Analysis of Authorizations in SAP R/3

Untersuchung des Berechtigungskonzepts im SAP R/3 System

Manuel Lamotte

Master’s Thesis

Supervisor: Prof. Dr. Christoph Weidenbach
Second Reader: Prof. Dr. Andreas Lux

Saarbrücken, January 31th, 2008
Acknowledgments

I’m indebted to my supervisor Prof. Dr. Christoph Weidenbach who introduced me to the field of automated theorem proving in first-order logic. Without his continuous support, engagement and of course his patience with me this work wouldn’t have been possible.
I am also grateful to Prof. Dr. Andreas Lux for supporting me throughout my whole studies.
I would like to thank Roxane Wetzel and Martin Kaiser for their support in the form of proof-reading and comments.
Finally, I thank my fiancée Agathe Schubert for her help and patience.
Today many companies use an ERP (Enterprise Resource Planning) system such as the SAP R/3 system to run their daily business ranging from financial issues down to the actual control of a production line. These systems are very complex from the view of administration and authorization. Hence they include a high potential for errors. In this thesis I analyze the authorization concept of the SAP R/3 system as well as different business regulations and construct a corresponding model in first-order logic. This model can be used to check the existence of errors automatically, i.e. a contradiction between given authorizations and a valid business regulation. The tool I use for these checks is the theorem prover SPASS which has been developed at the Max Planck Institute for Informatics. I selected the purchase process as an example to explore the model construction because it is a typical constituent of the SAP R/3 system.

## Contents

1. **Introduction** ................................................................. 1

2. **The SAP R/3 System** ...................................................... 5
   2.1 Authorization Checks .................................................. 6
      2.1.1 Transactions ...................................................... 6
      2.1.2 Authorization Objects .......................................... 7
      2.1.3 Authorizations .................................................. 8
      2.1.4 Authorization Check Procedure ............................... 9
   2.2 The Authorization Setup .............................................. 10
      2.2.1 Authorization Profiles ....................................... 11
      2.2.2 Roles ............................................................ 12
   2.3 Business Processes ................................................... 14
      2.3.1 The Purchase Process ......................................... 14
      2.3.2 Purchase Authorization Objects ............................. 20
      2.3.3 Purchase Authorizations .................................... 22
      2.3.4 Authorization Check Procedure ............................ 22
   2.4 Business Policies ..................................................... 24

3. **Extracting Authorization Information** ............................... 27
   3.1 Authorization Checks ................................................ 27
   3.2 Authorization Setup .................................................. 30
   3.3 Purchase Process .................................................... 35
      3.3.1 Create a Requisition ......................................... 36
      3.3.2 Release a Requisition ........................................ 36
      3.3.3 Create an Order .............................................. 39
   3.4 Business Policies ..................................................... 39

4. **Foundations** ............................................................. 41
   4.1 First-Order Logic ..................................................... 41
   4.2 Notation .................................................................... 42

5. **Theory Development** .................................................... 43
   5.1 Abstractions .............................................................. 43
   5.2 Authorization Setup .................................................... 44
## Contents

5.2.1 Roles ................................................................. 45  
5.2.2 Profile Generation and Authorization Profiles ............... 46  
5.3 Authorization Checks .................................................. 47  
5.4 Purchase Process ...................................................... 48  
  5.4.1 Create a Requisition .................................................. 48  
  5.4.2 Release a Requisition ................................................. 51  
  5.4.3 Create an Order ....................................................... 54  
5.5 Business Policies ..................................................... 55  

6 Using the Theory ............................................................. 57  
  6.1 Theorem Proving with SPASS ......................................... 57  
  6.2 Plausibility Check ....................................................... 58  
  6.3 Experiments ............................................................. 67  
  6.4 Results ................................................................. 68  

7 Conclusion and Outlook .................................................... 69  

Directory Structure of the CD ............................................. 71  

References ................................................................. 72
List of Figures

1. Authorization Object Structure .................................................. 7
2. Authorization (Instance of an Authorization Object) .................... 8
4. Structure of a Composite Authorization Profile ............................ 12
5. Structure of a Single Role .......................................................... 13
6. Structure of a Composite Role ..................................................... 14
7. Purchase Process .......................................................................... 15
8. Screen to Create a Requisition ..................................................... 17
9. Screen to Release a Requisition .................................................... 19
10. Screen to Create an Order .......................................................... 20
12. Part of the System Trace of the Transaction ME51N ...................... 29
13. Excerpt from the Export of Roles Assigned to Users in text-format .. 32
14. Export of a Single Role through the Download Function ................ 33
15. Export of a Composite Role via Download Function ........................ 33
16. Excerpt of the Export of Profiles Assigned to Users in text-format ... 34
17. Export of Authorizations of a Profile in text-format ................. 35
18. Characteristics in the Characteristics Class FRG_EBAN .................. 37
19. Release Strategy Setup ................................................................. 38
20. Authorization Setup Scenario .................................................... 59
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Authorization Values Denoting the Type of an Activity</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Value Formats in an Authorization</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Checked Authorization Objects</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Authorization Field Instances of the Authorization Objects listed by Table 3</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Identified Authorization Objects in Transaction ME51N</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>Identified Authorization Objects in Transaction ME54N</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>Characteristics Class for Purchase Requisitions</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>Identified Authorization Objects in Transaction ME21N</td>
<td>39</td>
</tr>
<tr>
<td>9</td>
<td>Identified Authorization Objects in Transaction ME21NOFF</td>
<td>40</td>
</tr>
</tbody>
</table>
1

Introduction

Enterprise Resource Planning (ERP) systems are build to integrate all facets of the business across a company including areas like finance, planning, manufacturing, sales, or marketing. The broader the functionality of such a system, the larger the number of users, the greater the dynamics of a company, the more complex is the administration of the authorizations. In particular, this applies to the SAP R/3 system offered by SAP. SAP is the worldwide market leader in the ERP software market with a market share of 27 percent in the third quarter of 2007 [SAP07]. Today there are over 100,000 installations including SAP R/3 and the subsequent versions of this system [Osw06]. Although SAP R/3 is not the newest release of SAP’s ERP software, the largest part of the total number of installations is the R/3 system [Erb02]. The actual release SAP ERP 6.0 has a slightly different architecture than the R/3 system, but the complexity of the administration of authorizations applies both to SAP R/3 and SAP ERP 6.0.

Authorizations in SAP R/3 and ERP 6.0 are used to protect data against random or malicious manipulation, damage or against unintended misuse. They also ensure the confidentiality of internal enterprise information. Authorizations are set up in SAP R/3 according to an individual and customized concept that defines the access rights to data or functions for each user in SAP R/3 and ERP 6.0. The administration of authorizations is necessary because companies continuously overwork their business processes and their organization. Therefore, the concept and the authorizations have to be adjusted, too. Today, changes will only be tested in a separate test system before the change applies to the productive system. But in any case, a change of authorizations includes a high potential for errors.

This leads to the idea of this thesis to provide the possibility of automatic proofs of the authorizations in a R/3 system with the notion to catch errors automatically. An exact formal description of the system is the basis for automatic proofs. A formal description (based on a formal language) is defined by precise mathematical formulae which are also machine processable. I decided to use first-order logic as a basis for the formal system specification because first-order formulae can be easily converted and used as an input for the theorem prover SPASS [WSH+07] provided by Weidenbach et al.

Authorizations in a company come on three layers. First, there is the business
process layer where a company decides on regulations and constraints. A typical business process is the purchase process which is later used as a case study. Companies usually have policies restricting different actions or prohibiting actions to certain persons. A business policy might be that the creator of a purchase request and the buyer must be two different persons. Hence, the purchase requisitioner for example is not allowed to hold authorizations granting access to create orders in the R/3 system.

The second layer is the authorization setup layer. It exactly defines for a user whether he/she is authorized to execute a certain (trans)action or not. Authorizations in SAP R/3 are assigned to the users in the form of roles and profiles.

Third, there is the authorization check layer where the authorization checks take place. The authorization checks are defined in the program code of the program associated with the functions available in SAP R/3. Hence, the authorization check layer is the code layer where the individual authorization checks are implemented.

The goal of this work is to prove or disprove that the authorization setup, the authorization check layer and the business layer specification including the business policies are in sync. The different layers as well as the business policies are formalized and modeled in first-order logic in order to prove the conformity of the authorizations with respect to the policies in the positive case or otherwise to obtain a proof for the violation of the business policies.

The construction of the theory strongly follows the SAP R/3 system structure. Therefore, the theory can be extended to other business processes.

A subsequent work task is the automatic extraction of the authorization data and the automatic generation of formal formulæ.

The SAP R/3 installations of both the Max-Planck Society and the Fachhochschule Trier provide the basis for the research. The possibility to create test scenarios was given in a test system of the Max-Planck Society where I had access to. The configuration of the SAP R/3 test system is the same as it is in the productive system of the Max-Planck Society.

SAP offers a tools collection for Governance, Risk and Compliance (GRC) which is related to our work. The collection consists of the tools Virsa Compliance Calibrator, Virsa Role Expert, Virsa FireFighter and Virsa Access Enforcer.

The Virsa Compliance Calibrator is an internal, permanent system auditor. The core of the calibrator is a set of rules which cover about 200 risks that are derived from several thousand standard transactions in SAP software. The Compliance Calibrator handles testing and enforcement of segregation of duties, performs analysis of access and duty violations, and provides remediation and mandatory risk analysis. If a potential risk or violation has been detected during runtime, then it will be reported by the Compliance Calibrator in real-time to the responsible person. Furthermore, the Calibrator checks potential violations against the stored rules using a preventive simulation before the assignment of authorizations takes place. In addition to the system auditor function the Compliance Calibrator also
suggests solutions, for example, the risk of misuse can be lessened by separating
duties when checking invoices.

The Virsa Role Expert is a profile generator that can be used to create and man-
age authorization roles. It provides an easy-to-use interface and offers the content
in a language that both IT and user departments understand. Business people de-
fine the functions and the IT responsibilities set the technical authorizations. Hence,
the roles are well documented for IT and non-IT users. The Virsa Compliance
Calibrator checks that the new roles follow all required specifications based on its
rule database before the authorization will be assigned.

The remaining tools Virsa FireFighter and Virsa Access Enforcer are not inter-
esting for this thesis because they aim to other fields of the SAP R/3 authorization
part. These fields are not connected to the topic and the goal of this thesis.

Using the theory which has been presented in this work I successfully proved that
the authorizations of SAP R/3 and the (formalized) business policies are in sync.
Different experiments showed that errors resulting by a change of authorizations
could be found automatically using the theorem prover SPASS. But the experiments
also yield some limitations. Some conjectures could not be proved automatically
with SPASS. But this s just a limitation of the theorem prover. Nevertheless, such
conjectures are manually provable using a special model for the theory (the so
called minimal model) but this is future work.

The experiments have also shown that the mapping is not just yet optimal. Any
large SAP R/3 systems includes a lot of occurrences of numbers and also compar-
isons between the numbers. Even the example of this thesis – the purchase process
which I used as a case study – works with numbers in the form of money amounts.
Any money amount is represented by a constant in the theory. The mapping of
arithmetic and comparisons for between monetary amounts is not yet available
by SPASS. An extension to the theory which enables, for example, the evaluation
whether a number is in a specific interval would improve the conformity of the
theory with respect to the real SAP R/3 structure a lot.

This thesis is structured into six chapters. After this introduction, the chapter 2 ex-
plains the constituents related to the authorization setup and authorization check
layer of the SAP R/3 software. Furthermore, it introduces the business processes
and describes the notion of business policies.

The chapter 3 explains the way used to extract the relevant authorization infor-
mation from a given SAP R/3 system instance. It also describes the extraction of
the business policies of a company.

The chapter 4 Foundations, defines the syntax of the first-order language which
is required in order to construct a theory mapping the SAP R/3 system.

The theory is constructed in first-order logic by taking the extracted data from
a concrete snapshot of an existing SAP R/3 system together with the business
policies. chapter 5 shows and explains the formulae of the theory.

Chapter 6 first shows that the constructed theory is valid with respect to different
business policies. Then, assuming that there is a violation of a business policy
in the SAP R/3 system, an experiment demonstrates that SPASS can prove the violation. Further experiments also show possible limits of the automatic proving with SPASS.

The last chapter summarizes the general results, points out open questions and gives an outlook to further work.
The SAP R/3 System

During the last decades a lot of mid-size and large companies introduced ERP (Enterprise Resource Planning) software like the SAP R/3 system.

From the organizational view, a single instance of an SAP R/3 system can map one department of a company (e.g. supply chain management) as well as worldwide corporations with a lot of departments and subsidiaries. The administration of authorizations in such systems is extensive.

The base structure of SAP R/3 is function oriented and consists of multiple core areas which in turn are divided into different modules:

- Financials: Financial Accounting (FI), Controlling (CO), Treasury (TR), Investment Management (IM), Enterprise Controlling (EC), Project System (PS), Real Estate Management (RE)
- Logistics: Plant Maintenance (PM), Material Management (MM), Quality Management (QM), Production Planning (PP), Sales and Distribution (SD), Customer Service (CS), Logistics Execution (LE)
- Human Resources (HR)
- Workflow (WF)

There are also different modules combining the application modules with additional industry-specific functionality. These modules are prefixed with IS (Industry Solution). The following list gives an excerpt of the variety of different industry modules: Health-care (IS-H), IS-ADEC (Aerospace and Defense), Oil (IS-OIL), Telecommunications (IS-T).

Before an SAP R/3 system becomes productive it is necessary to do so called customizing to meet the business needs. Therefore, two R/3 implementations as used by two different companies will typically never look the same.

During my work I had access to the R/3 installations of the Fachhochschule Trier and the Max-Planck Society and I used both systems to acquire information about the authorization mechanism and the authorizations.

The Fachhochschule Trier uses the (currently older, but still supported by SAP at this time) release *SAP R/3 Enterprise 4.7*. The core component including the authorization components is called *SAP Web Application Server* and is installed in the kernel release *640* on this system.
On the other hand the Max-Planck Society actually don’t use a R/3 system. They use the current release of SAP (newer than R/3) which is called *SAP ECC 6.0*, *mySAP ERP 2005* or *SAP ERP 6.0*. Here the core component is called SAP NetWeaver; the version used in the Max-Planck Society is *SAP NetWeaver 7.0 (2004s)*.

Concerning the authorization concept I didn’t find any differences during my examinations of these two releases. For this reason it doesn’t matter whether to speak from SAP R/3 or SAP ECC 6.0 in this thesis. As a consequence I just refer to SAP R/3 from this point.

This chapter provides the technical base for the theory development in chapter 5. The first section 2.1 explains the different components used for the authorization checks and describes also the relationship between these elements in the procedure of an authorization check.

The authorization setup represents the users with their authorizations. These authorizations can be assigned to the users using different kinds. Among the definition of different elements like single and composite roles the section 2.2 describes the difference and the relationship between roles and authorization profiles.

The third section 2.4 introduces into the concept of business policies and also gives some examples for business regulations.

The last section 2.3 eventually explains the business processes. An example of a process is the purchase process. Some screenshots are presented in order to explain and differ a number of terms from the SAP world which are essential to understand this process. This theoretical economic background is further necessary to understand the design decisions we made to develop the formal first-order logic model introduced in chapter 5.

2.1 Authorization Checks

The following section introduces the main components used for the authorization checks.

The first section 2.1.1 starts with the description of the term of a transaction. In general, transactions are known as database transactions. This basic understanding will be further specified by establishing a connection between transactions and authorization checks. Section 2.1.2 describes the notion of authorization objects to protect data and functions. Section 2.1.3 describes authorizations and explains the difference between authorization objects and authorizations.

In the end, the objects and terms introduced before are depicted together in a complete authorization check procedure.

2.1.1 Transactions

Transactions are known from database transactions. A transaction consists of multiple individual operations which either succeed or fail as a group. It must fulfill
the so called ACID criteria: Atomicity, Consistency, Isolation and Durability. This concept also applies to transactions in SAP. The program (code) where the authorization checks are implemented is called a SAP transaction. Typically, there are multiple individual authorization checks implemented. According to the ACID criteria for database transactions all single authorization checks must be successful in order to successfully execute the transaction. If one of the single checks fails then the overall transaction fails, too. The procedure of authorization checks in a transaction is further described later in section 2.1.4.

The execution of a transaction means in SAP the execution of the appropriate program. There is a unique identification code for every program which needs to be protected in any form. This code is called transaction code. The term transaction is always used to refer to the program itself. Short statements like “the transaction xy” denote the transaction where xy is the transaction code for that program.

The authorization checks often differ on the number for each run of a transaction. This refers to the fact that the individual checks depend on the data which has been entered by the user. This is, for example, implemented using traditional if-else statements for invoking or not invoking an authorization check.

