dynamic aspects of testing. Asserts refer to static aspects in requirements specification which specify the states and rules on system artifacts. Thus asserts constitute the validation points in testing. By designing test plans, which are sequences of test steps and asserts, we cover requirements of type Actions, States and Rules as explained in section III.

This approach is inspired by the work of Sneed who also differentiates between requirements in a similar way [23]. The difference of our approach is that Rules do not refer to Actions, but to Objects. Instead we use Conditions for setting differentiates between requirements in a similar way. Another similarity function \( c \) computes the dependency between \( r_1 \) and \( d_2 \) which is comparable different.

For differentiating between Actions, States and Rules, we look at the Process annotation of requirements [22]. A wordlist for Process verbs determines the type of the requirement (see Table I for example). Actions are identified by domain-specific verbs, but also by general verbs. States are identified by verbs which describe the properties of a system artifact. Rules state general terms about the system and its artifacts which cannot be classified as States.

<table>
<thead>
<tr>
<th>Action</th>
<th>State</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply</td>
<td>send</td>
<td>have</td>
</tr>
<tr>
<td>personalize</td>
<td>receive</td>
<td>be</td>
</tr>
<tr>
<td>deliver</td>
<td>update</td>
<td>comprise</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

2) Semantic Clustering

Having roughly grouped requirements into three coarse grained clusters; we refine these clusters into fine-grained ones which comprise semantic relations between requirements. In this step of clustering we use a graph-theoretical approach.

Each requirement is represented as a node. The edges between nodes represent relations between requirements. The label of an edge indicates the degree of the relationship. We group requirements into clusters, i.e., the nodes are distributed to new graphs. According to coarse-grained clustering as explained in the last section, this step starts with three clusters which are disjoint: \( A \): Actions, \( S \): States and \( R \): Rules. Initially the nodes in clusters are not linked to each other.

In order to determine the relationships between requirements we use a similarity function \( s \) which returns a similarity degree for two requirements as shown in Fig. 9:

\[
\begin{align*}
\text{Figure 9. Similarity function} \\
\end{align*}
\]

Using the similarity functions \( s \) and \( c \) we create a similarity matrix for each cluster \( A, S \) and \( R \) of size \( n \times n \) where \( n \) is the total number of requirements in each cluster. For each pair of requirement \( r_1 \) and \( r_2 \) we compute \( s(r_1, r_2) + c(r_1, d_2) + c(r_2, d_1) \). Because of the symmetric computation of similarity, the similarity matrix represents an undirected graph with weighted edges between nodes. We use well known graph traversal algorithms (e.g. spanning tree as explained in [10]) for splitting this graphs into clusters including highly related requirements.

Before we apply the clustering algorithm we apply a heuristic for improving the result of clustering for the next step of our approach, i.e. test plan specification. This heuristic assumes that requirements in the requirements documents are already organized semantically. Especially the requirements of type Action are typically captured in an order that conforms to the usage scenario. In test plan specification we are interested in executable sequences of test steps. Therefore we put requirements captured in the same use case in the same initial cluster. Starting with the initial clusters we add new requirements into these clusters considering the similarity measures and a predefined threshold. For each requirement in the cluster, all other requirements having a similarity degree equal or greater than the threshold value are added into the same cluster. At the end of clustering we achieve clusters

| Table I. Word List for Process Verbs |
|---------|--------|--------|
| domain specific | Action | State | Rule |
| apply | send | have | support |
| personalize | receive | be | conform to |
| deliver | update | comprise | allow |
| ... | ... | ... | ... |
with highly related requirements. The clusters can have overlapping elements.

The iterative process as shown in Fig. 4 enables reviewing the resulting clusters and improving the clustering by changing the threshold value. Later on we also want to supply other clustering algorithms in our implementation such that different algorithms and similarity measures can be used for better clustering.

High value of the similarity measure between two requirements indicates potential redundancy. In this case we analyze the Actor, Process and Direct Object annotation of the requirements. In case of detected redundancy a representative requirement is identified and used in further steps of the approach. The test results for the representative are transferred to the redundant requirements, thus saving efforts by avoiding redundant test activities.

C. Test Plan Specification

As shown in Fig. 3, by designing test plans we systematically cover requirements of type Action, Rule and State with test steps and asserts. For that we first design test steps which comprise the dynamical aspects of testing activities. In test steps users or components of the system (Actors) interact with the system (other components, respectively). We use asserts for checking whether states of system artifacts (Objects) conform to requirements and whether rules are fulfilled.

A test plan contains a sequence of test steps and asserts which address corresponding requirements. These have to be ordered such that they can be checked in the given order. We use the results of cluster analysis from the last section in order to cover similar and related requirements within the same test plan in order to make the test process efficient. We also choose a suitable ordering of test steps and asserts such that pre- and post-processing for individual test steps is handled systematically. For defining an order of requirements to be tested we use patterns as shown in Fig. 11 and Fig. 12. Using these patterns we can identify further relations between requirements which are useful for ordering them.

Fig. 11 depicts a pattern for identifying two requirements where the object of the first requirements of type Action is equal to the subject of the second requirement of type State or Rule. During acceptance testing these two requirements can be tested in the order \(<r_1, r_2>\) so that the requirements specified in \(r_2\) can be checked after the actions specified in \(r_1\) are conducted.

As for the other steps of our approach, also the results of test plan specification should be reviewed and if needed improved. For that the order of the test steps and asserts should be checked whether the semi-automatically computer order of requirements is testable. If problems are detected the clustering can be repeated by changing the applied heuristic or by changing the threshold values of the applied algorithm or by choosing other algorithms and similarity measures. Also the test plan specification can be repeated by manually extending or reducing the test steps or asserts.

D. Summary

Before introducing the tool support we shortly discuss the fulfillment of the issues we addressed in section II:

Clustering-based test planning as introduced in this paper helped us to reduce the size of the test plan specification. With the former procedure we used to construct a test case for each requirement. The new clustering approach helps us to group requirements such that we can cover more requirements with one test plan. Afterwards one test case implementing the test plan can cover more than one requirements, thus saving testing efforts.

After clustering requirements into fine-grained clusters, we are able to check the similarity of requirements by comparing their semantic attributes. A high similarity measure indicates redundancy. The final decision whether detected
similarities are real redundancies and how redundant requirements can be represented by which representative is still a manual process.

Our approach identifies different types of requirements, which are Actions, States and Rules. By creating test plans we concentrate first on Actions because they comprise the behavioral aspect of testing. Computation of the similarity and the conditional dependency between requirements helps to construct logical and temporal orders of test steps. Before or after a test step, asserts are added to the test plan corresponding to requirements from States and Rules. Thus test steps fulfill the preconditions of other test steps which depend on the results of the former ones. Finally, asserts are applied to the right time if changes are expected in the system or general properties should be examined.

IV. Tool Support

As shown in Fig. 4, our process contains three main phases: Annotation, Clustering and Test plan specification. Some of the activities in these phases can be automated, which makes the process faster, more efficient and reproducible. In this section we discuss the potential of automation in each phase and present the current status of our prototype. Table II shows an overview of the activities in our phases and the potential of the automation. Also some candidate tools are given.

In the phase of Annotation, textual requirements documents have to be prepared for further steps. For that the requirement texts have to be split into atomic sentences. Furthermore, the requirements Id has to be extracted. Then the sentences have to be syntactically analyzed in order to extract the semantic attributes as described in section III.A.3). For automating these steps we use, besides some classes we implemented ourselves, the Stanford Lexicalized Parser which is an Open Source library for linguistic analysis [24].

In the phase of Clustering the annotated requirements have to be grouped into coarse grained clusters of Actions, States and Rules. We do this by analyzing the Process annotation of the requirements. In fine-grained clustering the similarity of requirements has to be computed based on the similarity functions defined in section III.B.2). Then the clusters have to be computed by using some heuristics. For these steps we use self-implemented components and the Similarity Library [16]. We use the Similarity Library for identifying the similar Process verbs according to the WordNet database [18]. For computing the similarity matrix we use the Colt Library which is an open source library for high performance scientific and technical computing in Java [3].

In the phase of Test plan specification we use self-developed components for computing the ordering of requirements in test plans. Further activities for test case generation using concrete test data are currently supported by our prototype. However, we want to integrate FIT (Framework for Integrated Testing, http://fit.c2.com/) for transforming the test plans into test cases.

Fig. 14 shows the architecture of our desired tool environment for the future. We want to integrate our specification framework with state of the art modeling tools (e.g. Enterprise Architect) in order to include specification models in requirements clustering. Currently our prototype has integrated Stanford Lexical Parser, Similarity Library and the Colt Library successfully (see dashed region in Fig. 14).

<table>
<thead>
<tr>
<th>Activity</th>
<th>Degree of automation</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Splitting</td>
<td>high</td>
<td>Stanford Lexical Parser, HJP Framework</td>
</tr>
<tr>
<td>• Syntactical analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Coarse grained</td>
<td>high</td>
<td>Similarity Library, Colt Library, HJP Framework</td>
</tr>
<tr>
<td>• Fine grained</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test plan specification</td>
<td>medium</td>
<td>HJP Framework, FIT</td>
</tr>
<tr>
<td>• Test plans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Test cases</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V. Case Study

In this section we report on our early experiences with the clustering techniques and the prototype tool support. Our approach contains many steps and tools which should be integrated in a way such that the resulting concept brings benefit compared with the former procedure. We also aim at analyzing the scalability of the approach. In this section we first report on a case study we conducted and on the lessons learned.