Assume that the implementation of a number of authorization checks for a concrete transaction doesn’t change and the number of checks is also the same for two runs. Then the checks themselves are identical (the same thing is checked) in each run of the transaction, but the data is not. For example, a check in a transaction could be the verification of the users’ department. Then the basis in each run is equal (the department check), but the checked data is different because not all employees in a company belong to the same department.

2.1.2 Authorization Objects

Authorization objects are used to protect functions or data in the SAP system. Every authorization check occurring in a transaction checks the associated authorization object. The object itself is a logical entity used to group one or more value fields requiring authority checking within the system. There are one up to ten value fields in one object. Each field is a container for a value. The structure just described is illustrated in Fig. 1.

![Fig. 1. Authorization Object Structure](image-url)
The release 4.6 of SAP R/3 contains already about 900 predefined authorization objects. If none of these objects is suitable for an authorization check it is possible to define additional authorization objects which are liable to the structure described before.

### 2.1.3 Authorizations

The combination of the authorization object and a field value constitutes the authorization, that is an instance of the authorization object. This leads to the structure depicted in Fig. 2.

**Fig. 2. Authorization (Instance of an Authorization Object)**

The use of the term “authorization” leads to confusion because in general an authorization is the permission of a user to execute a function in a system. But in this context, authorizations are not user authorizations. An authorization is here (and in the SAP environment) a special notion for the combination of the authorization object and the values for its fields.

A value which is placed in a value field of an authorization object can be a single value or a value range. There are a number of numerical values which denote the type of an activity. The following Table 1 shows the action codes which occur in this thesis.

<table>
<thead>
<tr>
<th>Authorization Field</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTVT (Activity)</td>
<td>01 - Create</td>
</tr>
<tr>
<td></td>
<td>02 - Change</td>
</tr>
<tr>
<td></td>
<td>03 - Change</td>
</tr>
<tr>
<td></td>
<td>23 - Maintain</td>
</tr>
<tr>
<td></td>
<td>61 - Export</td>
</tr>
</tbody>
</table>

**Table 1. Authorization Values Denoting the Type of an Activity**

Additionally, values can use the wild-card characters “?” and “*”. The wild-card character “?” stands for a single and “*” stands for any combination of characters.
of any length. A value which contains wild-card characters is called *regular pattern*. Regular patterns consisting of a combination of wild-cards and an interval provides a lot of possibilities for authorizations. The following Table 2 depicts some formats and patterns and is taken from [SAP01].

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>Interval: values 1, 2, and 3</td>
</tr>
<tr>
<td>S_USER*</td>
<td></td>
<td>Single value: any character format beginning with “S_USER”</td>
</tr>
<tr>
<td>AB</td>
<td>C*</td>
<td>Interval: all values beginning with AB, AC, ... or B or C</td>
</tr>
<tr>
<td>0</td>
<td>9*</td>
<td>Interval: any numeric value</td>
</tr>
</tbody>
</table>
| ME5?N |     | Any string of the form ME5xN where x is a single character or an unary numerical value. Matches, for example, a multitude of purchase transactions codes: ME51N, ME52N, ME53N, ME56N, ...

Table 2. Value Formats in an Authorization

Value ranges for numbers and letters are specified separately. The combined use of numbers and letters like “0 to Z*” which would include all numbers and letters is admissible but not encouraged by SAP [SAP01, p. 92].

The built-in authorization policy of the SAP R/3 system forbids all actions which are not explicitly permitted by an authorization. Therefore, authorizations can only grant permissions but not restrict or forbid them.

### 2.1.4 Authorization Check Procedure

Authorization checks occur whenever a user requests access to a particular transaction. The user’s credentials are checked against the request in such a check; if the authorizations match (both the authorization object and its values must match), the user is permitted to access the information which is protected by the authorization object.

The authorization check consists of two parts: (i) it checks the presence of the required authorization in the authorization profile of the users’ master record (for this purpose only the authorization object name is compared), (ii) the comparison of the required value(s) with the value(s) present in the value field(s) of the authorization. The check (i) is successful if the right authorization object is available in the users’ profile. The second check (ii) succeeds if all value fields with the corresponding values of the object match to the required fields and values (AND-combination). Matching means that one of the following conditions holds:

- The present value in the value field is the wild-card character *. The star matches to any required value.
- The present value in the value field and the required value match exactly, i.e. they are equal.
• The required value lies within the interval which has been specified as a value for the value field.
• If the value in the value field consists of a regular pattern, then the required value must satisfy the condition of the expression. For example, an expression like $SA*$ matches any value starting with the characters $S$ and $A$, followed by any combination of characters of any length or no character.

A successful authorization check requires that the authorization object check (i) and all single field/value checks (ii) succeed. If one of these checks fails, then the overall check of the authorization fails.

The authorization check itself always follows the structure just described but there are two cases when an authorization check is triggered in a transaction:

• **Check at the start of a transaction**
  The authorization object $S_TCODE$ is always checked at first before the actual program will be executed. This verification is set up by the system and can’t be turned off. It is possible to let the system check one arbitrary further object before the actual program starts.

• **Check during the progress of a program**
  This is the default case wherein all authorization checks are located in the program code. The integration of an authorization check in a program is done by using the `AUTHORITY-CHECK` statement. Whenever the system reads this keyword during a program execution it checks the associated authorization object with its fields except the check of this object has been disabled. The state (disabled or not) is indicated by the so called check indicator.
  It is possible to check one object repeatedly in a program or to group several different objects to protect special data or a specific part of the program.

  In earlier times, the authorization objects had to be manually identified in the program code. Today this is easier because there are database tables.

### 2.2 The Authorization Setup

The authorization setup holds the information about the users and their authorizations in the SAP R/3 system. In other words, it is the layer on which authorizations are assigned to users.

The authorizations can be assigned to the users using different ways: using so called roles or authorization profiles or using a combination of both ways. Effective authorizations are always and only stored in the users authorization profile. Therefore, any authorization which has been assigned to a user on the base of a role implies an assignment of the auto-generated authorization profile.

The following sub-sections describe authorization profiles and roles and also explain the differences between these two notions. Furthermore, the question why

---

1 The table USOBX holds the default values for authorizations delivered by SAP. The customer table USOBX,C is initially (during the overall system setup) filled with the contents of the table USOBX.
it is clever to use roles for authorization assignments instead of (direct) assignments of authorization profiles will be answered.

### 2.2.1 Authorization Profiles

An authorization profile is a group of authorizations. If an authorization profile has been assigned to a user then he/she is authorized to access all transactions/functions/data granted by the authorizations in the profile.

SAP distinguishes between simple (or single-level) profiles and composite profiles.

- Simple (single-level) profiles have authorizations on one level. Nested levels are not possible. The following Fig. 3 shows the structure of a simple profile. The assignment of a simple profile to a user results in the assignment of all authorizations to the user which are contained in the profile.

![Simple Auth. Profile](image)

**Fig. 3.** Structure of a Simple (Single-Level) Authorization Profile

- Composite profiles can contain single profiles or other composite profiles. The notion is to reduce the maintenance effort with a better structure in the authorizations part. Therefore, a composite profile groups different simple and/or other composite profiles together. The following Fig. 4 depicts the structure of a composite profile.

From the technical side there is no difference between the assignment of a simple profile and a composite profile to a user. The assignment of a composite profile just results in assignments of all authorizations which are contained at any level (union). Authorizations can only grant access to transactions/functions/data but they can’t forbid the access. Therefore, conflicts resulting from the union cannot occur because the union of access grants implies again access grants and never the denial to access something. If, for example one profile contains only the permission to read some data and another profile contains the permission to write then the result is the permission to read and write the data.

The existence of the composite profiles remains from earlier releases of SAP R/3 without the support for roles (see next section). SAP strongly recommends to only use the concept of roles. [SAP01] p. 88] The main reason concerns the ability to structure. Roles offer more structure features than profiles and are the newer concept.
2.2 The Authorization Setup

2.2.2 Roles

The so-called Role Based Access Control (RBAC – also called role-based security) is an essential concept in SAP R/3. Role-based security is a form of user-level security where the application doesn’t focus on the individual user’s identity; but rather on a logical role they occupy.

The concept of roles is important to SAP R/3 because roles offer great possibilities to compose structures. It is possible to create single roles as well as composite roles and even inheritance between roles is supported. The tool SAP Profile Generator\(^2\) is used to create single and composite roles. The name Profile Generator comes from the fact that an authorization profile will be generated for each role and also after every change of the role. The generation is required because authorizations are only effective for a user if he/she holds them in his/her master record. If a user holds authorizations then he/she in fact holds one or more authorization profiles which group the authorizations.

If a role inherits properties from another role then the parent role is called template role. For example, a template role can inherit the contained authorizations to the child roles. Just like inheritance in programming languages, changes to the template role will be automatically applied to the derived roles. Template roles are often used to define roles having the same functionality (i.e. allowing to execute the same transactions), but for different organizational levels such as company code, purchasing organization, etc.

\(^2\) Transaction PFCG

Fig. 4. Structure of a Composite Authorization Profile
Single Roles

The structure of a single role is similar to the authorization profile, i.e. it also contains authorization values. Fig. 5 depicts the structure of a single role.

![Single Role Diagram]

Fig. 5. Structure of a Single Role

Whenever a single role is created using the Profile Generator the appropriate authorization profile should be generated, too. A user effectively holds only the authorizations which are present in the (generated) authorization profile which has been assigned to the user. Therefore, the assignment of a role to a user without the existence of the generated authorization profile is useless. This is due to the fact that authorization checks always and only compare the required authorizations with the authorizations present in the authorization profile of the users’ master record. Therefore, an authorization check will fail if the required authorization is only present in the role which has been assigned to the user and not in the users’ authorization profile.

A difference to the (direct) authorization profile is that changes to the authorizations of a user which are grouped in auto-generated authorization profiles must take place through the role definition. It isn’t possible to change values directly in (auto-)generated profiles as it is in direct authorization profiles.

Composite Roles

Composite roles are used to reduce the maintenance effort. They can bundle an arbitrary number of single roles. The structure of a composite role is shown in Fig. 6. In contrast to a composite profile, composite roles can make use of the feature of inheritance. Therefore, the structure capabilities are better for roles than for direct profiles.

The notion of a composite role is that it should reflect the position of a user with its tasks and responsibilities in a company whereas a single role holds all authorization information needed to perform one concrete task.
2.3 Business Processes

A business process is a sequence of interrelated steps which solve a particular issue. It can be part of another process and can also contain other processes or initiate one.

There are three types of business processes:

1. **Management processes** govern the operation of a company. Typical management processes include “Corporate Governance” and “Strategic Management”.

2. **Operational processes** are processes that constitute the core business and create the primary value stream. Typical operational processes are Purchasing, Manufacturing, Marketing, and Sales.

3. **Supporting processes** support the core processes. Examples include Accounting, Recruitment or IT-support.

Management processes are indirectly modeled in SAP R/3 because they typically partly initiate or consist of other non-management processes. On the other hand, the operational and supporting processes are mapped in SAP R/3 by sequences of different transactions.

In general, all business processes that are mapped by a number of SAP R/3 transactions use the same authorization check mechanism. The analysis of all business processes which are mapped in the SAP R/3 system would go beyond the scope of this thesis. Therefore, I selected the purchase process as a case study for my work. This process is a typical constituent of the SAP R/3 system. The following sections introduce (the steps of) the purchase process. In this thesis I focus on the analysis of authorizations in this operational process. It is mapped in SAP R/3 and consists of several transactions. The following sections describe these transactions as well as the essential data which must be entered to complete the transaction.

2.3.1 The Purchase Process

The purchase process is used as a case study to analyze the authorizations in SAP R/3. It consists of several steps which are depicted in Fig. 7. However, this thesis
only focuses on the creation of a requisition, the release and the eventual order (marked by a box in the Figure) in order to keep the example straightforward and comprehensible. The creation of the requisition as well as the creation of the order is mapped by exactly one transaction in SAP R/3. The release transaction is mapped by two transactions, one to view the requisition and the second to apply the release.

The following sections describe at first the essential data which must be entered in the transactions. In the end, a screenshot is presented for each transaction in order to give a rough idea of the appearance of the purchase transaction screens in SAP R/3.

Create a Requisition

An R/3 purchase requisition document is created by a department and the purchasing group, respectively to request the purchase of goods or services. Such a document has some mandatory fields – the fields which must be filled in order to proceed the transaction. Therefore, the values of most of the required fields are subject to authorization checks. The exact enumeration of fields related to authorization checks is presented in the subsequent sections.

The first mandatory field is a selection field denoting the document type – the type of the requisition object. Typical values are NB which stands for regular or “normal business”, RSU which stands for re-supply or SH for special handling. However, the possible values in a concrete R/3 system depend on the customizing settings. In general, customizing is the adjustment of a default SAP R/3 system to match the business needs of a company.
The next mandatory field is the *item number* which is located in the item area (see Fig. 8). It prohibits the creation of empty requisitions.

Each item which is going to be purchased is destined for a concrete *plant*. Therefore, the plant is a mandatory field, too. The value depends on the structure of the company (and consequently of the customizing settings) and is typically prefilled. An empty value suggests a non-complete customizing.

Furthermore, each item must be assigned to a concrete purchasing group. The term purchasing group is somewhat misleading because in the Max-Planck Society a purchasing group is often composed of just one person.

Each item requires the input of either a material number or a short description of the item. The material number is used in order to load the material related data from the material master record which is stored in the database. Further mandatory fields like the material description, material group and the unit of measure are then prefilled automatically. The second option is to use just the material description. Then the fields like material group and unit of measure have to be filled manually.

Further required fields are, of course, the quantity of each item which is going to be purchased, the delivery date and a price valuation.

The account assignment category is also mandatory. It determines which account assignment details are further required for an item (for example, the specification of a cost center or an account number).

The screenshot depicted in Fig. 8 shows most of the previous mentioned fields of the screen on the basis of the requisition for a Porsche sports car.

**Release a Requisition**

Release procedures for requisitions are used in SAP R/3 to approve requisitions which exceed a certain budget limit before they can be converted to an order.

The SAP R/3 system uses so called *release strategies* to achieve such approvals. They have to be defined in the customizing of the purchasing before the execution of any purchase activity.

In order to provide a wide variety of possibilities the release strategies make use of the so called *class system* which is a general component of SAP R/3 system that is not only used for release strategies. This class system is used to describe and classify any objects, for example, materials or requisitions, through *characteristics*.

A release typically consists of several single release steps which have to be specified in the release strategy. Each step is defined by its *release code*. These release codes are again grouped in the *release group*. Both elements will later be explained in detail. The last constituent of the release strategy are the *release indicators* indicates the approval state of a requisition and defines the possible actions after a certain release step.

Let’s start with the characteristics class. It has a unique name and it describes types of objects by their characteristics. The characteristic itself is again basically specified by a *property* and its *value(s)*. For example, every car has a certain
weight and a color. Then “car” is the class and “weight” and “color” are the properties. Possible values are, e.g., “1000 kg” for the weight property and “blue” for the color property. Then any object that satisfies the properties, i.e., any blue vehicle weighing 1000 kg will be classified as a car. It is further possible to define multiple values or a regular pattern using the wild-card characters “?” and “*” as usual as a value. Recall the previous example for the class “car”. Assume that the value for the property weight is “1??? kg” (only numerical values allowed) and the color is “blue”. Consequently, all blue vehicles weighing between 1000 and 1999 kg will be classified as cars. At the time of characteristic definition it is not mandatory to enter a value or a number of values, but values can be entered for suggesting when the characteristics class containing the characteristic is used in the release strategy setup. A characteristic class describing requisitions has, for example, different properties where each property is associated with an input field of a requisition.

The next constituent of a release strategy is the release code. It is simply a two-digit value and it is used in order to grant the release of a requisition. In other words, it denotes the state of a requisition in which it has been released with a certain code.

Every release code and the release strategy itself must be assigned to a release group. The release group is used to group different purchase requisition types with

Fig. 8. Screen to Create a Requisition
individual release codes. In other words, a release group arranges release strategies into groups according to the kind of purchase requisition. Consequently, one can control which user is able to release which purchase requisition types. Furthermore, the release group is linked to a characteristics class.

The release indicator indicates the approval state of a requisition. It specifies, which activities are allowed after the release with a certain release code. Activity means, for example, changes to the requisition after the release procedure has started or future actions like the creation of an order.

The release strategy combines all the previously mentioned items. It is labeled by two characters and has a short description. A characteristics class is assigned to the release strategy to decide when the strategy applies. For this purpose, the release strategy needs concrete values in the characteristic fields of the characteristics class. The values must either be entered manually or adopted from a list of possible values (created in the class definition). Moreover, it holds the information, which release codes are necessary and in which order they are required to release a requisition completely. It also controls which actions are allowed after a certain release step, i.e. when the requisition is in a certain release state.

Assume for the rest of this section that the customizing of the purchase area is set up correctly which is the typical situation in a productive SAP R/3 system. A release strategy then applies to a requisition if the conditions defined in the strategy (by the characteristics class) are satisfied. The screenshot depicted in Fig. 9 shows the release screen for such a requisition. It continues the previous example of the Porsche sports car. The active release strategy is VF in this example, which stands for a management release. It consists of two single release steps: (i) the group leader and (ii) a person of the management, e.g., the director.

The permission to execute a (general) release step depends on the following conditions:

- The actual step must be active. Active means, that it doesn’t depend on the completion of other steps occurring before in the line of uncompleted release steps. Fig. 9 shows that the release step W2 is not yet ready, it can only be executed after the step W1 – which is currently active – has been completed.
- The releasing person needs specific authorizations in its authorization profile. The required values are defined in the release strategy settings.

Create an Order

The order is the request to the supplier or another plant of the company to deliver the requisitioned material or services under the terms and conditions agreed before.

In practice there are often negotiations with possible suppliers about the exact prices, conditions and perhaps possible rebates after the release of a requisition. If the price or the condition changes, it is typically a minimal change. A big change results in the re-appliance of the release strategy, i.e. either the changed item or the overall requisition has to be released again, depending on the release settings.
Assume that the release has succeeded and the conditions have not changed. A released requisition can be transformed to an order (which is connected to the requisition). In contrast to the requisition the order requires at least the following additional information to the item(s):

- The accounting settings which means that the item will be booked on a specific cost center, project, etc. The selection of an accounting setting in turn leads to further fields. In this thesis I always used the cost center accounting which requires the specification of the G/L account\(^3\) and the cost center.
- The vendor.

The Fig. 10 depicts an example of an order. The final step in a typical purchase process chain is eventually to print out the order and send it to the supplier.

The previously presented way to create orders is a default method. However, companies often need additional ways to create orders because the run through all steps of the complete purchase process often takes a lot of time. The solution is to have extensions of the SAP R/3 system allowing alternative ways to create orders. A typical example is an extension which allows direct orders but does not

---

\(^3\) G/L account = General Ledger Account
Fig. 10. Screen to Create an Order

require a released requisition. However, these direct orders are typically restricted to a certain type of material and with a limit on the monetary amount. Assume that direct orders are allowed for office equipment. The create order transaction presented before is used as usual, but the role granting authorizations to buy office equipment is slightly different to the one granting the default buyer authorizations and consequently only orders for office equipment material can be created.

2.3.2 Purchase Authorization Objects

Each transaction in SAP R/3 includes a number of different authorization checks. The transaction code is checked through the authorization object S_TCODE for each transaction before the actual transaction starts. Remember, there can be a check of one further object at this time. The object checked in the create requisition transaction is M_BANF_WRK, in the release transaction that is M_EINK_FRG and in the order transaction that is M_BEST_EKO. All further checks of authorization objects are implemented in the program code using the AUTHORITY-CHECK statement.

The Table 3 shows the authorization objects which are subject to authorization checks in the purchase transactions in SAP R/3. The first two columns denote the
appropriate authorization object with its fields. A checked object in the transaction is marked by an ‘X’.

<table>
<thead>
<tr>
<th>Auth. Object Name</th>
<th>Field</th>
<th>Create a Requisition</th>
<th>Release a Requisition</th>
<th>Create an Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_TCODE</td>
<td>TCD (Transaction Code)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_BANF_WRK</td>
<td>ACTVT (Activity)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WERKS (Plant)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>M_BANF_BSA</td>
<td>ACTVT (Activity)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>BSART (Document Type)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>M_BANF_EKG</td>
<td>ACTVT (Activity)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EKGRP (Purchasing Group)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>M_EINK_FRG</td>
<td>FRGGR (Release Group)</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FRGCO (Release Code)</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>M_BEST_EKO</td>
<td>ACTVT (Activity)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>EKORG (Purchasing Organization)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_BEST_BSA</td>
<td>ACTVT (Activity)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>BSART (Document Type)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_BEST_WRK</td>
<td>ACTVT (Activity)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>WERKS (Plant)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_BEST_EKG</td>
<td>ACTVT (Activity)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>EKGRP (Purchasing Group)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 3. Checked Authorization Objects
However, the Table only shows the authorization objects which are directly associated to the purchase transactions. There are some further authorization checks but they are not directly connected to the main task of the purchase transactions. These checks occur only sometimes which means that the check depends on the concrete selection of the user in the transaction. The authorization checks are implemented modular. Every SAP R/3 module has its special authorization objects which protect certain data and functions. For example, when a user uses the input assistance dialog to find specific data matching some criterion for a requisition/order, then the data must be exported from the result window into the requisition/order. In this case, the objects $S\_GUI$ and $S\_ALV\_LAY0$ are additionally checked to verify that the user is authorized to export the data which has been found.

### 2.3.3 Purchase Authorizations

Authorizations are instances of authorization objects which include the field values. Table 4 lists possible values for the authorization object fields of the purchase transactions. Some values like the transaction code are specified by the R/3 system (i.e. only the values $ME51N$, $ME52N$ and $ME53N$ are possible to work with requisitions) and other values like the purchasing group are freely selectable (according to the customizing).

A lot of authorization objects have the second field $ACTVT$ additional to the main field which protects certain data. This second field controls the type of the task. There are combinations of authorizations which are counterproductive since the final access to a transaction consists of several checks. For example, it makes no sense to combine single authorizations where one authorization just allows to view (activity 03) while the other authorization only allows to create (activity 01) because the final access to the transaction depends on all objects checked for this transaction. For the previously mentioned example, the access to the transaction will be denied.

### 2.3.4 Authorization Check Procedure

The authorization check procedure is basically the concatenation of all authorization checks which occur during the execution of a transaction. The access is only granted if all single authorization checks have been passed successfully.

The example for a complete procedure with its individual authorization checks is here the creation of a requisition. The order of the authorization checks is depicted in Fig. 11. At first, the objects $S\_TCODE$ and $M\_BANF\_WRK$ are checked when a user is going to create a requisition. These checks are executed automatically by the R/3 system. The next checks are $M\_BANF\_BSA$ and $M\_BANF\_EKG$ which check the document type of a requisition and the purchasing group, respectively. Further checks depend on the selection of the user in the transaction.

In order to succeed all these checks, the user needs the right authorizations in his/her authorization profile. Provided that the required authorization objects are
<table>
<thead>
<tr>
<th>Authorization Fields</th>
<th>Possible Values</th>
</tr>
</thead>
</table>
| **TCD (Transaction Code)** | ME51N (Create a Requisition)  
ME52N (Change a Requisition)  
ME53N (View a Requisition)  
ME54N (Release a Requisition)  
ME21N (Create an Order)  
ME22N (Change an Order)  
ME23N (View an Order) |
| **WERKS (Plant)** | Depends on the company, e.g.,  
INFO (MPI for Informatics) or  
SOFS (MPI for Software Systems) |
| **BSART (Document Type)** | NB (Normal Business, default)  
further (non-default) values depend on customizing settings |
| **EKGRP (Purchasing Group)** | The code depends on the customizing settings, e.g.,  
I26  
P13 |
| **FRGGR (Release Group)** | The release group depends on the customizing settings for the release strategy, e.g.,  
01 |
| **FRGCO (Release Code)** | The release code depends on the customizing settings for the release strategy, e.g.,  
W1  
W2 |
| **EKORG (Purchasing Organization)** | The purchasing organization depends on the customizing settings, e.g.,  
INFO (MPI for Informatics) or  
SOFS (MPI for Software Systems) |
| **ACTVT (Activity)** | 01 - Create  
02 - Change  
03 - Change  
23 - Maintain  
61 - Export |

**Table 4.** Authorization Field Instances of the Authorization Objects listed by Table 3
Fig. 11. Process of Authorization Checks in Transaction “Create a Requisition”

present in the users’ authorization profile, the access depends on the comparison of the required and present field values. The access to the transaction is forbidden if the field value of one of the authorization objects listed in Fig. 11 is missing in the authorization profile or if the comparison of the values fails. This comparison is a matching comparison, i.e., the present value is either equal to the required value, it lies within the specified interval or it matches to the pattern which describes the conditions. In case of a missing authorization in the user profile the access is denied to the transaction anyway.

If all authorizations which are required by the transaction to create a requisition are present in the user’s profile, then he/she is able to perform the transaction in at least one instance. One instance means that the user is able to execute the transaction with certain values which are specified by his authorizations. For example, a user which has access to the plant INFO (because the value of the corresponding object M_BANF_WRK in his authorization profile is INFO) is only able to create/view/change requisitions for this plant. He/She cannot access other plants.

2.4 Business Policies

Business policies are constraints on the business. According to the Business Rules Group [1100] a business policy or business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. Much of the industry’s
understanding of business policies has been historically shaped by the Business Rules Group, which grouped business rules in one of four categories:

- **Definitions of business terms**
  The most basic element of a business rule is the language used to express it. The very definition of a term is itself a business rule which describes how people think and talk about things. Thus, defining a term is establishing a category of business rule.

- **Facts relating terms to each other**
  The nature or operating structure of an organization can be described in terms of the facts which relate terms to each other. To say that a customer can place an order is a business rule. Facts can be documented as natural language sentences or as relationships, attributes, and generalization structures in a graphical model.

- **Constraints**
  Every enterprise constrains behavior in a non-formal way, e.g., saying that a complete process cannot be executed by just one person and this is closely related to constraints on what data may or may not be updated. To prevent a record from being made is, in many cases, to prevent an action from taking place.

- **Derivations**
  Business rules (including laws of nature) define how knowledge in one form may be transformed into other knowledge, possibly in a different form.

The semantic essence of a business policy expresses a logical definition of some facet of the organization’s way of doing business. An important feature of business policies is that they are usually specified by business people – the people who have responsibility for the business activities to which the rules apply.

In this thesis I regard business policies composed of definitions, facts and constraints.

Today, one best-practice approach is the *Segregation of Duties* or *Separation of Duties* (SoD). It is basically the concept of having more than one person required to complete a task. The use of SoD leads to individual statements specifying conditions and/or limitations. Each statement is a business policy. In more detail, SoD means that there is no single individual having the control over two or more phases of a process, transaction or operation, respectively, so that a deliberate fraud is more difficult to occur because it requires collusion of two or more individuals or parties. There is no industry standard for the separation of duties, but the ISACA (Information Systems Audit and Control Association) has provided a segregation of duties control matrix [ISA05], which is a general guideline suggesting which positions should be separated and which require compensating controls when combined.

A further approach – which is connected to SoD – is the *four-eyes principle*. It should be applied to critical activities or processes which should be performed
by different persons to ensure the correctness of the transactions. As in the SoD concept, the application of four eyes principle results in conditions and limitations – the business policy statements.

The business policies generally impact the execution of a business process. However, only the processes which are mapped in the SAP R/3 system as a sequence of transactions can be regarded. To be more concrete: It is necessary to focus on one specific process. The purchase process which I have selected as a case study for this thesis is one of the typical processes where the previous mentioned approaches SoD as well as the four eyes principle applies. The cost of abuse is substantially greater than the cost of sometimes failing to perform the process. A single business process step is represented in SAP R/3 by a collection of transactions with the associated authorization checks. Therefore, a business policy which has been specified by business people in a business terminology is applied in SAP R/3 by permissions and restrictions to a set of transactions.

A typical, general policy for the purchase process is, for example, the prohibition that the complete process consisting of the steps “create a requisition”, “release a requisition” and “create the order” for this requisition can be performed by one single user.

The purchase process example presented in this thesis also contains a special case for creating direct orders without the existence of a requisition and consequently without the need to perform the release of the requisition. However, this possibility of direct orders is limited to office material with a tight money amount. The business policy is here that it is prohibited to create arbitrary orders greater than the defined money amount without a requisition. All items which do not match the conditions for direct orders must be ordered through the default way which requires the requisition and the subsequent release of the requisition.
Extracting Authorization Information

The properties of a given instance of an SAP R/3 system must be retrieved in order to be able to build a theory. The properties include authorizations which are assigned to users, customizing settings related to the purchase process as well as the transactions with the associated authorization checks.

The first section 3.1 explains how to obtain the number of authorization objects which are checked during the execution of a certain transaction.

The section 3.2 shows the retrieval of the users and the roles and profiles containing the authorizations. Furthermore, the relationship between these elements is retrieved and explained, i.e., which user holds which role/profile. The elements mentioned so far can be obtained automatically.

The last sections 3.4 and 3.3 eventually explain the extraction of the business policies and purchase process respectively and the reason why these tasks must be done manually.

3.1 Authorization Checks

The intuitive approach to retrieve the information of the authorization checks in a transaction is to locate them by looking directly into the source code and by exploring the connection between the transaction (code) and the associated program. The connection definition possibly contains the first authorization object which is checked besides the transaction code check. All further authorization checks are implemented in the source codes of the program associated with the transaction and/or other programs which are called by the first program.

However, most of the SAP R/3 systems are customized to fit the business needs of the company. The identification of authorization checks in the program code is tedious because customized systems typically include customized program code additional to the general customizing. The source codes of SAP R/3 programs is very complex. Without background knowledge or a instruction manual about the structure of the program code it is almost impossible to find out the correct lines doing the authorization checks in the source code.

A solution to this dilemma is the usage of the System Trace tool (transaction code ST01) which is offered by the R/3 system. It monitors among other things the authorization checks occurring during the execution of a transaction.
To identify the authorization checks for a transaction, the System Trace has been executed several times with different data in order to figure out the required objects. The log files of these runs have been compared with the result that some authorization objects were checked in each run and others that were only sometimes checked depending on the data. This leads to the result that the objects present in each run are the objects which are required in order to successfully execute the transaction.

The limitation of this approach is that it is really extensive to check all different possibilities to perform the (purchase) transactions. I could explore only a bunch of possibilities but there are typically a lot of ways (the number of total possibilities is finite) to fill out the forms (for example with special accounting assignments or cost center assignments) and also lots of screen navigation options (buttons, additional dialogs) which potentially lead to additional or different authorization checks. However, the System Trace tool provides a reasonable certainty but only a proof either via the code or by checking all ways to perform a transaction provides a save pair of hands.

Consider the transaction to create a requisition (ME51N) as an example for the identification. After the transaction has been executed (a requisition has been created) the System Trace probably looks like the one shown in Fig. 12. The screenshot shows a section of the different authorization checks for this transaction. It contains both the authorization object names and the required or checked values. Note that the trace lists all checks occurring in the progress of the transaction, it doesn’t matter whether the check is performed at the start of the transaction by the system or whether it is directly implemented in the program code. The System Trace entries in SAP R/3 are linked to the position in the source code of the corresponding program where the authorization check occurs.

The screenshot shows, that some objects occur more than once in the trace. The value in the beginning is different than the one in the end or there is no value in the beginning and some value in the end. Furthermore, some objects occur repeatedly with the same field value. The repeated listings are based on the fact that some objects are checked more than once. Corresponding to the program code it is not always required that an object must have a certain value in an authorization check. Sometimes the value will be entered later or will be changed in the process. Therefore, the appropriate object is checked again by the program. If an object occurs always with the same value in a field during one transaction in the trace, then we say that the field is required in any case with a fixed value. On the other side we have the fields with an empty value in the beginning and some value later in the progress which means that the field is required in any case but with some value depending on the input of the user. The former values typically identify the type of task like create or view and the latter values restrict the access to the specified value, e.g., the field WERKS of the object M_BANF_WRK limits the access to the plant specified in the value; the user is not authorized to create or work with data (depending on the type of task) of another plant.
3.1 Authorization Checks

Fig. 12. Part of the System Trace of the Transaction ME51N

The objects S_ALV_LAYO and S_GUI which are displayed in the screenshot are not directly related to the purchase process. They result from the export of data, e.g., using the input assistance dialog.

The Table 5 shows the identified authorization objects and the required field values which are relevant for the transaction ME51N based on the trace displayed above.

<table>
<thead>
<tr>
<th>Authorization Object</th>
<th>Field = Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_TCODE</td>
<td>TCD = ME51N</td>
</tr>
<tr>
<td>M_BANF_WRK</td>
<td>ACTVT = 01</td>
</tr>
<tr>
<td></td>
<td>WERKS = some value</td>
</tr>
<tr>
<td>M_BANF_BSA</td>
<td>ACTVT = 01</td>
</tr>
<tr>
<td></td>
<td>BSART = some value</td>
</tr>
<tr>
<td>M_BANF_EKG</td>
<td>ACTVT = 01</td>
</tr>
<tr>
<td></td>
<td>EKGRP = some value</td>
</tr>
</tbody>
</table>

Table 5. Identified Authorization Objects in Transaction ME51N
The System Trace log as shown in Fig. 12 can be exported to a simple text file. Using the export function it is possible to automatically generate the appropriate formulae for the theory in first-order logic. The trace lists the authorization checks happened during one execution of the transaction. This list represents the typical authorization checks for the create requisition transaction. Any special user behavior during the execution of this transaction has not been considered.