A. e-Passport System

We have evaluated our approach on a typical architecture specification of a national e-passport issuance system. Fig. 15 depicts a simplified representation of such a system. The users of the system are mainly citizens and diplomatic citizens and officers of several ministries. It should support users during the activities of e-passport issuance: enrollment, personalization and delivery. During the enrollment process citizen’s biometric data is captured which is then stored on a RFID chip integrated in the e-passport booklet. The system is distributed in offices in 10 cities. It contains 70 workstations, 10 servers, and 6 personalization systems. The system should be able to process up to 5000 applications daily.
The exemplary specification document we considered in our case study contains 319 requirements. This specification document is part of a system specification with more than 7 requirements documents. After splitting the requirements we gained 405 atomic sentences. Table III shows some statistics about the distribution of the requirements onto the RM-ODP viewpoints and their types.

Table III. Statistics on Analyzed Requirements

<table>
<thead>
<tr>
<th>Viewpoints</th>
<th># of Req.</th>
<th>Action</th>
<th>State</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise</td>
<td>103</td>
<td>84</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>Information</td>
<td>61</td>
<td>25</td>
<td>29</td>
<td>7</td>
</tr>
<tr>
<td>Computation</td>
<td>141</td>
<td>89</td>
<td>39</td>
<td>13</td>
</tr>
<tr>
<td>Engineering</td>
<td>67</td>
<td>27</td>
<td>26</td>
<td>14</td>
</tr>
<tr>
<td>Technology</td>
<td>33</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>405</td>
<td>225</td>
<td>133</td>
<td>47</td>
</tr>
</tbody>
</table>

As explained in section III.C.2 we heuristically set initial clusters of requirements which are already logically grouped in chapters of the requirements document. In our case study we have chosen the three components “Enrollment”, “Personalization” and “Delivery” in computational viewpoint as starting points. Thus the requirements in these sections were grouped into initial clusters (see Table IV). By applying the clustering algorithm described in section III.B we extended the clusters by further requirements from other viewpoints. Table IV gives some statistics on the number of requirements in clusters. Initially added requirements could be extended in each of three cases which means that relationships between requirements from several viewpoints were detected.

For the three scenarios above no related requirement from the technology view could be indentified automatically. The reason for that is that the word list for synonymous was not complete at the time point of case study and thus some related requirements from the technology view could not be identified automatically. For example, technology view specifies requirements on how the “fingerprints” should be stored on the RFID-chip. The requirements in the considered scenarios handle “biometric data”. Even if fingerprint is also a biometric data, textual similarity analysis ignored the similarity between these requirements. By extending the word list for synonymous this relation can be considered by the computation of clusters.

Table V shows the overall reduction of the redundant Action-requirements using different thresholds for edge weights used for clustering.

Table V. Number of Action-Requirements after Clustering Identified Redundancies

<table>
<thead>
<tr>
<th># of Req.</th>
<th>Annotation</th>
<th>Similarity</th>
<th>Clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4.594</td>
<td>&lt;1</td>
<td>103</td>
</tr>
<tr>
<td>50</td>
<td>19.938</td>
<td>9</td>
<td>1.534</td>
</tr>
<tr>
<td>100</td>
<td>35.797</td>
<td>15</td>
<td>4.602</td>
</tr>
<tr>
<td>400</td>
<td>141.859</td>
<td>31</td>
<td>55.198</td>
</tr>
</tbody>
</table>

B. Lessons Learned

In our case study we evaluated the algorithms and the tool support we have developed. Thereby we have realized many aspects not only about our approach of requirements clustering, but also about our requirements specification process. In this section we briefly talk about our experience.

First of all we have realized that we have to improve the requirements capturing. We have learned the following lessons:

**Do not use ambiguous formulations!**

By parsing requirements texts we have realized some sentences cannot be parsed reasonably because of ambiguities. For example, if words (pronouns) like “it”, “these” are used which refer to some other words in former sentences, we could not automatically determine which words were
exactly referred. However, such formulations make long texts easier readable. That is why we do not completely let using these words; nevertheless, we consistently use them for the subject of the first sentence in a requirement.

**Avoid sentences in passive!**

If sentences are written in passive voice most of the time the actual instance which actively acts is left out, e.g.: “The application has to be processed”. In this sentence it is not specified who processes the application. Thus we have no chance to determine the actual Actor. Therefore we propose using only active sentences in S+V+O order.