3.2 Authorization Setup

The authorization setup is the layer on which authorizations are assigned to the users using the concept of roles and direct profiles. Hence, in order to build a fine-grained theory which maps the real SAP R/3 system structure as good as possible it is necessary to retrieve all constituents of the authorization part: on the one hand the users holding certain authorizations (within the authorization profiles) and on the other hand authorization assignment mechanism which makes use of the different concepts like roles, inheritance, profiles, etc.

The following section describes first the retrieval of the users holding the roles. The role itself is either a single or a composite role. A composite role again contains a number of single roles. Authorizations are only specified in the definition of the single roles at any level. The resolution of the relation between the composite and the single roles is done through the composite role definition. The last step constitutes the retrieval of the authorizations which are grouped in direct authorization profiles. The structure of the exported files containing the authorization data will be described for both roles and direct profiles. The authorizations information can then be used in order to create the appropriate formulae for the theory.

The SAP R/3 system offers the User Information System\footnote{Transaction SUIM} which is able to report information about users, their roles and profiles as well as information about authorizations, authorization objects or transactions.

In the first step, the information system is used to obtain all users and the roles which are associated to each user. The result is an output similar to the one displayed in Fig. 13. Rows marked with a C in the column “Indirect” indicate a single role which has been automatically assigned through a composite role (which is assigned to a user). The relation between the composite roles and single roles will be resolved later through the role definition.

The name of each non-marked role is used to obtain the role definition. For this purpose, the SAP R/3 system offers a download function for roles. It produces a file which contains the authorization data and additionally links to the single roles if the exported role is a composite role. Fig. 14 shows an excerpt of a single role export. The exported file contains all information about the role including the authorizations and the menu definition but only the lines holding the role name and the authorization data are important to us. Each line starts with a keyword followed by different data. The first important line starts with \texttt{LOADED\_ARGS} and

\footnote{Transaction SUIM}
refers to the role name. The lines starting with \texttt{AGR.1251} contain the authorization information. The Figure shows, for example, the authorization object \texttt{S.TCODE} with the field \texttt{TCD} and the value \texttt{ME51N} in the first line starting with \texttt{AGR.1251}. The last line starting with \texttt{AGR.1251} has a variable as a value for the authorization object field. Variables represent the organizational units which is just a design issue of SAP R/3 but it doesn't impact the retrieval of the authorizations. Values of the variables are defined by extra lines starting with \texttt{AGR.1252}.

If the role described in the exported file is a composite role then there are additional lines starting with \texttt{AGR.ARGS} followed by the single role name(s). Fig. 15 shows a section of an exported composite role definition. The composite role \texttt{Z: LAMOTTEGENERAL} points to the two single roles \texttt{Z: LAMOTTEVIEWAUTHLOG} and \texttt{Z: LAMOTTEEXPORTDATA}.

The second step is similar to the first one, the information system is again used but here to obtain all users and the profiles which are associated to each user. An example for the resulting output is displayed in Fig. 16. There are different types of profiles: single profiles, composite profiles and generated profiles which are uninteresting at this point because they result from role definitions. The profile name of each non-generated profile is used to obtain the authorizations. The Profile Maintenance Tool\textsuperscript{2} lists all authorizations for a given profile. A section of the export of a resulting list is depicted in Fig. 17. It shows the authorizations consisting of the authorization object (marked with \texttt{<OBJ>}) and the values for its fields (the fields are marked with \texttt{<FLD>}).

\textsuperscript{2} Transaction SU02
<table>
<thead>
<tr>
<th>User</th>
<th>Full Name</th>
<th>Role</th>
<th>Type</th>
<th>Assignment</th>
<th>Indirect</th>
<th>Start date</th>
<th>End date</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:LAMOTTE_GENERAL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>19.07.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:LAMOTTE_VIEW_AUTHLOG</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>03.08.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:LAMOTTE_PURCHREQ_INFO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>03.08.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:LAMOTTE_EXPORT_DATA</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>03.08.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>John Smith</td>
<td>Z:LAMOTTE_PURCHORD_INFO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>03.08.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>John Smith</td>
<td>Z:LAMOTTE_GENERAL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>19.07.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>John Smith</td>
<td>Z:LAMOTTE_VIEW_AUTHLOG</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>03.08.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>John Smith</td>
<td>Z:LAMOTTE_EXPORT_DATA</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>03.08.2007</td>
<td>31.12.9999</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 13.** Excerpt from the Export of Roles Assigned to Users in text-format
Fig. 14. Export of a Single Role through the Download Function

Fig. 15. Export of a Composite Role via Download Function
<table>
<thead>
<tr>
<th>User</th>
<th>Full Name</th>
<th>Profile</th>
<th>Text</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>T:R0480014</td>
<td>Profile to role LAMOTTE_VIEW_PURCHREQ_INFO</td>
<td>G Generated</td>
</tr>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:EK1_INFO</td>
<td>Purchasing INFO</td>
<td>S Sgle profile</td>
</tr>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:MAT_INFO</td>
<td>Material master records INFO</td>
<td>S Sgle profile</td>
</tr>
<tr>
<td>LAMOTTE</td>
<td>Manuel Lamotte</td>
<td>Z:MAT_INFO</td>
<td>Material master records INFO</td>
<td>S Sgle profile</td>
</tr>
<tr>
<td>SMITH</td>
<td>John Smith</td>
<td>Z:EK1_INFO</td>
<td>Purchasing INFO</td>
<td>S Sgle profile</td>
</tr>
<tr>
<td>SMITH</td>
<td>John Smith</td>
<td>Z:MAT_INFO</td>
<td>Material master records INFO</td>
<td>S Sgle profile</td>
</tr>
<tr>
<td>LINCOLN</td>
<td>Abraham Lincoln</td>
<td>SAP_ALL</td>
<td>All authorizations</td>
<td>C Comp. profile</td>
</tr>
</tbody>
</table>

Fig. 16. Excerpt of the Export of Profiles Assigned to Users in text-format
3.3 Purchase Process

The purchase process – which is considered here – consists of the steps “Create a Requisition”, “Release of the Requisition” and the “Creation of an Order”. As already described in section 2.3.1 the process steps are mapped by a number of transactions in SAP R/3. Therefore, the extraction of the authorization information first results in the identification of the transactions which are required for the steps and subsequently in the identification of the individual authorization checks which occur in each transaction. The identification of authorization checks is done as described in section 3.1 using the System Trace tool. The following sections show the identified authorization checks and the required values for each trans-
action. However, the execution of the release requisition transaction additionally requires the purchase customizing. This customizing defines the settings for the release strategies and is essential to release requisitions. Therefore, these customizing settings also have to be extracted. They are presented together with the release transaction in section 3.3.2.

3.3.1 Create a Requisition

The extraction of the authorization information which occurs in the create requisition transaction has already been explained as an example in section 3.1.

3.3.2 Release a Requisition

The following Table 6 lists the result of the identification of authorization objects for the release requisition transaction. The Table shows that there are two transaction codes in the transaction field. This is due to the fact that a user first must have (at least he/she should have) the permission to view the requisition before he/she can release it. Therefore, additionally to the permission to release requisitions (through transaction ME54N) the user need access to view requisitions (transaction ME53N). This additional permission requirement also leads to the different activity values which are subject to the authorization checks.

<table>
<thead>
<tr>
<th>Authorization Object</th>
<th>Field = Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_TCODE</td>
<td>TCD = ME54N, ME53N</td>
</tr>
<tr>
<td>M_BANF_WRK</td>
<td>ACTVT = 03, 02, WERKS = some value</td>
</tr>
<tr>
<td>M_BANF_BSA</td>
<td>ACTVT = 03, 02, 08, BSART = some value</td>
</tr>
<tr>
<td>M_BANF_EKG</td>
<td>ACTVT = 03, 02, EKGRP = some value</td>
</tr>
<tr>
<td>M_EINK_FRG</td>
<td>FRGGR = some value</td>
</tr>
<tr>
<td>M_EINK_FRG</td>
<td>FRGCO = some value</td>
</tr>
</tbody>
</table>

Table 6. Identified Authorization Objects in Transaction ME54N

In order to release requisitions, the purchase customizing (that means the release strategy settings) have to be extracted, too. Unfortunately, these settings can not be extracted automatically. Thus, the extraction is done by a simple manual transcription of the release strategy settings including the characteristics class with their characteristics or properties, the release codes, the release group and the release indicators from the R/3 system. In the following, the manual transcription of the mentioned elements is shown by an example release strategy. This strategy consists of two release steps: the “cost center release” and the “director release”.

The first elements that need to be extracted are the characteristics class(es) with their properties. The example release strategy in this thesis uses just the
class FRG_EBAN. After inserting the class name (either directly or indirectly through a selection in the input assistance dialog which lists all available classes) in the transaction CL02 the R/3 system presents the details of the class and the assigned characteristics. Fig. 18 is a screenshot of the transaction showing the class FRG_EBAN and its characteristics. The characteristics are connected to fields in the requisition, FRG_EBAN_EKGRP is connected to the purchasing group, FRG_EBAN_WERKS to the plant and FRG_EBAN_GSWRT to the overall money amount of the requisition items.

![Display Class:](image)

The extracted data of the characteristics class which is shown in Fig. 18 is listed in Table 7. The value(s) for the property fields are available in the classification dialog inside the release strategy definition (see Fig. 19).

All remaining elements will be extracted (manually) from the release strategy definition, too. Each release strategy is assigned to a release group and has a number of release codes. The cost center release (identifier KF), for example, is assigned to the group 01 and has the release code W1. On the other hand, the
3.3 Purchase Process

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRG_EBAN_EKGRP</td>
<td>Matches the Purchasing Group</td>
<td>I26</td>
</tr>
<tr>
<td>FRG_EBAN_WERKS</td>
<td>Matches the Plant</td>
<td>INFO</td>
</tr>
<tr>
<td>FRG_EBAN_GSWRT</td>
<td>Matches the Total Money Amount</td>
<td>&lt;=10000</td>
</tr>
</tbody>
</table>

Table 7. Characteristics Class for Purchase Requisitions

director release (identifier VF) is also assigned to the group 01 and requires two single release: one release with the code W1 and one with W2. This is shown in Fig. 19.

![Change View "Release Strategies: Purchase Requisition": Details](image)

Fig. 19. Release Strategy Setup

The release prerequisites accessible via the button “Release prerequisites” define the order of the single release steps and the prerequisites for each step. The actions which are permitted in this strategy for a requisition after the release with a certain code are accessible via the button “Release statuses”. However, the release prerequisites and the release statuses settings are not considered in the theory due to abstractions (see chapter 5).
3.3.3 Create an Order

The identified authorization objects which are checked in the create order transaction are shown in Table 8. This is the default case where the requisition has to be released before the creation of the order.

<table>
<thead>
<tr>
<th>Authorization Object</th>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_TCODE</td>
<td>TCD</td>
<td>ME21N</td>
</tr>
<tr>
<td>M_BEST_EKO</td>
<td>ACTVT</td>
<td>01, 09</td>
</tr>
<tr>
<td></td>
<td>EKORG</td>
<td>some value</td>
</tr>
<tr>
<td>M_BEST_BSA</td>
<td>ACTVT</td>
<td>01, 09</td>
</tr>
<tr>
<td></td>
<td>BSART</td>
<td>some value</td>
</tr>
<tr>
<td>M_BANF_BSA</td>
<td>ACTVT</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>BSART</td>
<td>some value</td>
</tr>
<tr>
<td>M_BEST_WRK</td>
<td>ACTVT</td>
<td>01, 08, 09</td>
</tr>
<tr>
<td></td>
<td>WERKS</td>
<td>some value</td>
</tr>
<tr>
<td>M_BEST_EKG</td>
<td>ACTVT</td>
<td>01, 09</td>
</tr>
<tr>
<td></td>
<td>EKGRP</td>
<td>some value</td>
</tr>
</tbody>
</table>

Table 8. Identified Authorization Objects in Transaction ME21N

The special case introduced in section 2.3.1 describes the possibility to create direct orders for office equipment with a tight limit on the monetary amount. The following Table 9 shows the identified checks for the direct creation of office equipment orders. There is a separate transaction code (ME21NOFF) which is used to perform the transaction to create office equipment orders. The start with this special code also implies a check of the material (object M_EINK_MKL) and the monetary amount (object M_EINK_GWT).

3.4 Business Policies

Business Policies – specified by business people – grant or restrict the access to a set of transactions in SAP R/3. A general policy for the purchase process which has been presented in section 2.4 prohibits that the complete process consisting of the steps “create a requisition”, “release a requisition” and “create the order” (for the same requisition) can be performed by one single user. In more detail, the meaning of this policy is that a single user is not permitted to perform these steps for one concrete plant, material group, purchasing group and organization. However, for example, it is admissible that a single user has the authorizations to perform all these steps but for different plants, i.e. he/she can create requisitions for plant X, release requisitions for plant Y and create orders for plant Z. In terms of transactions, this means that the user should not have the authorizations to perform the appropriate transactions to create requisitions, release requisitions and to create orders for one plant, material group, purchasing group and organization. This example has been formalized in this thesis and is presented in section 5.5.
Unfortunately, the business policies are not directly represented in SAP R/3. Therefore, the policies cannot be extracted automatically. Since they restrict business processes which are mapped in SAP R/3 by a set of transactions, they can be expressed in terms of transactions, too. In general, business policies will be “translated” from the business people language to a language that composes SAP R/3 elements like transactions and authorizations. Then the translated policies can be formalized.

<table>
<thead>
<tr>
<th>Authorization Object</th>
<th>Field=Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_TCODE</td>
<td>TCD=ME21N, ME21NOFF</td>
</tr>
<tr>
<td>M_EINK_MKL</td>
<td>MATKL=B0002</td>
</tr>
<tr>
<td>M_EINK_GWT</td>
<td>GSWRT=some value</td>
</tr>
<tr>
<td>M_BEST_EKO</td>
<td>ACTVT=01, 09, EKORG=some value</td>
</tr>
<tr>
<td>M_BEST_BSA</td>
<td>ACTVT=01, 09, BSART=some value</td>
</tr>
<tr>
<td>M_BANF_BSA</td>
<td>ACTVT=01, BSART=some value</td>
</tr>
<tr>
<td>M_BEST_WRK</td>
<td>ACTVT=01, 08, 09, WERKS=some value</td>
</tr>
<tr>
<td>M_BEST_EKG</td>
<td>ACTVT=01, 09, EKGRP=some value</td>
</tr>
</tbody>
</table>

Table 9. Identified Authorization Objects in Transaction ME21NOFF
The formalization of the SAP R/3 system is accomplished in [chapter 5] using first-order logic without equality. Thus, the syntax of the used language has to be defined. The following syntax definition is taken from [NW01].

A first-order language is constructed over a signature \( \Sigma = (F, R) \), where \( F \) and \( R \) are non-empty, disjoint, in general infinite sets of function and predicate symbols, respectively. Every function or predicate symbol has some fixed arity. In addition to these sets that are specific for a first-order language, we assume a further, infinite set \( \mathcal{X} \) of variable symbols disjoint from the symbols in \( \Sigma \). Then the set of all terms \( T(F, \mathcal{X}) \) is recursively defined by: (i) every constant symbol \( c \in F \) with arity zero is a term, (ii) every variable \( x \in \mathcal{X} \) is a term and (iii) whenever \( t_1, \ldots, t_n \) are terms and \( f \in F \) is a function symbol with arity \( n \), then \( f(t_1, \ldots, t_n) \) is a term. A term not containing a variable is a ground term. If \( t_1, \ldots, t_n \) are terms and \( R \in R \) is a predicate symbol with arity \( n \), then \( R(t_1, \ldots, t_n) \) is an atom. An atom or the negation of an atom is called literal. Disjunctions of literals are clauses where all variables are implicitly universally quantified. Formulae are recursively constructed over atoms, the logical constants \( \top \) (truth), \( \bot \) (falsity) and the operators \( \lor \) (implication), \( \equiv \) (equivalence), \( \land \) (conjunction), \( \lor \) (disjunction), \( \neg \) (negation) and the quantifiers \( \forall \) (universal), \( \exists \) (existential) as usual: (i) every atom is a formula, (ii) \( \top \) and \( \bot \) are formulae. (iii) if \( \phi_1 \) and \( \phi_2 \) are formulae, so are \( \phi_1 \equiv \phi_2, \phi_1 \lor \phi_2, \phi_1 \land \phi_2, \phi_1 \lor \phi_2 \) and \( \neg \phi_1 \) and (iv) if \( \phi \) is a formula and \( x \in \mathcal{X} \) a variable, then \( \forall x \phi \) and \( \exists x \phi \) are formulae.

For convenience, we often write \( \forall x_1, \ldots, x_n . \phi \) instead of \( \forall x_1 \ldots \forall x_n . \phi \) and analogously for the existential quantifier. The semantics of the defined language is described in [NW01].

The theory is here the formalization of the functionality and the authorization setup of an SAP R/3 system together with the (formalized) business policies.

The (Herbrand) interpretation \( I \) is a set of ground atoms \( G \subseteq S \) where \( S \) is the set of all ground atoms in the first-order language. A ground atom \( A \) is said to be true in an interpretation \( I \) if \( A \in I \), and false otherwise. The interpretation \( I \) is called a model of an expression \( E \) if \( E \) is true in \( I \); and a model of a set of expressions \( N \), if it is a model of all expressions in \( N \). The model \( M \subseteq S \) is a
minimal model for a given theory if there exists no $M' \subset M$ which is also a model for the theory.

4.2 Notation

The general notation used in the following chapter Theory Development is defined as follows:

- Predicates: The first letter of the predicate symbol is written in uppercase letters and the predicate itself is written in italic font: $Access(a, b)$, $SingleRole()$.

- Variables: Variables are written *italics*, in lower case font and labeled with a prefixed “$x$”, e.g. $xu$, $xpn$.

- Functions: Function names are always written *italics* and they start with a lower case letter different from $x$, e.g. $authObj(x, y, z)$, $userProfileEntry(x, y)$.

- Constants: Functions with arity 0 (= constants) are written in typewriter font and uppercase letters, for example $LESS_EQUAL_1000_EUR$, $STAR$. Constants from the SAP R/3 system are adopted 1:1, e.g. $S_TCODE$, $01$, $M_BANF_WRK$, $LAMOTTE$.

- Function and predicate descriptions: They introduce the function/predicate name and describe the required arguments for the function/predicate. The arguments in this description are written in square brackets, for example, $<user>$, $<transaction code>$.
5

Theory Development

We are now going to develop a theory in first-order logic representing the authorization setup layer, the authorization check layer and the business process layer. The theory is based on the purchase process as a case study. Therefore, it includes the formalized purchase business policies. The prerequisite for the construction of the theory is a snapshot of the SAP R/3 system, i.e. the extracted authorization information (cf. chapter 3). The corresponding theory represents the state of the system at a given time. Any change of the authorizations in the system or in the business policies always require an adjustment of the formulae.

The syntax and the notation used in the following sections have already been introduced in chapter 4.

5.1 Abstractions

Keeping the aim of this thesis in mind, – the automatic proof of violations of authorizations – the theory shall have a fine grained structure on the one hand in order to be consistent as much as possible with the actual SAP R/3 structure. On the other hand, the model has to be easily understandable and it has to show that the automatic proof of violations of authorizations is possible.

The theory presented in the following sections is based on some abstractions. The first abstraction refers to time. It would be possible to model the time when an authorization check happens during the execution of a transaction in first-order logic but this would lead to an immense increase of the runtime of theorem provers because of the obvious state explosion, at least for large R/3 systems and the resulting theories. For example, a time line can be modeled using a list holding a concatenated sequence of authorization checks. This approach makes it possible to speak about the order of the different authorization checks and also about the order of a number of single release steps. However, the theory should only examine the validity of authorizations of a user in a transaction or the validity of the overall release of a requisition, respectively, which is an admissible abstraction. It doesn’t matter when a check or release step occurs, it is only important whether it occurs or not.

Additionally, the assumption for this theory are unique data. Dynamics in the data or the authorizations will not considered in this theory. Consequently, for
example, a change of user authorizations while the user is performing a transaction is excluded.

There are two further abstractions which are related to the purchase process. In SAP R/3 a requisition or order can include many individual items. In the first abstraction the formal model uses only one item per requisition/order. In other words, a new requisition/order object is created for each item. In the second abstraction the theory omits the release indicators in the release strategy. The assumption for the theory presented here is that each change of a requisition when the release procedure has already started results in a restart of the release procedure. This assumption is not modeled explicitly.

The last abstraction refers to the pattern matching of strings. In the SAP R/3 system it is used during the authorization checks and the release strategy appliance checks. So far, the matching mechanism is limited to the symbol * (no composed values containing the *) which matches every required value and the direct comparison, i.e., both the required and the present value are equal. However, it would be possible to model the pattern matching of strings in first-order logic. It has been omitted in order to keep the model more simple. Pattern matching of strings can be achieved using a set \( \Sigma = \{A-Z, 0-9, ?, *\} \) which defines the terminal symbols, a binary matching function \( \text{match}(x,y) \) saying that \( x \) matches \( y \) and a function \( \text{cons}(x,y) \) which creates a list consisting of the first element (the character \( x \)) and the rest of the list (the string \( y \)). Strings can then be composed as follows:

\[
\begin{align*}
\text{String:} & \quad \text{cons:} \quad \Sigma \times \text{String} \rightarrow \text{String} \\
\varepsilon: & \quad \text{String}
\end{align*}
\]

According to the previous definition, the string “abc”, for example, will be written as:

\[
\text{cons}(a, \text{cons}(b, \text{cons}(c, \varepsilon)))
\]

A matching relation for strings could be defined using the following rewrite rules\(^1\):

\[
\begin{align*}
\text{match}(\varepsilon, \varepsilon) & \rightarrow \text{true} \\
\text{match}(\text{cons}(x, y), \text{cons}(x, z)) & \rightarrow \text{match}(y, z) \\
\text{match}(\text{cons}(?, y), \text{cons}(x, z)) & \rightarrow \text{match}(y, z) \\
\text{match}(\text{cons}(*, y), z) & \rightarrow \text{match}(y, z) \\
\text{match}(\text{cons}(*, y), \text{cons}(x, z)) & \rightarrow \text{match}(\text{cons}(*, y), z)
\end{align*}
\]

5.2 Authorization Setup

This section describes the formalization of the authorization setup. It includes the authorization roles, the generation of profiles and the authorization check presented in chapter 2 as well as a few Customizing settings (release strategy settings) necessary to do purchasing activities.

\(^1\) A rewrite rule is here a rule of the form \( A \rightarrow X \) expressing the fact that \( A \) can be replaced by \( X \) which generalizes the overall term
5.2 Authorization Setup

5.2.1 Roles

A role contains either different authorizations (a single role) or in turn further roles with authorizations (a composite role). But this or is an exclusive or, it isn’t possible to mix single roles and authorizations in one composite role level.

SAP R/3 offers a great variety of possibilities in the field of roles. However, due to the complexity only the structure capabilities of roles has been mapped – other features like inheritance have not been implemented in the theory.

A single role is modeled by the unary atom `SingleRole`. The function `authObj()` with arity 3 therein maps the authorization value to the authorization field of the authorization object. The authorization object together with the value in turn represents the authorization which is mapped to the single role by the binary function `singleRoleEntry()`.

\[
\text{SingleRole}(\text{singleRoleEntry}(<\text{single role name}>, \text{authObj}(<\text{auth object name}>, <\text{auth field}>, <\text{value}>)))
\]

- Consider the following two formulae as example instances of the previous definition. They are part of the single role `ZLAMOTTE_BANF_INFO` granting the permission to create a requisitions. The first authorization object which is checked in the transaction to create a requisition is the object `S_TCODE` with its field `TCD` (the transaction code check). The constant `ME51N` is assigned as a value to the field `TCD` in the first formula. Eventually, the function `singleRoleEntry()` maps the authorization to the single role.

The second formula assigns the constant `INFO` as a value to the field `WERKS` of the authorization object `M_BANF_WRK` which in turn maps the authorization to the single role. The result of this formula is a restriction of any action to the plant `INFO` concerning purchase requisitions.

\[
\text{SingleRole}(\text{singleRoleEntry}(\text{ZLAMOTTE_BANF_INFO}, \\
\text{authObj}(\text{S_TCODE}, \text{TCD}, \text{ME51N})))
\]

\[
\text{SingleRole}(\text{singleRoleEntry}(\text{ZLAMOTTE_BANF_INFO}, \\
\text{authObj}(\text{M_BANF_WRK}, \text{WERKS}, \text{INFO})))
\]

A composite role is modeled using the unary atom `CompositeRole`. The function `compositeRoleEntry()` therein associates the specified single role with the composite role.

\[
\text{CompositeRole}(\text{compositeRoleEntry}(<\text{composite role name}>, <\text{single role name}>)
\]

- In what follows, there is an example for a composite role. The single roles `ZLAMOTTE_ORDER_INFO_BASE` and `ZLAMOTTE_ORDER_INFO_INFO` are contained in the role `ZLAMOTTE_ORDER_INFO`. On the one hand, a transaction typically requires a number of basic authorizations which are equal for every user authorized to perform a certain transaction. These authorizations are collected in the single role `ZLAMOTTE_ORDER_INFO_BASE`. On the other hand, there are
5.2 Authorization Setup

different groups of people allowed to execute this transaction, but limited to
different domains, for example to a concrete plant or company code. The lim-
itations are specified in further (different) single roles (here for instance the
role ZLAMOTTE_ORDER_INFO which limits the plant to INFO). A success-
ful execution of the transaction is only possible in combination with the basic
authorizations which is the notion of the composite role.

\[
\text{CompositeRole(compositeRoleEntry(ZLAMOTTE\_ORDER\_INFO,}
\text{ZLAMOTTE\_ORDER\_INFO\_BASE))}
\]

\[
\text{CompositeRole(compositeRoleEntry(ZLAMOTTE\_ORDER\_INFO,}
\text{ZLAMOTTE\_ORDER\_INFO\_INFO))}
\]

5.2.2 Profile Generation and Authorization Profiles

During an authorization check the required value of an authorization is compared
with the value in the users’ authorization profile. There are auto-generated and
direct authorization profiles. Both types of profiles represent authorizations that
a user holds.

The general authorization profile of a user which provides his/her effective au-
thorizations is modeled using the unary atom \textit{UserProfile}. The function \textit{userPro-
fileEntry()} maps the different authorizations to the user:

\[
\text{UserProfile(userProfileEntry(<user>,}
\text{authObj(<auth object name>, <auth field>, <value>))))}
\]

- A concrete instance of this atom is similar to a single role instance; it also
  contains the authorization (the authorization object with its value). The differ-
  ence to the single role instance is that it is directly mapped to the user by the
  function \textit{userProfileEntry()}. A very important point is that instances of this
  atom will only be created through a special transition which is explained at
  the end of this section \textit{ssectFormModelProfGenAndAuthProfiles}. Of course,
  a complete user authorization profile typically consists of many single instances
  of this atom. Here is an example for one instance:

\[
\text{UserProfile(userProfileEntry(LAMOTTE,}
\text{authObj(M\_BANF\_WRK, WERKS, INFO))}}
\]

The value INFO in this example enables all actions of the user LAMOTTE concern-
ing purchase requisitions to the plant INFO.

The previous mentioned atom \textit{UserProfile()} models the effective authorizations
of a user; any authorization check looks for the required authorization in these
authorization profile instances.

The possibility of SAP R/3 to assign authorizations via a direct authorization
profile is eventually modeled by the following unary atom \textit{Profile()}:
5.3 Authorization Checks

Any authorization must be assigned to a user in order to make it effective. However, authorizations cannot be assigned directly because they are contained in structures like roles or profiles. Therefore, the assignment of authorizations is accomplished by the assignment of the profile, single or composite role to a user. This is modeled by the following predicate \( \text{Holds} \) denoting the fact that user holds the authorizations of a profile, a single or composite role.

\[
\text{Holds(}\langle \text{user}\rangle, \langle \text{profile name/single role name/composite role name}\rangle)\)
\]

As already mentioned, effective authorization instances (represented by instances of the atom \( \text{UserProfile()} \)) are not created manually. Instead, the assignment of any structure (direct profile/role) containing authorizations to a user implies the creation of the appropriate authorization profile instance. This reflects also the auto-generation of the authorization profile according to a role definition. The generation is achieved by a transition which creates the corresponding user authorization profile instance from a given profile, single or composite role. It determines the type of the item assigned to the user via the predicate \( \text{Holds} \), extracts the authorization part and creates the authorization profile entry (representing the effective authorization) for the user. The following formulae shows the creation of authorization profile instances:

\[
\forall xu, xpn, xsrn, xcrn, xaon, xao, xav . \\
(\text{Profile(profileEntry(xpn, authObj(xaon, xao, xav)))} \land \\
\text{Holds(xu, xpn)}) \lor \\
(\text{SingleRole(singleRoleEntry(xsrn, authObj(xaon, xao, xav)))} \land \\
\text{Holds(xu, xsrn)}) \lor \\
(\text{CompositeRole(compositeRoleEntry(xcrn, xsrn))} \land \\
\text{SingleRole(singleRoleEntry(xsrn, authObj(xaon, xao, xav)))} \land \\
\text{Holds(xu, xcrn)}) \\
\supset \\
\text{UserProfile(userProfileEntry(xu, authObj(xaon, xao, xav)))}
\]

5.3 Authorization Checks

The authorization check result – access or decline – is represented in our first-order model by the binary atom \( \text{Access()} \). If the predicate is valid, the access to the checked authorization object is granted otherwise it is not.

\[
\text{Access(}\langle \text{user}\rangle, \text{authObj(}\langle \text{auth object name}\rangle, \langle \text{auth field}\rangle, \langle \text{value}\rangle)\rangle)
\]

The function \( \text{authObj()} \) with arity 3 maps the authorization value to the authorization field of the authorization object. The resulting authorization is then associated with the user.
As explained in section 2.1.1 a transaction consists of several authorization checks. Therefore, the predicate \textit{Access} is reused as an abstraction to model the access to an overall transaction, too. As a consequence, it is valid if all individual authorization checks of the entire transaction have been successful.

\textit{Access}(\textless\textit{user}\textgreater, \textless\textit{transaction code}\textgreater)

- Examples for the abstraction are provided in section 5.4 for the different transactions of the purchase process.

The authorization check itself compares the value of the authorization present in the (generated) user authorization profile with the required authorization. The result of the check is indicated by validity of the predicate \textit{Access} which has been introduced before. The theory presented in this thesis provides two basic ways to pass such an authorization check. The first way is that the user has the exact required authorization in its authorization profile. The authorization object name, the field and the value must match. This is achieved by the following formula:

$$\forall xu, xanon, xaof, xav. \\
\textit{UserProfile}\left(\textit{UserProfileEntry}(xu, \textit{authObj}(xanon, xaof, xav))\right) \\
\supseteq \\
\textit{Access}(xu, \textit{authObj}(xanon, xaof, xav))$$

The second way is that the authorization value of an authorization in the users’ authorization profile is the wild-card symbol. In this case, if the authorization objects match then the access is always granted, no matter what the required value is. The authorization in the user authorization profile must contain the constant \texttt{STAR} as a value which is depicted by the following transition.

$$\forall xu, xanon, xaof, xav. \\
\textit{UserProfile}\left(\textit{UserProfileEntry}(xu, \textit{authObj}(xanon, xaof, \texttt{STAR}))\right) \\
\supseteq \\
\textit{Access}(xu, \textit{authObj}(xanon, xaof, xav))$$

5.4 Purchase Process

This following sections describe the formalization of the purchase process as a business process example with its authorization checks.

5.4.1 Create a Requisition

The following transition shows the abstraction of the single authorization checks using the predicate symbol \textit{Access} (which has been introduced in section 5.3).

The first check represents the check of the transaction code to create a requisition which is \texttt{ME51N}. The authorizations \texttt{M\_BANF\_WRK}, \texttt{M\_BANF\_BSA} and \texttt{M\_BANF\_EKG}
have a constant in the first field (ACTVT) and a variable value in their second field. The atom $Access(xu, \text{ME51N})$ only becomes valid if the user $xu$ has the exact constants in his authorization profile and some value for every variable value. This means that the user is allowed to execute the transaction in at least one instance, for example for one plant, one document type and one purchase group. The concrete value of the variables will be evaluated and checked at runtime when a requisition is going to be created and the exact values are known.

\[ \forall xu, xwrk, xbsa, xekg . \]
\[ Access(xu, auth\text{Obj}(\text{S\_TCODE, TCD, ME51N})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_WRK, ACTVT, 01})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_WRK, WERKS, xwrk})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_BSA, ACTVT, 01})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_BSA, BSART, xbsa})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_EKG, ACTVT, 01})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_EKG, EKGRP, xekg})) \]
\[ \supset \]
\[ Access(xu, \text{ME51N}) \]

The transitions to change or view requisitions are similar. They only differ in the activity code which is either 01 (create), 02 (change) or 03 (view). The following formula represents the transition which models the general access to the transaction to change requisitions ($\text{ME52N}$).

\[ \forall xu, xwrk, xbsa, xekg . \]
\[ Access(xu, auth\text{Obj}(\text{S\_TCODE, TCD, ME52N})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_WRK, ACTVT, 02})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_WRK, WERKS, xwrk})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_BSA, ACTVT, 02})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_BSA, BSART, xbsa})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_EKG, ACTVT, 02})) \land \]
\[ Access(xu, auth\text{Obj}(\text{M\_BANF\_EKG, EKGRP, xekg})) \]
\[ \supset \]
\[ Access(xu, \text{ME52N}) \]

The abstraction groups a number of authorizations together and labels this group with the transaction code. It is based on the concept of transactions in SAP R/3. The exact value of the variables is not known at this point in time. Therefore, the validity of an atom like $Access(xu, \text{ME52N})$ denotes that the user holds the authorizations to change requisitions (Activity 02) for a certain but still unknown plant (variable $xwrk$), type of requisition (variable $xbsa$) and purchasing group (variable $xekg$). Therefore, the latter authorization checks must occur again when there are concrete values for the variables. Consequently, the first occurrences are redundant but they are mentioned in order to give a better understanding for the overall abstraction from the single checks to the transaction.