**Use consistent terminology by creating a glossary!**

When writing textual requirements, it is common practice that synonyms are used in order to avoid monotonous formulations. This was a problem for our similarity analysis because requirements containing synonyms could not be put into them same clusters directly. Therefore, we have used word lists for synonyms in order to assign similar words for semantic attributes in annotation. The word list should be a part of the glossary.

**Split requirements into atomic sentences!**

It was common practice to formulate long requirements texts containing many sentences. Such texts are desirable for specifying complex requirements. However, for requirements analysis and test plan specification we need atomic sentences for annotating requirements with a unique Actor, Process and Object. After checking the fulfillment of each atomic sentence of a requirement, the results should be put together in order to decide on the fulfillment of the original requirement.

**Use syntactic templates for specifying requirements!**

We have found using templates [22] for formulating requirements very useful. In this way we improved the quality of syntactic structures of our requirements which further improved the accuracy of the syntactic analysis by identifying Actors, Process verbs, Objects and Conditions.

Additionally, we have realized in our case study that there are already mature tools for linguistic analysis which we could use successfully. For different tasks in our approach we could use different tools. Stanford Lexical Parser and the Similarity Library offer a Java-based application programming interface (API) which enables us to invoke their functionalities from our own components. We were able to show that integrating these tools using the API is possible.

Last but not least we also see potential for improvement in our approach. In coarse-grained clustering as explained in section III.B.1), we have recognized that grouping into Actions, States and Rules by only considering the Process verb may cause problems because the verbs can have different meaning in different context.

We currently compute the similarity between requirements by considering the similarity between Actors, Process verbs and direct Objects. Including further indirect Objects (e.g. prepositional objects) into the similarity measurement could improve the clustering results.

As explained in the beginning of section III we propose an iterative process where we evaluate the results of each phase and decide on repeating the phase or continuing the process. We need more automated support for comfortably inputting our evaluation results into the process and for conducting further activities, e.g. changing the requirements, correcting the annotations manually, adding/removing elements into/from clusters and test plans, etc.

In our current approach we mainly concentrate on textual requirements. However our specification documents also contain numerous UML models which also should be considered in our approach. Formal object models based on UML 2.0 (e.g. UML4ODP) may be used to further analyze the informal requirements. Therefore, we want to further extend the approach by integrating current functionalities into a modeling tool, e.g. Enterprise Architect by Sparx Systems (www.sparxsystems.de) as depicted in Fig. 14. Another extension would be integrating our prototype into a state of the art requirements engineering tool which supports defining semantic attributes, e.g. DOORS by IBM (www.ibm.com, formerly Telelogic), IRqA by QA Systems (www.qa-systems.de) or Serena Dimensions RM by Serena Software (www.serena.com).

**VI. CONCLUSION & FUTURE WORK**

Specifying e-Id systems using the RM-ODP methodology inherently contains redundant requirements. This is good for the readers of the requirements documents for better understanding; however, the redundancies may cause unnecessary effort in acceptance testing where the requirements are checked for fulfillment by the developed system. For that reason we looked for a way how the redundancies and dependencies can be identified systematically. After identifying the redundancies and dependencies we aim at reducing the testing efforts for acceptance testing. For this aim several research questions were recognized.

We applied requirements clustering techniques for identifying redundancies between requirements. For that we first annotate requirements with semantic attributes like Subject, Verb and Object. Using the semantic attributes we can differentiate between requirements types Actions, States and Rules. Semantic attributes also enable the computation of similarity between each pair of requirements. The similarity computation results in a graph represented by a similarity matrix giving a similarity measure between 0 and 6. High similarity measures indicate potential redundancy which should be verified manually. By defining a threshold value we can group related requirements into fine-grained clusters. For each cluster efficient test plans can be designed by ordering related requirements from Actions, States and Rules.

Using our approach we can identify potential redundancies to some extent given by the semantic information of Subject, Verb and Object. Our approach does not only help in identifying redundancies, but also in designing efficient test plans by grouping related requirements in similar clusters. We could automate most of the activities in our process. This makes our approach less error-prone than manual activities. Also reproducibility is granted by automation. The iterative process enables to review the results of each step and make improvements.
Our contribution in requirements clustering is that we could apply clustering techniques for testing. To the best of our knowledge, existing approaches for requirements clustering do not address testing problems. We could also automate the clustering activities which are done manually in existing approaches. Our case study shows that the presented approach scales if applied in industrial context.

We want to gain further experience by applying our approach in other projects. We will adapt our requirements capturing process in order to fulfill the preconditions of the requirements clustering approach. Furthermore we want to experiment with further algorithms for clustering and similarity computations.

Currently our tool is just a prototype. We want to improve the functionality and usability of the prototype and integrate it into the requirements management tools we used. Also the iterative process should be implemented using user friendly interactions of the tool with the user.

VII. REFERENCES

URL: http://dsd.lbl.gov/~hoschek/colt-download