The abstraction of authorizations to a higher level (namely the transaction) provides a good basis to model now the business process steps which typically consist
of several transactions. The purchase request for an arbitrary item is modeled by the atom *Requisition* with arity 8. The structure is described as follows:

\[
\text{Requisition}(\langle \text{document type} \rangle, \langle \text{position} \rangle, \langle \text{material} \rangle, \langle \text{plant} \rangle,
\langle \text{purchasing group} \rangle, \langle \text{purchasing organization} \rangle,
\langle \text{material group} \rangle, \langle \text{price} \rangle)
\]

Similar to the request there is the state *RequisitionCreated* which indicates that the requisition object has been created by a user in SAP R/3. The difference to the *Request* is that the user who has created the requisition is now also connected to it. This leads to the following atom with arity 9:

\[
\text{RequisitionCreated}(\langle \text{user} \rangle, \langle \text{document type} \rangle, \langle \text{position} \rangle, \langle \text{material} \rangle, \langle \text{plant} \rangle,
\langle \text{purchasing group} \rangle, \langle \text{purchasing organization} \rangle,
\langle \text{material group} \rangle, \langle \text{price} \rangle)
\]

The definition of the two atoms *Requisition* and *RequisitionCreated* lead to the following transition which represents the business step to create a requisition.

\[
\forall xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgsurt .
\text{Requisition}(xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xp) \land
\text{Access}(xu, \text{ME51N}) \land
\text{Access}(xu, \text{authObj}(\text{M\_BANF\_WRK, WERKS, xwrk})) \land
\text{Access}(xu, \text{authObj}(\text{M\_BANF\_BSA, BSART, xbsa})) \land
\text{Access}(xu, \text{authObj}(\text{M\_BANF\_EKG, EKGRP, xekg})) \land
\text{Access}(xu, \text{ME53N}) \land
\text{Access}(xu, \text{ME52N})
\implies
\text{RequisitionCreated}(xu, bsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgsurt)
\]

The process starts with the existence of a purchase request modeled by the atom *Requisition*. In order to create a purchase requisition object in SAP R/3, a user needs access to the corresponding transaction ME51N. The atom Access(xu, ME51N) stands for the group of single authorization checks in this transaction. As mentioned, the variables xwrk, xbsa, xekg must be checked again because at this point the value of the variables is known (namely the value from the requisition that is going to be created). Typically, a user authorized to create a requisition is also authorized to view it or to change it in case of a mistake. This is modeled by the lines checking the access to the transactions ME53N (View) and ME52N (Change). A concrete request for a requisition refers to a specific plant and also requires a specific purchase document type and purchasing group, when it is going to be entered in SAP R/3. The authorization to access these properties is modeled by the check of the values for the variables xwrk, xbsa and xekg. In order to succeed these checks the user must be authorized to create the requisition object for the plant, document type and purchasing group specified by the *Requisition*. All authorization checks refer to the same user denoted by the variable xu. If all conditions are satisfied, then the state *RequisitionCreated* is implied. The result represents the state in SAP R/3 that the requisition object has been created.
5.4.2 Release a Requisition

The release of requisitions requires different settings in the Customizing of purchases. At first, these settings have to be formalized in order to apply a release strategy to a requisition.

The first step is the definition of the release strategy. It is modeled by the atom $ReleaseStrategy$ with arity 3.

$$ReleaseStrategy(<release\ strategy\ name>,$ $<release\ group>,$ $class(<characteristics\ class\ name>,$ $property(<property\ name>,$ $<value>)))$$

The nested structure is required in order to obtain a theory which is very close to the structure of the real R/3 system wherein the release strategy combines a characteristics class (containing single characteristics, each given by a property and its value(s)), the release group and code and the release strategy name. The binary function $property()$ maps the value to its property which is in turn mapped to the class name by the binary function $class()$. At the end, the complete characteristic is assigned to the release strategy.

- Consider the release strategy $VF$ as an example. This strategy stands for the director release strategy. The strategy $VF$ applies when the following conditions are satisfied:
  i) the plant entered in the requisition is equal to the value $INFO$,
  ii) the specified purchasing group is equal to $I26$ and
  iii) the total money amount of the item which is subject to the release strategy application check is greater than 10000 EUR.

The different properties which match to the fields in a requisition can either be selected from the database table $CEBAN$ where the properties are already associated to the fields or they can be specified manually in the characteristic class definition. In the latter case, the connection to the appropriate field of the requisition must be established manually, too. The properties $FRG.CEBAN.WERKS$, $FRG.CEBAN.EKGRP$ and $FRG.CEBAN.GSWRT$ are defined in the table $CEBAN$ and match the plant, the purchasing group and the total money amount of the item, respectively. The single conditions of the previous list are united in the class $FRG.EBAN$. This fictive example results in the following instances of the atom $ReleaseStrategy$:

$$ReleaseStrategy(VF,\ 01,\ class(FRG.EBAN,$ $property(FRG.CEBAN.WERKS,\ INFO))))$$

$$ReleaseStrategy(VF,\ 01,\ class(FRG.EBAN,$ $property(FRG.CEBAN.EKGRP,\ I26))))$$

$$ReleaseStrategy(VF,\ 01,\ class(FRG.EBAN,$ $property(FRG.CEBAN.GSWRT,\ GREATER_{10000\ EUR}))))$$
A concrete release strategy consists of one or more release steps. Each of the steps requires a release based on a release group and the release code. This requirement is modeled with the atom \( \text{ReleaseRequirement} \) with arity 3.

\[
\text{ReleaseRequirement}(\langle \text{release strategy name} \rangle, \langle \text{release group} \rangle, \langle \text{release code} \rangle)
\]

- The following two formulae again refer to the release strategy \( \text{VF} \) – the director release. This release strategy consists of two steps, the cost center release in the beginning (denoted by the code \( \text{W1} \)) and the director release at the end (denoted by the code \( \text{W2} \)). The release strategy and the release codes are grouped in the release group denoted by the number \( \text{01} \). In order to perform the release first step, the releasing person needs the right release group and code in his authorization profile.

\[
\text{ReleaseRequirement}(\text{VF}, \text{01}, \text{W1})
\]

\[
\text{ReleaseRequirement}(\text{VF}, \text{01}, \text{W2})
\]

The abstraction of the single authorization checks which are bundled in the transaction to release a requisition are again modeled using the atom \( \text{Access} \). The following formula shows this transition. A user authorized to execute this transaction needs on the one hand authorizations to view the requisition (access to \( \text{ME53N} \) checked in the beginning) and on the other hand authorizations to save the release (access with activity \( \text{02} \) checked at the end), assumed that he/she has the right authorizations for the plant (variable \( \text{xwrk} \)), the document type (variable \( \text{xbsa} \)), purchasing group (variable \( \text{xekg} \)), release group (variable \( \text{xfrggr} \)) and release code (variable \( \text{xfrgco} \)) for the actual requisition. Note that the permission to change the requisition (activity \( \text{02} \)) here is not equal to the general transaction to change a requisition (transaction code \( \text{ME52N} \)) because there is no check of this transaction code.

\[
\forall \ xu, \ xwrk, \ xbsa, \ xekg, \ xfrggr, \ xfrgco .
\]

\[
\text{Access}(\ xu, \ authObj(S\_TCODE, \ TCD, \ ME54N)) \wedge
\]

\[
\text{Access}(\ xu, \ ME53N) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_BANF\_WRK, \ WERKS, \ xwrk)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_BANF\_BSA, \ BSART, \ xbsa)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_BANF\_EKG, \ EKGRP, \ xekg)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_EINK\_FRG, \ FRGGR, \ xfrggr)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_EINK\_FRG, \ FRGCO, \ xfrgco)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_BANF\_WRK, \ ACTVT, \ 02)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_BANF\_BSA, \ ACTVT, \ 02)) \wedge
\]

\[
\text{Access}(\ xu, \ authObj(M\_BANF\_EKG, \ ACTVT, \ 02)) \wedge
\]

\[
\supset
\text{Access}(\ xu, \ ME54N)
\]

Using the previously defined atoms it is now possible to model a release step. The atom \( \text{RequisitionReleasedStep} \) with arity 11 denotes a requisition item that
has been released with a certain release group and code. Similar to the abstraction of the create requisition transaction, the single authorization checks which check concrete variables occur again in the following formula. This is because the values of these variables are not known at the time of the abstraction. Here it is known, namely, the values belonging to the requisition that is going to be released.

\[\forall xu_1, xu_2, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt, xfrgstrat, xfrggr, xfrgco, xcl .
RequisitionCreated(xu_1, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \land
ReleaseStrategy(xfrgstrat, xfrggr, \text{class}(xcl, \text{property}(\text{FRG}_\text{CEBAN}_\text{EKGRP}, xekg))) \land
ReleaseStrategy(xfrgstrat, xfrggr, \text{class}(xcl, \text{property}(\text{FRG}_\text{CEBAN}_\text{WERKS}, xwrk))) \land
ReleaseStrategy(xfrgstrat, xfrggr, \text{class}(xcl, \text{property}(\text{FRG}_\text{CEBAN}_\text{GSWRT}, xgswrt))) \land
ReleaseRequirement(xfrgstrat, xfrggr, xfrgco) \land
Access(xu_2, \text{authObj}(\text{M}_\text{EINK}_\text{FRG}, \text{FRGGR}, xfrggr)) \land
Access(xu_2, \text{authObj}(\text{M}_\text{EINK}_\text{FRG}, \text{FRGCO}, xfrgco)) \land
Access(xu_2, \text{ME54N}) \land
Access(xu_2, \text{authObj}(\text{M}_\text{BANF}_\text{WRK}, \text{WERKS}, xwrk)) \land
Access(xu_2, \text{authObj}(\text{M}_\text{BANF}_\text{BSA}, \text{BSART}, xbsa)) \land
Access(xu_2, \text{authObj}(\text{M}_\text{BANF}_\text{EKG}, \text{EKGRP}, xekg))
\supset
RequisitionReleasedStep(xu_2, xfrggr, xfrgstrat, xfrgco, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt)\]

As mentioned, a release of a requisition typically consists of one or more release steps. In the R/3 test system of the Max-Planck Society there are two release strategies KF and VF. The former strategy represents the cost center release and consists of one step (with the code W1). The latter strategy denotes the director release which consists of two steps (codes W1 and W2). Both strategies belong to the release group 01. The steps required for each release strategy are modeled by the following transition. The resulting state RequisitionReleased is valid, if all required release steps have been succeeded.

\[\forall xu_1, xu_2, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt, xfrgstrat, xfrggr, xfrgco, xcl .
RequisitionReleasedStep(xu_2, 01, KF, W1, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \lor
(RequisitionReleasedStep(xu_1, 01, VF, W1, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \land
RequisitionReleasedStep(xu_2, 01, VF, W2, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt))
\supset
RequisitionReleased(xu_2, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt)\]

If none of the available release strategies applies to a requisition, then it is immediately released and ready to order. A release strategy doesn’t apply if at least
one of the properties defined in the strategy doesn’t match. So far, the theory only contains assertions that say that a certain property matches. In this case, however, it is required to verify whether a property does not match. For that reason, the theory must be extended with formulae for each possible values that define when a release strategy does not apply. The following transition shows the transition for the plant property and its value INFO. It means, that if the value INFO matches the plant property, then, of course, the other (possible) values SOFS and MPG do not match. Similar formulae are required for each element of the Cartesian product of property and possible value.

\[ \forall xfrstrat, xfrggr .
   \text{ReleaseStrategy}(xfrstrat, xfrggr, \text{class(FRG\_EBAN, property(FRG\_CEBAN\_WERKS, INFO)))} \]

\[ \supset
   \neg \text{ReleaseStrategy}(xfrstrat, xfrggr, \text{class(FRG\_EBAN, property(FRG\_CEBAN\_WERKS, SOFS)))} \land
   \neg \text{ReleaseStrategy}(xfrstrat, xfrggr, \text{class(FRG\_EBAN, property(FRG\_CEBAN\_WERKS, MPG)))} \]

The ability to decide when a property does not match leads to the following transition. It models, that no release strategy (here neither KF nor VF) applies. Therefore, the requisition is automatically released and ready for an order.

\[ \forall xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt, xfrstrat, xfrggr, xfrgco, xcl .
   \text{RequisitionCreated}(xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \land
   \neg \text{ReleaseStrategy}(\text{KF, xfrggr, class(FRG\_EBAN, property(FRG\_CEBAN\_EKGRP, xekg)))} \lor
   \neg \text{ReleaseStrategy}(\text{KF, xfrggr, class(FRG\_EBAN, property(FRG\_CEBAN\_WERKS, xwrk)))} \lor
   \neg \text{ReleaseStrategy}(\text{KF, xfrggr, class(FRG\_EBAN, property(FRG\_CEBAN\_GSWRT, xgswrt)))} \land
   \neg \text{ReleaseStrategy}(\text{VF, xfrggr, class(FRG\_EBAN, property(FRG\_CEBAN\_EKGRP, xekg)))} \lor
   \neg \text{ReleaseStrategy}(\text{VF, xfrggr, class(FRG\_EBAN, property(FRG\_CEBAN\_WERKS, xwrk)))} \lor
   \neg \text{ReleaseStrategy}(\text{VF, xfrggr, class(FRG\_EBAN, property(FRG\_CEBAN\_GSWRT, xgswrt)))} \]

\[ \supset
   \text{RequisitionReleased}(xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \]

5.4.3 Create an Order

A successful released requisition is the precondition to create an order object which is connected to the requisition in SAP R/3. The following transition models this business step. Firstly, the existence of the released requisition is checked. It has been released by the user xu1. Then the access to the transaction ME21N is checked because it is required to create order objects. Similar to the creation of a requisition a user authorized to create is typically also authorized to change and view orders. Therefore, the access to ME22N and ME23N is checked as well. The last checks ensure that the user is authorized to create orders for the plant, purchase document type,
purchasing group and purchasing organization respectively which is specified by 
the released requisition. All checks must succeed to enter the state *OrderCreated*.
Note, that the user who has released the requisition and the user who creates the 
order object can be the same person in this transition.

∀ \(xu1, xu2, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt \).
\(\text{RequisitionReleased}(xu1, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \land \newline \text{Access}(xu2, \text{ME21N}) \land \newline \text{Access}(xu2, \text{ME22N}) \land \newline \text{Access}(xu2, \text{ME23N}) \land \newline \text{Access}(xu2, \text{authObj(M\_BEST\_WRK, WERKS, xwrk)}) \land \newline \text{Access}(xu2, \text{authObj(M\_BANF\_BSA, BSART, xbsa)}) \land \newline \text{Access}(xu2, \text{authObj(M\_BEST\_BSA, BSART, xbsa)}) \land \newline \text{Access}(xu2, \text{authObj(M\_BEST\_EKG, EKGRP, xekg)}) \land \newline \text{Access}(xu2, \text{authObj(M\_BEST\_EKO, EKORG, xekorg)}) \land \newline \supset \newline \text{OrderCreated}(xu2, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \land \newline \neg \text{RequisitionCreated}(xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) \land \newline \neg \text{RequisitionReleased}(xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt)

5.5 Business Policies

The business policies are expressed by business people in a business terminology.
They impact the way of performing the business processes. For example, they
prohibit the possibility for a user to perform all process steps alone. The connection between the business process (steps) and the SAP R/3 system is quite simple – there is a mapping which maps a number of transactions to each process step. Since, the theory – which has been presented in the previous sections – already represents the process steps and the mapping to the individual transactions, it is easy to model business policies.

The general policy from section 2.4 prohibits that the complete process consisting of the steps “create a requisition”, “release a requisition” and “create the order” for this requisition can be performed by one single user for one concrete plant, material group, purchasing group and organization. The following formula represents this policy: there is no requisition such that a user \( x_u \) can reach all states \( \text{RequisitionCreated} \), \( \text{RequisitionReleased} \) and \( \text{OrderCreated} \). Of course, the assumption for such a policy are unique data in the SAP R/3 system.

\[
\neg \exists x_u, x_{bsa}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, x_{gswrt}.
\forall x_{pos}, x_{mat}.
\text{Requisition}(x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, x_{gswrt})
\supset
\text{RequisitionCreated}(x_u, x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, x_{gswrt}) \land
\text{RequisitionReleased}(x_u, x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, x_{gswrt}) \land
\text{OrderCreated}(x_u, x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, x_{gswrt})
\]

The policy for the direct office equipment purchase is that the creation of orders is only allowed for office equipment material and with a limit on the monetary amount. This policy is modeled by the following transition.

\[
\neg \exists x_u, x_{bsa}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}.
\forall x_{pos}, x_{mat}.
\text{Requisition}(x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, \text{GREATER}_10000\_\text{EUR})
\supset
\text{OrderCreated}(x_u, x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, \text{GREATER}_10000\_\text{EUR}) \land
\neg \text{RequisitionCreated}(x_u, x_{bsa}, x_{pos}, x_{mat}, x_{wrk}, x_{ekg}, x_{ekorg}, x_{matkl}, \text{GREATER}_10000\_\text{EUR})
\]

Since the money amounts are expressed as constants in the theory, it is required to have multiple transitions for every interval of money amounts. For example, if there are the intervals \( \text{LESS}_1000\_\text{EUR}, \text{GREATER}_10000\_\text{EUR} \) and \( \text{GREATER}_10000\_\text{EUR} \) and the creation of office material is only allowed for monetary amounts below 1000 EUR, then an additional transition is necessary. The additional transitions are all equal to the previous one except that the constant is different. Continuing the example, the new transition would have \( \text{GREATER}_10000\_\text{LESS}_10000\_\text{EUR} \).
Using the Theory

This chapter is about the analysis of the theory which has been developed in the previous chapter. On one hand, it will show that our formalization in fact maps the behavior and the authorization setup of an SAP R/3 system. On the other hand, it will show that it is possible to verify whether there are any errors in an SAP R/3 authorization setup with respect to the business policies using the theory. The proofs will be done using the tool SPASS.

Furthermore, I have performed different experiments which show that the proof of different conjectures is possible using the constructed theory.

The complete source used for the plausibility checks and the experiments which are described in this chapter can be found on the CD attached to this thesis.

6.1 Theorem Proving with SPASS

The automatic proving of the constructed theory is done using the tool SPASS (Synergetic Prover Augmenting Superposition with Sorts). SPASS is an automated theorem prover for first-order logic with equality. It is based on the superposition calculus from Bachmair and Ganzinger [BG94] and it has first been published by Christoph Weidenbach et al. [WGR96] in 1996. I used the version SPASS 3.0 which has been released in 2007 by Weidenbach et al. [WSH+07] for the plausibility checks and experiments. Internally, SPASS uses the tool FLOTTER to translate formulae into clause normal form. SPASS bases on a number of theoretical foundations which are described in [NRW98], [WAB+99], [WBH+02], [WSH+07] and [NW01].

All computations with SPASS were performed on a DELL PowerEdge 2650 with two 2.4 GHz processors and 4GB of memory running Debian Linux, Kernel 2.6.16.52.1.p4-smp.

SPASS uses text files in DFG-Syntax for input. The DFG language is intended to be a common exchange format for first-order problem settings. A SPASS input file consists of three parts:

1. Symbols: The symbol part defines the function and predicate symbols and their arity.
2. Axioms: Axioms are formulae in prefix notation. This part holds the mapping of the SAP R/3 system description, the authorization setup and the business policies.

3. Conjectures: Conjectures are statements which should be proved using SPASS. The transformation of the formulae of the theory which has been presented in Chapter 5 into formulae in DFG-Syntax is trivial because it is just a matter of syntax change. The transformed formulae will be placed in the axiom part. Additionally, the input file for SPASS requires the definition of the function and predicate symbols which have been used by the transformed formulae in the axiom part.

The result after invoking SPASS on an input file (in the case that it terminates) is either a proof or a saturated clause set. A set of clauses is called saturated if all possible inferences between the clauses have been done.

A proof indicates that there is either a contradiction in the axiom part or that the conjecture could be proven, assuming that a conjecture has been specified. If SPASS could find a proof, it lists all steps which have lead to the proof. This is useful in order to explore the plausibility of the formulae of our theory.

The termination of SPASS without a proof results in the output of the saturated clause set. This set contains a number of ground atoms. They can be used in order to construct a model (of expressions).

6.2 Plausibility Check

The theory has been developed on the basis of the purchase process. The plausibility of the typical use cases will be tested by the creation of a purchase requisition, the release of the requisition and the creation of an order. The plausibility check should confirm that the appropriate formulae of the theory correctly map the behavior of the real SAP R/3 system. The basis for the plausibility check and the experiments with SPASS is the theory which has been described in Chapter 5 together with the following authorization setup.

The authorization setup used for the experiments presented in this chapter is depicted in Fig. 20. The user “MUELLER” is authorized to create requisitions for the Max-Plank Institute for Informatics within the purchase group I26. Similarly, the user “KAISER” holds the authorizations to release requisitions for the previous mentioned plant and the purchase group with the release code W1. The user “MEIER” is authorized to do the final release for the requisitions with the release code W2. The users “LAMOTTE” and “SCHMITT” hold authorizations to create direct office equipment orders and arbitrary orders, respectively. There are different roles which are used to assign the previous mentioned authorizations to the appropriate users. The role names are are depicted in the Figure in the boxes above the blue user symbols. Basing on the role definitions, the auto-generated profiles which are assigned automatically to the users eventually combine the effective authorizations which grant the access to the appropriate purchase transactions in SAP R/3.
The plausibility checks which are described in this thesis assume that the purchase process will be performed for the plant Max-Planck Institute for Informatics (INFO) and all users mentioned before are members of the purchase group I26. This is necessary because the plausibility check of all formulae is only possible using concrete values for the requisition and the order.

Fig. 20. Authorization Setup Scenario

In more detail, the single role named ZLAMOTTE_BANF_INFO which is shown in Fig. 20 groups those authorizations which are required to create requisitions with the purchasing group ‘I26’ and the plant Max-Planck Institute for Informatics. The role is assigned to the user “MUELLER”. This automatically implies that the auto-generated authorization profile (containing the authorizations) based on the single role is assigned to the user, too.

The single role ZLAMOTTE_REL_INFO_BASE groups all basic authorizations except the release code(s) which are required to auto-generate the authorization profile granting the access to the release transaction. The release codes are separately defined in the two different single roles ZLAMOTTE_RELEASE_INFO_W1 for the code W1 and ZLAMOTTE_RELEASE_INFO_W2 for the code W2. The combination of the basic release authorizations and the appropriate release code authorization to a complete role has been achieved using composite roles (the Figure shows the single roles which are the elements of the composite roles). The composite role comprising the authorization with the code W1 is eventually assigned to the user “KAISER” (he/she is the group leader) and the one comprising the authorization with the code W2 is assigned to the user “MEIER” (he/she is the director). These assignments again imply the assignment of the auto-generated authorization profiles to the appropriate users.
Finally, the single role `ZLAMOTTE_ORDER_INFO_BASE` groups the basic authorizations required for the auto-generated authorization profile granting the access to create orders. The authorizations in this role don’t yet include the type of the material and the total allowed monetary amount. These two authorizations are defined separately in the single role `ZLAMOTTE_ORDER_INFO_ALL` allowing all kind of material of any monetary amount while the single role `ZLAMOTTE_ORDER_INFO_OFFICE` restricts the material type to office equipment and the monetary amount to ≤ 1000 EUR. The combination of the basic authorizations and the additional role is again done using composite roles. After the assignment of the roles to the users “LAMOTTE” and “SCHMITT”, respectively, the user “LAMOTTE” will be authorized to create office equipment orders while the user “SCHMITT” will be authorized to create arbitrary orders. However, any arbitrary order performed by the user “SCHMITT” requires a released requisition. As usual, the assignment of the role to the user again implies the assignment of the appropriate auto-generated authorization profile to that user. This finishes the formalization of the authorization setup and allows to run the plausibility checks.

SPASS has recognized the formulae representing the theory and the authorization setup described in Section 6.2 as a first-order Non-Horn problem without equality. Non-Horn means, that there are clauses in the model which have more than one positive literal. The theory presented in this thesis doesn’t make use of equality.

All checks presented in this thesis are numbered and the tests as well as the theorem prover SPASS itself are available on the CD attached to this thesis.

The first test for plausibility is the check whether there is a contradiction in the set of formulae. Since the theory contains also the business policies, a contradiction would indicate that one of the business rules has been violated. At this point, the conjecture part of the SPASS input file is still empty (Test 0).

**Fact 1.** SPASS (finitely) saturates the formulae representing the theory from chapter 5 and the authorization setup described in Section 6.2 in about 20 seconds.

**Fact 2.** Using a more efficient selection strategy (Option: -Select=2) for the clause selection, SPASS saturates the set of formulae from Fact 1 in less than 1 second.

The saturation of the formulae indicates that there is no contradiction in the set of formulae representing the theory. Therefore, none of the business policies has been violated.

Each of the following checks contains a statement which should be proved using SPASS. The statements are placed in the conjecture part of the input file.

The following test checks whether the auto-generation of the user profile works as expected when a single or composite role will be assigned to a user. If a composite role has been assigned to a user then all authorizations must be present in the authorization profile. This is formalized by the following conjecture 1.
∀ xu, xcrn, xsrn, xaon, xaof, xav.
    CompositeRole(compositeRoleEntry(xcrn, xsrn)) ∧
    SingleRole(singleRoleEntry(xsrn, authObj(xaon, xaof, xav))) ∧
    Holds(xu, xcrn) ∧
    ⊃
    UserProfile(xu, authObj(xaon, xaof, xav))

(1)

Fact 3. SPASS proves the conjecture 1 in less than 1 second.

The checks for single roles and the manual authorization profiles have been accomplished analogously.

All further plausibility checks refer to the steps of the purchase process. The following check concerns the creation of a requisition. If an arbitrary user holds the authorizations to create requisitions and there is a requisition which needs to be created in SAP R/3 then the requisition will be created by the user xu. The conjecture 2 formalizes this statement. The user xu has the authorizations in his/her user profile and consequently he/she is authorized to create the requisition.

∀ xu, xpos, xmat, xbsa, xwrk, xekg, xekorg, xmatkl, xgswrt.
    Requisition(xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt) ∧
    UserProfile(userprofileEntry(xu, authObj(S_TCODE, TCD, ME51N))) ∧
    UserProfile(userprofileEntry(xu, authObj(M_BANF_WRK, ACTVT, 01))) ∧
    UserProfile(userprofileEntry(xu, authObj(M_BANF_WRK, WERKS, xwrk))) ∧
    UserProfile(userprofileEntry(xu, authObj(M_BANF_BSA, ACTVT, 01))) ∧
    UserProfile(userprofileEntry(xu, authObj(M_BANF_BSA, BSART, xbsa))) ∧
    UserProfile(userprofileEntry(xu, authObj(M_BANF_EKG, ACTVT, 01))) ∧
    UserProfile(userprofileEntry(xu, authObj(M_BANF_EKG, EKGRP, xekg))) ∧
    Access(xu, authObj(S_TCODE, TCD, ME51N)) ∧
    Access(xu, authObj(M_BANF_WRK, ACTVT, 01)) ∧
    Access(xu, authObj(M_BANF_WRK, WERKS, xwrk)) ∧
    Access(xu, authObj(M_BANF_BSA, ACTVT, 01)) ∧
    Access(xu, authObj(M_BANF_BSA, BSART, xbsa)) ∧
    Access(xu, authObj(M_BANF_EKG, ACTVT, 01)) ∧
    Access(xu, authObj(M_BANF_EKG, EKGRP, xekg))
    ⊃
    RequisitionCreated(xu, xbsa, xpos, xmat, xwrk, xekg, xekorg, xmatkl, xgswrt)

(2)

If SPASS finds a proof for this conjecture then the mapping of the create requisition process step is correct.

Fact 4. SPASS proves the conjecture 2 in less than 1 second.

The next step in the purchase process is the release of the requisition. The requisition is either immediately released because no release strategy has been applied or the requisition needs to be released according to the applied strategy. As
explained, a release strategy applies if all conditions defined by the strategy are satisfied. Assume that the monetary amount of a requisition is \(< 1000\) EUR. In this example, neither KF nor the strategy VF will apply. Consequently, the requisition is immediately released and ready for order. The following conjecture\(^3\) formalizes this question using the plant INFO and the purchasing group I26 as example values for the requisition. SPASS should end with a proof because the requisition is immediately released due to the low monetary amount of \(< 1000\) EUR.

\[
\forall xu, xpos, xmat, xbsa, xekorg, xmatkl, xclass .
Requisition(xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, < 1000) \land
\neg ReleaseStrategy(VF, 01, class(xclass, property(FRG_CEBAN_EKGRP, I26))) \land
\neg ReleaseStrategy(VF, 01, class(xclass, property(FRG_CEBAN_WERKS, INFO))) \land
\neg ReleaseStrategy(VF, 01, class(xclass, property(FRG_CEBAN_GSWRT, < 1000))) \land
\neg ReleaseStrategy(KF, 01, class(xclass, property(FRG_CEBAN_EKGRP, I26))) \land
\neg ReleaseStrategy(KF, 01, class(xclass, property(FRG_CEBAN_WERKS, INFO))) \land
\neg ReleaseStrategy(KF, 01, class(xclass, property(FRG_CEBAN_GSWRT, < 1000))) \\
\subseteq
RequisitionReleased(xu, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, < 1000) \land
\tag{3}
\]

**Fact 5.** SPASS proves the conjecture\(^3\) in less than 1 second.

The next checks test the plausibility of the release steps. The release strategy VF requires two single steps, one with the code \(W1\) and one with \(W2\). The release steps have been successful if both RequisitionReleasedStep() predicates of conjecture\(^4\) are valid. Each of the RequisitionReleasedStep() predicates represents the release with the appropriate code. A further condition for the validity of the conjecture is that the persons who execute the release steps must be distinct from each other. This is formalized using three different variables \(xu1, xu2\) and \(xu3\) for the users. Of course, a release can only be performed by a user if he/she has the required authorizations for (i) the appropriate release step (the right release code) and (ii) executing the release transaction. The bunch of single authorizations which are required to perform the release transaction is abbreviated using the atoms Access(xu2, ME54N) and Access(xu3, ME54N). The user authorization profiles for the users \(xu2\) and \(xu3\) are also abbreviated because of readability reasons. They contain all authorizations to execute the release transaction and to release with the code \(W1\) and with \(W2\), respectively.
∀ xu1, xu2, xu3, xpos, xmat, xfrgstrat, xfrggr, xcl, xbsa, xekorg, xmatkl .
RequisitionCreated(xu1, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000) ∧
ReleaseStrategy(VF, 01, class(xclass, property(FRG_CEBAN_EKGRP, I26))) ∧
ReleaseStrategy(VF, 01, class(xclass, property(FRG_CEBAN_WERKS, INFO))) ∧
ReleaseStrategy(VF, 01, class(xclass, property(FRG_CEBAN_GSWRT, > 10000))) ∧
ReleaseRequirement(VF, 01, W1) ∧
ReleaseRequirement(VF, 01, W2) ∧
Access(xu2, authObj(M_EINK_FRG, FRGGR, 01)) ∧
Access(xu2, authObj(M_EINK_FRG, FRGCO, W1)) ∧
Access(xu2, ME54N) ∧
Access(xu2, authObj(M_BANF_WRK, WERKS, INFO)) ∧
Access(xu2, authObj(M_BANF_BSA, BSART, xbsa)) ∧
Access(xu2, authObj(M_BANF_EKG, EKGRP, I26)) ∧
Userprofile(userprofileEntry(...)) ∧
Access(xu3, authObj(M_EINK_FRG, FRGGR, 01)) ∧
Access(xu3, authObj(M_EINK_FRG, FRGCO, W2)) ∧
Access(xu3, ME54N) ∧
Access(xu3, authObj(M_BANF_WRK, WERKS, INFO)) ∧
Access(xu3, authObj(M_BANF_BSA, BSART, xbsa)) ∧
Userprofile(userprofileEntry(...)) ∧
□
RequisitionReleasedStep(xu2, 01, W1, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000) ∧
RequisitionReleasedStep(xu3, 01, W2, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000) 

(4)

Fact 6. SPASS proves the conjecture 4 in 1.808 seconds.

To be sure that the users performing the release steps as well as the one who created the requisition are really distinct from each other I tested the previous conjecture with the following changes. First, I used the atom RequisitionReleasedStep(xu1,...) which shouldn’t be valid because the person creating the requisition is equal to the one who performed the first release step (Test 4a). SPASS terminates on saturating the theory which is the expected result since the user xu1 doesn’t hold the authorizations to execute the release transaction. Similarly, the combination RequisitionReleasedStep(xu2, 01, W1, ...) and RequisitionReleasedStep(xu2, 01, W2, ...) (both release steps can be executed by the same person) has been checked (Test 4b). The expected result for the different modifications mentioned before is that SPASS terminates without finding a proof for each run:

Fact 7. Using the option -Select=2, SPASS terminates on saturating the theory for each of the modified conjectures saying that the persons involved in the creation of a requisition and the release must be distinct to each other in less than 1 second. Without the more efficient selection strategy option, SPASS terminates for each conjecture after around 30 seconds.
The user performing the last release step (W2) is also the one who does the final release of the requisition. Therefore, SPASS should end with a proof for the following conjecture\[5\]:

\[
\forall \, xu1, \, xu2, \, xu3, \, xpos, \, xmat, \, xbsa, \, xekorg, \, xmatkl.
\]
\[
\text{Requisition}(xbsa, \, xpos, \, xmat, \, INFO, \, I26, \, xekory, \, xmatkl, > 10000) \land
\text{RequisitionCreated}(xu1, \, xbsa, \, xpos, \, xmat, \, INFO, \, I26, \, xekory, \, xmatkl, > 10000) \land
\text{RequisitionReleasedStep}(xu2, \, 01, \, W1, \, xbsa, \, xpos, \, xmat, \, INFO, \, I26, \, xekory, \, xmatkl, \, > 10000) \land
\text{ReleaseStrategy}(\text{VF}, \, 01, \, \text{class}(xclass, \, \text{property}(\text{FRG\_CEBAN\_EKG\_R}, \, \text{I26}))) \land
\text{ReleaseStrategy}(\text{VF}, \, 01, \, \text{class}(xclass, \, \text{property}(\text{FRG\_CEBAN\_WERKS}, \, \text{INFO}))) \land
\text{ReleaseStrategy}(\text{VF}, \, 01, \, \text{class}(xclass, \, \text{property}(\text{FRG\_CEBAN\_GSWRT}, \, > 10000))) \land
\text{ReleaseRequirement}(\text{VF}, \, 01, \, W1) \land
\text{ReleaseRequirement}(\text{VF}, \, 01, \, W2) \land
\text{Access}(xu3, \, \text{authObj}(\text{M\_EINK\_FRG}, \, \text{FRG\_GR}, \, 01)) \land
\text{Access}(xu3, \, \text{authObj}(\text{M\_EINK\_FRG}, \, \text{FRG\_CO}, \, W2)) \land
\text{Access}(xu3, \, \text{ME54N}) \land
\text{Access}(xu3, \, \text{authObj}(\text{M\_BANF\_WRK}, \, \text{WERKS}, \, \text{INFO})) \land
\text{Access}(xu3, \, \text{authObj}(\text{M\_BANF\_BSA}, \, \text{BSART}, \, xbsa)) \land
\text{Access}(xu3, \, \text{authObj}(\text{M\_BANF\_EKG}, \, \text{EKGRP}, \, I26)) \land
\text{Userprofile}(\text{userprofileEntry}(\ldots)) \land
\text{RequisitionReleasedStep}(xu3, \, 01, \, W2, \, xbsa, \, xpos, \, xmat, \, INFO, \, I26, \, xekory, \, xmatkl, \, > 10000) \land
\top
\]
\[
\text{RequisitionReleased}(xu3, \, xbsa, \, xpos, \, xmat, \, INFO, \, I26, \, xekory, \, xmatkl, \, > 10000)
\]

\[5\]

**Fact 8.** SPASS proves the conjecture\[5\] in less than 1 second.

The last step of the standard purchase process is the creation of the order which requires a released requisition. If there is a released requisition (created by user \(xu1\) and released by user \(xu2\)) and there is a user \(xu3\) who holds (i) the authorization to create orders and (ii) he/she is distinct from the one who created the requisition and the one who released the requisition then the order will be created. He/She has the required authorizations in the user profile and consequently he/she has the access to the create order transaction. The formula mapping the creation of the order is valid if SPASS can prove the conjectures\[6\].
∀ xu1, xu2, xu3, xpos, xmat, xbsa, xekorg, xmatkl.
Requisition(xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000) ∧
RequisitionCreated(xu1, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000) ∧
RequisitionReleased(xu2, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000) ∧
Access(xu3, authObj(S_TCODE, TCD, ME21N)) ∧
Access(xu3, authObj(M_BEST_EKO, ACTVT, 01)) ∧
Access(xu3, authObj(M_BEST_EKO, ACTVT, 09)) ∧
Access(xu3, authObj(M_BEST_EKO, EKORG, xekorg)) ∧
Access(xu3, authObj(M_BEST_BSA, ACTVT, 01)) ∧
Access(xu3, authObj(M_BEST_BSA, ACTVT, 09)) ∧
Access(xu3, authObj(M_BEST_BSA, BSART, xbsa)) ∧
Access(xu3, authObj(M_BANF_BSA, ACTVT, 01)) ∧
Access(xu3, authObj(M_BANF_BSA, BSART, xbsa)) ∧
Access(xu3, authObj(M_BEST_WRK, ACTVT, 01)) ∧
Access(xu3, authObj(M_BEST_WRK, ACTVT, 08)) ∧
Access(xu3, authObj(M_BEST_WRK, ACTVT, 09)) ∧
Access(xu3, authObj(M_BEST_WRK, WERKS, INFO)) ∧
Access(xu3, authObj(M_BEST_EKG, ACTVT, 01)) ∧
Access(xu3, authObj(M_BEST_EKG, ACTVT, 09)) ∧
Access(xu3, authObj(M_BEST_EKG, EKGRP, I26)) ∧
Access(xu3, authObj(M_EINK_MKL, MATKL, xmatkl)) ∧
Access(xu3, authObj(M_EINK_GWT, GSWRT, > 10000)) ∧
Userprofile(userprofileEntry(...))) ⊃
OrderCreated(xu3, xbsa, xpos, xmat, INFO, I26, xekorg, xmatkl, > 10000)

Fact 9. SPASS proves the conjecture 6 in around 2 seconds.

The special case which is also included in this thesis is the possibility to create direct orders for office equipment. This kind of orders has a limit on the monetary amount and is restricted to office material. If the formalization of the special case has been done correctly, SPASS should find a proof for the conjecture 7. Whenever a requisition for office equipment (represented with the constant B0002) with a monetary amount below or equal to 1000 EUR needs to be created and there is a user holding the authorizations to create direct office orders, then the order will be directly created. The user holding the authorizations is just schematized by the atom Userprofile() in the following conjecture.
∀ xu, xpos, xmat, xbsa, xekory .
Requisition(xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000) ∧
Access(xu, authObj(M_EINK_MKL, MATKL, B0002)) ∧
Access(xu, authObj(M_EINK_GWT, GSWRT, ≤ 1000)) ∧
Access(xu, authObj(S_TCODE, TCD, ME21N)) ∧
Access(xu, ME21N) ∧
Access(xu, authObj(M_BEST_WRK, WERKS, INFO)) ∧
Access(xu, authObj(M_BANF_BSA, BSART, xbsa)) ∧
Access(xu, authObj(M_BEST_BSA, BSART, xbsa)) ∧
Access(xu, authObj(M_BEST_EKG, EKGRP, I26)) ∧
Access(xu, authObj(M_BEST_EKO, EKORG, xekory)) ∧
Userprofile(userprofileEntry(...) )

\[ \supset \]
OrderCreated(xu, xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000) ∧
¬RequisitionCreated(xu, xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000) ∧
¬RequisitionReleased(xu, xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000)

Fact 10. Spass proves the conjecture \[ \textcolor{red}{\checkmark} \] in less than 1 second.

In contrast, the direct creation of office material orders will fail because at least one of the required authorizations is missing in the users’ authorization profile (in the following conjecture the authorization to access the special transaction code ME21NOFF is missing for the user “SCHMITT”). In this case, Spass should end with the saturation of the model.

∀ xpos, xmat, xbsa, xekory .
Requisition(xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000) ∧
Access(SCHMITT, authObj(M_EINK_MKL, MATKL, B0002)) ∧
Access(SCHMITT, authObj(M_EINK_GWT, GSWRT, ≤ 1000)) ∧
Access(SCHMITT, authObj(S_TCODE, TCD, ME21N)) ∧
Access(SCHMITT, ME21N) ∧
Access(SCHMITT, authObj(M_BEST_WRK, WERKS, INFO)) ∧
Access(SCHMITT, authObj(M_BANF_BSA, BSART, xbsa)) ∧
Access(SCHMITT, authObj(M_BEST_BSA, BSART, xbsa)) ∧
Access(SCHMITT, authObj(M_BEST_EKG, EKGRP, I26)) ∧
Access(SCHMITT, authObj(M_BEST_EKO, EKORG, xekory)) ∧
Userprofile(userprofileEntry(...) )

\[ \supset \]
OrderCreated(SCHMITT, xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000) ∧
¬RequisitionCreated(SCHMITT, xbsa, xpos, xmat, INFO, I26, xekory, xmatKL, ≤ 1000) ∧
¬RequisitionReleased(SCHMITT, xbsa, xpos, xmat, INFO, I26, xekory, B0002, ≤ 1000)
Fact 11. Using the option `-Select=2`, Spass terminates on saturating the theory and the conjecture in less than 1 second, without the select option the saturation takes 30.779 seconds.

6.3 Experiments

This section describes the experiments I have done in order to prove some typical conjectures. Furthermore, I have changed the authorization setup in order to determine whether Spass is able to catch potential errors resulting from the change. At the end, the capacity of the formal model and in particular the limits of the automatic proofs with Spass will be presented.

Assume that the authorization setup from Section 6.2 has changed in the way that the user “SCHMITT” holds the authorizations to create direct office orders (role ZLAMOTTE_ORDER_INFO_OFFICE) additionally to the default buyer authorizations he has hold before. The idea of this change is that the user “SCHMITT” should be authorized to (i) create arbitrary orders which require a released requisition and (ii) create direct orders for office equipment without a requisition. The change is achieved by the assignment of the role ZLAMOTTE_ORDER_INFO_OFFICE to the user “SCHMITT”. An open question at this time is whether the result of this change in the authorization setup possibly contradicts the business policies. The changed setup is passed to Spass without any conjectures in order to find an answer to the previous question.

Scenario Direct Office Equipment

Fact 12. Spass found a proof for the theory including the authorization setup change for the scenario in less than 1 second.

Spass provides a proof which means that there are formulae in the theory which contradict the business policies. Furthermore, the proof output shows exactly the position where the conflict occurs. In this case, it refers to the user “SCHMITT” who holds both the roles ZLAMOTTE_ORDER_INFO_ALL and ZLAMOTTE_ORDER_INFO_OFFICE.

The limit for the automatic proofs with Spass are conjectures which say that a user is not authorized to do something, for example, to create requisitions or orders. Assume, that the user “MUELLER” is not authorized to create orders because he/she is already the purchase requisitioner. When he/she is not authorized than at least one of the required authorizations to create orders is missing in his/her authorization profile. The result which is expected from Spass is that it should be able to proof conjectures like “Is the user MUELLER not authorized to create orders?”. However, in such cases Spass terminates providing just the saturated set of clauses describing the model instead of a proof.

The problem of such conjectures is that the clause set representing the model only contains clauses which grant the access to something. Spass tries to prove
an arbitrary conjecture by looking for a contradiction to the negated conjecture. The negated conjecture from the previous example is that the user “MUELLER” is authorized to create orders: he/she holds all authorizations which are required to create orders. As mentioned SPASS provides only the clauses describing a model where the user “MUELLER” is authorized to create orders. This is a valid model because all expressions which have been defined by the formulae of the theory are true in the interpretation. However, the intrinsic comprehension of the specification what a user is allowed to do in SAP R/3 (this is the meaning of the authorization setup) includes that the access is forbidden to everything for which he/she has not been authorized.

However, theorem proving with SPASS refers to any models but some conjectures are only provable using the minimal model. The minimal model is based on subset relations to a model $I$.

6.4 Results

The plausibility checks have shown that the formalization of the SAP R/3 system is correct and the formulae provide the expected result. The checks have been performed using the purchase process as a case study.

Furthermore, the experiments have shown that certain kind of conjectures could not be proved automatically using SPASS. The reason lies in the formalization which expresses only the actions which are permitted in SAP R/3. It doesn’t say anything about what action is prohibited. However, the intrinsic meaning of the authorization setup is that the access is forbidden to everything for which a user has not been authorized. Fortunately, this meaning is also represented by the minimal model. Therefore, conjectures which refer to the prohibition of some action can be proved using this special model but this future work.

Another problem is the representation of monetary amounts. The monetary amounts have been formalized in our theory using constants. Therefore, the implementation of a comparison of the amounts requires a lot of formulae. The problem of the representation of numbers and arithmetic is a current research topic. SPASS itself doesn’t provide any opportunity to use numbers and to do comparisons of numbers. A partial solution to this problem is the work of Prevosto and Waldmann [PW06]. They have developed an extension to SPASS called SPASS+T which provides the possibility to enlarge the reasoning capabilities of SPASS using an arbitrary SMT (Satisfiability Modulo Theories) procedure for arithmetic and free function symbols as a black-box. SMT is the problem of satisfiability of formulae with respect to background theories for which specialized decision procedures exist – such as, for instance, the theory of real numbers, of integers or linear arithmetic. The problem of the extension is that it is not complete. The termination of SPASS+T neither implies that a model for the theory exists nor that no model exists. Therefore, SPASS+T provides only a limited solution to our problem of the monetary amounts.
Conclusion and Outlook

In this thesis, I presented the possibility of an automatic verification of the authorizations in the SAP R/3 system with respect to given business policies. The automatic verification has been shown using the purchase process as a case study.

The first chapter has introduced into the SAP R/3 system and the objects involved to the authorizations concept. The understanding of a SAP transaction as well as the concept of authorization objects and authorizations has been presented. The second part describes the so called authorization setup. This setup holds the information about the authorizations of each user. Authorizations can be assigned either using (direct) authorization profiles or through roles. Both alternatives have been presented. The general description of business processes eventually lead to the purchase process which I have selected as an example. The second chapter describes the extraction of the authorization information related to the purchase process. It illustrates the way I used to determine the different authorization checks occurring during the execution of the purchase transactions to create a requisition, release the requisition and to create the order as well as the required authorizations for these transactions. The business policies are specified by business people in a business language and need to be extracted, too. These policies depend on the company and are not directly represented in the SAP R/3 system. Therefore, some typical policies concerning the purchase process have been presented.

To accomplish the automatic verification, the SAP R/3 authorization mechanism, the authorization setup and the business policies have been formalized in first-order logic using the purchase process as a case study. The resulting theory has been transformed to match the syntax requirements for the theorem prover SPASS. The last chapter shows first that the developed theory correctly maps the SAP R/3 system and the policies using SPASS. It further shows that the system is consistent with the business regulations. Different experiments demonstrate the ability of SPASS to find errors in the authorization setup.

However, the experiments have revealed different limitations. The first limitation is that SPASS is not able to prove every kind of conjecture. Conjectures asking whether something is forbidden can not be answered automatically by SPASS. Nevertheless, such proofs can be accomplished using the minimal model for a theory but this is future work.
Another problem is the representation of monetary amounts which are currently represented as constants in the theory. I have selected this representation because a mapping of numbers and arithmetic is not available for SPASS. In future, a solution approach could be the use of SPASS+T which is an extension for SPASS which provides the possibility using an arbitrary SMT procedure for arithmetic and free function symbols. The problem of this extension is the missing completeness. Therefore, SPASS+T will probably provide only a limited solution to the problem of mapping monetary amounts.

A further task for the future could be the proof of decidability for the theory. So far, I have just some clues that the theory is decidable but not a proof.
A

Directory Structure of the CD

This overview shows the directory structure of the CD which is attached to this thesis. This work has been created using \LaTeX.

/  
  |  
  |--> Thesis  
  |     |  
  |     |--> LaTeX  LaTeX Sources of the Thesis.  
  |     |     |  
  |     |--> PDF  PDF File of the Thesis.  
  |     |     |  
  |     |--> Tests  Input Files (.dfg) for SPASS.  
  |     |     |  
  |     |--> Talk  Slides and Text of the Presentation Talk.  
  |   
  |--> Programs  
  |     |  
  |     |--> TeXnicCenter  Tool to Edit LaTeX Source Files on Windows.  
  |     |     |  
  |     |--> MikTeX  Required to Compile Source Files.  
  |     |     |  
  |     |--> SPASS  Theorem Prover Tool.  
  |     |     |  
  |     |--> Adobe Reader  Free Tool for Reading PDF Files.  
  |  

References


PW06. *Prevosto, Virgile and Uwe Waldmann: SPASS+T.* In *Sutcliffe, Geoff, Renate Schmidt and Stephan Schulz (editors): EScRo: FLoC’06 Workshop on Empirically Successful Computerized...*
References


SAP01. SAP Library: Users and Roles (BC-CCM-USR), 2001.

SAP07. SAP AG: Third Quarter 2007 Results and Nine Month 2007 Results, 2007.


